

1

**1.1 Scope**

IEEE Std 1003.1-200x defines a standard operating system interface and environment, including a command interpreter (or “shell”), and common utility programs to support applications portability at the source code level. It is intended to be used by both applications developers and system implementors.

IEEE Std 1003.1-200x comprises four major components (each in an associated volume):

1. General terms, concepts, and interfaces common to all volumes of IEEE Std 1003.1-200x, including utility conventions and C language header definitions, are included in the Base Definitions volume of IEEE Std 1003.1-200x.
2. Definitions for system service functions and subroutines, language-specific system services for the C programming language, function issues, including portability, error handling, and error recovery, are included in the System Interfaces volume of IEEE Std 1003.1-200x.
3. Definitions for a standard source code-level interface to command interpretation services (a “shell”) and common utility programs for application programs are included in the Shell and Utilities volume of IEEE Std 1003.1-200x.
4. Extended rationale that did not fit well into the rest of the document structure, containing historical information concerning the contents of IEEE Std 1003.1-200x and why features were included or discarded by the standard developers, is included in the Rationale (Informative) volume of IEEE Std 1003.1-200x.

The following areas are outside of the scope of IEEE Std 1003.1-200x:

- Graphics interfaces
- Database management system interfaces
- Record I/O considerations
- Object or binary code portability
- System configuration and resource availability

IEEE Std 1003.1-200x describes the external characteristics and facilities that are of importance to applications developers, rather than the internal construction techniques employed to achieve these capabilities. Special emphasis is placed on those functions and facilities that are needed in a wide variety of commercial applications.

The facilities provided in IEEE Std 1003.1-200x are drawn from the following base documents:

- IEEE Std 1003.1-1996 (POSIX-1) (incorporating IEEE Stds. 1003.1-1990, 1003.1b-1993, 1003.1c-1995, and 1003.1i-1995)
- The following amendments to the POSIX.1-1990 standard:
  - IEEE P1003.1a draft standard (Additional System Services)
  - IEEE Std 1003.1d-1999 (Additional Realtime Extensions)

- 38 — IEEE Std 1003.1g-2000 (Protocol-Independent Interfaces (PII))
- 39 — IEEE Std 1003.1j-2000 (Advanced Realtime Extensions)
- 40 — IEEE Std 1003.1q-2000 (Tracing)
- 41 • IEEE Std 1003.2-1992 (POSIX-2) (includes IEEE Std 1003.2a-1992)
- 42 • The following amendments to the ISO POSIX-2: 1993 standard:
- 43 — IEEE P1003.2b draft standard (Additional Utilities)
- 44 — IEEE Std 1003.2d-1994 (Batch Environment)
- 45 • Open Group Technical Standard, February 1997, System Interface Definitions, Issue 5 (XBD5)
- 46 (ISBN: 1-85912-186-1, C605)
- 47 • Open Group Technical Standard, February 1997, Commands and Utilities, Issue 5 (XCU5)
- 48 (ISBN: 1-85912-191-8, C604)
- 49 • Open Group Technical Standard, February 1997, System Interfaces and Headers, Issue 5
- 50 (XSH5) (in 2 Volumes) (ISBN: 1-85912-181-0, C606)
- 51 **Note:** XBD5, XCU5, and XSH5 are collectively referred to as the *Base Specifications*.
- 52 • Open Group Technical Standard, January 2000, Networking Services, Issue 5.2 (XNS5.2)
- 53 (ISBN: 1-85912-241-8, C808)
- 54 • ISO/IEC 9899: 1999, Programming Languages — C.

55 IEEE Std 1003.1-200x uses the *Base Specifications* as its organizational basis and adds the  
56 following additional functionality to them drawn from the base documents above:

- 57 • Normative text from the ISO POSIX-1: 1996 standard and the ISO POSIX-2: 1993 standard not
- 58 included in the *Base Specifications*
- 59 • The amendments to the POSIX.1-1990 standard and the ISO POSIX-2: 1993 standard listed
- 60 above, except for parts of IEEE Std 1003.1g-2000
- 61 • Portability Considerations
- 62 • Additional rationale and notes

63 The following features, marked legacy or obsolescent in the base documents, are not carried  
64 forward into IEEE Std 1003.1-200x. Other features from the base documents marked legacy or  
65 obsolescent are carried forward unless otherwise noted.

66 From XSH5, the following legacy interfaces, headers, and external variables are not carried  
67 forward:

68 *advance()*, *brk()*, *chroot()*, *compile()*, *cuserid()*, *gamma()*, *getdtablesize()*, *getpagesize()*, *getpass()*,  
69 *getw()*, *putw()*, *re\_comp()*, *re\_exec()*, *regcmp()*, *sbrk()*, *sigstack()*, *step()*, *wait3()*, **<re\_comp.h>**,  
70 **<regexp.h>**, **<varargs.h>**, *loc1*, *\_\_loc1*, *loc2*, *locs*

71 From XCU5, the following legacy utilities are not carried forward:

72 *calendar*, *cancel*, *cc*, *col*, *cpio*, *cu*, *dircmp*, *dis*, *egrep*, *fgrep*, *line*, *lint*, *lpstat*, *mail*, *pack*, *pcat*, *pg*, *spell*,  
73 *sum*, *tar*, *unpack*, *uulog*, *uname*, *uupick*, *uuto*

74 In addition, legacy features within non-legacy reference pages (for example, headers) are not  
75 carried forward.

76 From the ISO POSIX-1:1996 standard, the following obsolescent features are not carried  
77 forward:

- 78 Page 112, CLK\_TCK  
79 Page 197 *tcgetattr()* rate returned option
- 80 From the ISO POSIX-2:1993 standard, obsolescent features within the following pages are not  
81 carried forward:
- 82 Page 75, zero-length prefix within PATH  
83 Page 156, 159 *set*  
84 Page 178, *awk*, use of no argument and no parentheses with length  
85 Page 259, *ed*  
86 Page 272, *env*  
87 Page 282, *find -perm[-]onum*  
88 Page 295-296, *egrep*  
89 Page 299-300, *head*  
90 Page 305-306, *join*  
91 Page 309-310, *kill*  
92 Page 431-433, 435-436, *sort*  
93 Page 444-445, *tail*  
94 Page 453, 455-456, *touch*  
95 Page 464-465, *tty*  
96 Page 472, *uniq*  
97 Page 515-516, *ex*  
98 Page 542-543, *expand*  
99 Page 563-565, *more*  
100 Page 574-576, *newgrp*  
101 Page 578, *nice*  
102 Page 594-596, *renice*  
103 Page 597-598, *split*  
104 Page 600-601, *strings*  
105 Page 624-625, *vi*  
106 Page 693, *lex*
- 107 The *c89* utility (which specified a compiler for the C Language specified by the  
108 ISO/IEC 9899:1990 standard) has been replaced by a *c99* utility (which specifies a compiler for  
109 the C Language specified by the ISO/IEC 9899:1999 standard).
- 110 From XSH5, text marked OH (Optional Header) has been reviewed on a case-by-case basis and |  
111 removed where appropriate. The XCU5 text marked OF (Output Format Incompletely Specified) |  
112 and UN (Possibly Unsupportable Feature) has been reviewed on a case-by-case basis and |  
113 removed where appropriate |
- 114 For the networking interfaces, the base document is the XNS, Issue 5.2 specification. The  
115 following parts of the XNS, Issue 5.2 specification are out of scope and not included in  
116 IEEE Std 1003.1-200x:
- 117 • Part 3 (XTI)
  - 118 • Part 4 (Appendixes)
- 119 Since there is much duplication between the XNS, Issue 5.2 specification and  
120 IEEE Std 1003.1g-2000, material only from the following sections of IEEE Std 1003.1g-2000 has  
121 been included:
- 122 • General terms related to sockets (2.2.2)
  - 123 • Socket concepts (5.1 through 5.3, inclusive)

- 124 • The *pselect()* function (6.2.2.1 and 6.2.3)
- 125 • The `<sys/select.h>` header (6.2)

126 Emphasis is placed on standardizing existing practice for existing users, with changes and  
 127 additions limited to correcting deficiencies in the following areas:

- 128 • Issues raised by IEEE or ISO/IEC Interpretations against IEEE Std 1003.1 and IEEE Std 1003.2
- 129 • Issues raised in corrigenda for the *Base Specifications* and working group resolutions from The  
 130 Open Group
- 131 • Corrigenda and resolutions passed by The Open Group for the XNS, Issue 5.2 specification
- 132 • Changes to make the text self-consistent with the additional material merged
- 133 • A reorganization of the options in order to facilitate profiling, both for smaller profiles such  
 134 as IEEE Std 1003.13, and larger profiles such as the Single UNIX Specification
- 135 • Alignment with the ISO/IEC 9899: 1999 standard

## 136 1.2 Conformance

137 Conformance requirements for IEEE Std 1003.1-200x are defined in Chapter 2 (on page 15).

## 138 1.3 Normative References

139 The following standards contain provisions which, through references in IEEE Std 1003.1-200x,  
 140 constitute provisions of IEEE Std 1003.1-200x. At the time of publication, the editions indicated  
 141 were valid. All standards are subject to revision, and parties to agreements based on  
 142 IEEE Std 1003.1-200x are encouraged to investigate the possibility of applying the most recent  
 143 editions of the standards listed below. Members of IEC and ISO maintain registers of currently  
 144 valid International Standards.

145 ANS X3.9-1978

146 (Reaffirmed 1989) American National Standard for Information Systems: Standard  
 147 X3.9-1978, Programming Language FORTRAN.<sup>1</sup>

148 ISO/IEC 646: 1991

149 ISO/IEC 646: 1991, Information Processing — ISO 7-bit Coded Character Set for Information  
 150 Interchange.<sup>2</sup>

151 The reference version of the standard contains 95 graphic characters, which are identical to  
 152 the graphic characters defined in the ASCII coded character set.

153 ISO 4217: 1995

154 ISO 4217: 1995, Codes for the Representation of Currencies and Funds. |

---

155

156 1. ANSI documents can be obtained from the Sales Department, American National Standards Institute, 1430 Broadway, New  
 157 York, NY 10018, U.S.A.

158 2. ISO/IEC documents can be obtained from the ISO office: 1 Rue de Varembe, Case Postale 56, CH-1211, Genève 20,  
 159 Switzerland/Suisse

160	ISO 8601:2000	
161	ISO 8601:2000, Data Elements and Interchange Formats — Information Interchange —	
162	Representation of Dates and Times.	
163	ISO C (1999)	
164	ISO/IEC 9899:1999, Programming Languages — C, including Technical Corrigendum No. 1.	
165	ISO/IEC 10646-1:2000	
166	ISO/IEC 10646-1:2000, Information Technology — Universal Multiple-Octet Coded	
167	Character Set (UCS) — Part 1: Architecture and Basic Multilingual Plane.	

## 168 1.4 Terminology

169 For the purposes of IEEE Std 1003.1-200x, the following terminology definitions apply:

### 170 **can**

171 Describes a permissible optional feature or behavior available to the user or application. The  
172 feature or behavior is mandatory for an implementation that conforms to  
173 IEEE Std 1003.1-200x. An application can rely on the existence of the feature or behavior.

### 174 **implementation-defined**

175 Describes a value or behavior that is not defined by IEEE Std 1003.1-200x but is selected by  
176 an implementor. The value or behavior may vary among implementations that conform to  
177 IEEE Std 1003.1-200x. An application should not rely on the existence of the value or  
178 behavior. An application that relies on such a value or behavior cannot be assured to be  
179 portable across conforming implementations.

180 The implementor shall document such a value or behavior so that it can be used correctly  
181 by an application.

### 182 **legacy**

183 Describes a feature or behavior that is being retained for compatibility with older  
184 applications, but which has limitations which make it inappropriate for developing portable  
185 applications. New applications should use alternative means of obtaining equivalent  
186 functionality.

### 187 **may**

188 Describes a feature or behavior that is optional for an implementation that conforms to  
189 IEEE Std 1003.1-200x. An application should not rely on the existence of the feature or  
190 behavior. An application that relies on such a feature or behavior cannot be assured to be  
191 portable across conforming implementations.

192 To avoid ambiguity, the opposite of *may* is expressed as *need not*, instead of *may not*.

### 193 **shall**

194 For an implementation that conforms to IEEE Std 1003.1-200x, describes a feature or  
195 behavior that is mandatory. An application can rely on the existence of the feature or  
196 behavior.

197 For an application or user, describes a behavior that is mandatory.

### 198 **should**

199 For an implementation that conforms to IEEE Std 1003.1-200x, describes a feature or  
200 behavior that is recommended but not mandatory. An application should not rely on the  
201 existence of the feature or behavior. An application that relies on such a feature or behavior  
202 cannot be assured to be portable across conforming implementations.

203 For an application, describes a feature or behavior that is recommended programming  
204 practice for optimum portability.

205 **undefined**

206 Describes the nature of a value or behavior not defined by IEEE Std 1003.1-200x which  
207 results from use of an invalid program construct or invalid data input.

208 The value or behavior may vary among implementations that conform to  
209 IEEE Std 1003.1-200x. An application should not rely on the existence or validity of the  
210 value or behavior. An application that relies on any particular value or behavior cannot be  
211 assured to be portable across conforming implementations.

212 **unspecified**

213 Describes the nature of a value or behavior not specified by IEEE Std 1003.1-200x which  
214 results from use of a valid program construct or valid data input.

215 The value or behavior may vary among implementations that conform to  
216 IEEE Std 1003.1-200x. An application should not rely on the existence or validity of the  
217 value or behavior. An application that relies on any particular value or behavior cannot be  
218 assured to be portable across conforming implementations.

219 **1.5 Portability**

220 Some of the utilities in the Shell and Utilities volume of IEEE Std 1003.1-200x and functions in  
221 the System Interfaces volume of IEEE Std 1003.1-200x describe functionality that might not be  
222 fully portable to systems meeting the requirements for POSIX conformance (see the Base  
223 Definitions volume of IEEE Std 1003.1-200x, Chapter 2, Conformance).

224 Where optional, enhanced, or reduced functionality is specified, the text is shaded and a code in  
225 the margin identifies the nature of the option, extension, or warning (see Section 1.5.1). For  
226 maximum portability, an application should avoid such functionality.

227 Unless the primary task of a utility is to produce textual material on its standard output,  
228 application developers should not rely on the format or content of any such material that may be  
229 produced. Where the primary task *is* to provide such material, but the output format is  
230 incompletely specified, the description is marked with the OF margin code and shading.  
231 Application developers are warned not to expect that the output of such an interface on one  
232 system is any guide to its behavior on another system.

233 **1.5.1 Codes**

234 The codes and their meanings are as follows. See also Section 1.5.2 (on page 14).

235 ADV **Advisory Information**

236 The functionality described is optional. The functionality described is also an extension to the  
237 ISO C standard.

238 Where applicable, functions are marked with the ADV margin legend in the SYNOPSIS section.  
239 Where additional semantics apply to a function, the material is identified by use of the ADV  
240 margin legend.

241 AIO **Asynchronous Input and Output**

242 The functionality described is optional. The functionality described is also an extension to the  
243 ISO C standard.

244 Where applicable, functions are marked with the AIO margin legend in the SYNOPSIS section.  
245 Where additional semantics apply to a function, the material is identified by use of the AIO

246		margin legend.
247	BAR	<b>Barriers</b>
248		The functionality described is optional. The functionality described is also an extension to the
249		ISO C standard.
250		Where applicable, functions are marked with the BAR margin legend in the SYNOPSIS section.
251		Where additional semantics apply to a function, the material is identified by use of the BAR
252		margin legend.
253	BE	<b>Batch Environment Services and Utilities</b>
254		The functionality described is optional.
255		Where applicable, utilities are marked with the BE margin legend in the SYNOPSIS section.
256		Where additional semantics apply to a utility, the material is identified by use of the BE margin
257		legend.
258	CD	<b>C-Language Development Utilities</b>
259		The functionality described is optional.
260		Where applicable, utilities are marked with the CD margin legend in the SYNOPSIS section.
261		Where additional semantics apply to a utility, the material is identified by use of the CD margin
262		legend.
263	CPT	<b>Process CPU-Time Clocks</b>
264		The functionality described is optional. The functionality described is also an extension to the
265		ISO C standard.
266		Where applicable, functions are marked with the CPT margin legend in the SYNOPSIS section.
267		Where additional semantics apply to a function, the material is identified by use of the CPT
268		margin legend.
269	CS	<b>Clock Selection</b>
270		The functionality described is optional. The functionality described is also an extension to the
271		ISO C standard.
272		Where applicable, functions are marked with the CS margin legend in the SYNOPSIS section.
273		Where additional semantics apply to a function, the material is identified by use of the CS
274		margin legend.
275	CX	<b>Extension to the ISO C standard</b>
276		The functionality described is an extension to the ISO C standard. Application writers may make
277		use of an extension as it is supported on all IEEE Std 1003.1-200x-conforming systems.
278		With each function or header from the ISO C standard, a statement to the effect that “any
279		conflict is unintentional” is included. That is intended to refer to a direct conflict.
280		IEEE Std 1003.1-200x acts in part as a profile of the ISO C standard, and it may choose to further
281		constrain behaviors allowed to vary by the ISO C standard. Such limitations are not considered
282		conflicts.
283	FD	<b>FORTTRAN Development Utilities</b>
284		The functionality described is optional.
285		Where applicable, utilities are marked with the FD margin legend in the SYNOPSIS section.
286		Where additional semantics apply to a utility, the material is identified by use of the FD margin
287		legend.
288	FR	<b>FORTTRAN Runtime Utilities</b>
289		The functionality described is optional.

290 Where applicable, utilities are marked with the FR margin legend in the SYNOPSIS section.  
291 Where additional semantics apply to a utility, the material is identified by use of the FR margin  
292 legend.

293 FSC **File Synchronization**

294 The functionality described is optional. The functionality described is also an extension to the  
295 ISO C standard.

296 Where applicable, functions are marked with the FSC margin legend in the SYNOPSIS section.  
297 Where additional semantics apply to a function, the material is identified by use of the FSC  
298 margin legend.

299 IP6 **IPV6**

300 The functionality described is optional. The functionality described is also an extension to the  
301 ISO C standard.

302 Where applicable, functions are marked with the IP6 margin legend in the SYNOPSIS section.  
303 Where additional semantics apply to a function, the material is identified by use of the IP6  
304 margin legend.

305 MC1 **Advisory Information and either Memory Mapped Files or Shared Memory Objects**

306 The functionality described is optional. The functionality described is also an extension to the  
307 ISO C standard.

308 This is a shorthand notation for combinations of multiple option codes.

309 Where applicable, functions are marked with the MC1 margin legend in the SYNOPSIS section.  
310 Where additional semantics apply to a function, the material is identified by use of the MC1  
311 margin legend.

312 Refer to Section 1.5.2 (on page 14).

313 MC2 **Memory Mapped Files, Shared Memory Objects, or Memory Protection**

314 The functionality described is optional. The functionality described is also an extension to the  
315 ISO C standard.

316 This is a shorthand notation for combinations of multiple option codes.

317 Where applicable, functions are marked with the MC2 margin legend in the SYNOPSIS section.  
318 Where additional semantics apply to a function, the material is identified by use of the MC2  
319 margin legend.

320 Refer to Section 1.5.2 (on page 14).

321 MF **Memory Mapped Files**

322 The functionality described is optional. The functionality described is also an extension to the  
323 ISO C standard.

324 Where applicable, functions are marked with the MF margin legend in the SYNOPSIS section.  
325 Where additional semantics apply to a function, the material is identified by use of the MF  
326 margin legend.

327 ML **Process Memory Locking**

328 The functionality described is optional. The functionality described is also an extension to the  
329 ISO C standard.

330 Where applicable, functions are marked with the ML margin legend in the SYNOPSIS section.  
331 Where additional semantics apply to a function, the material is identified by use of the ML  
332 margin legend.

333	MLR	<b>Range Memory Locking</b>
334		The functionality described is optional. The functionality described is also an extension to the
335		ISO C standard.
336		Where applicable, functions are marked with the MLR margin legend in the SYNOPSIS section.
337		Where additional semantics apply to a function, the material is identified by use of the MLR
338		margin legend.
339	MON	<b>Monotonic Clock</b>
340		The functionality described is optional. The functionality described is also an extension to the
341		ISO C standard.
342		Where applicable, functions are marked with the MON margin legend in the SYNOPSIS section.
343		Where additional semantics apply to a function, the material is identified by use of the MON
344		margin legend.
345	MPR	<b>Memory Protection</b>
346		The functionality described is optional. The functionality described is also an extension to the
347		ISO C standard.
348		Where applicable, functions are marked with the MPR margin legend in the SYNOPSIS section.
349		Where additional semantics apply to a function, the material is identified by use of the MPR
350		margin legend.
351	MSG	<b>Message Passing</b>
352		The functionality described is optional. The functionality described is also an extension to the
353		ISO C standard.
354		Where applicable, functions are marked with the MSG margin legend in the SYNOPSIS section.
355		Where additional semantics apply to a function, the material is identified by use of the MSG
356		margin legend.
357	MX	<b>IEC 60559 Floating-Point Option</b>
358		The functionality described is optional. The functionality described is also an extension to the
359		ISO C standard.
360		Where applicable, functions are marked with the MX margin legend in the SYNOPSIS section.
361		Where additional semantics apply to a function, the material is identified by use of the MX
362		margin legend.
363	OB	<b>Obsolescent</b>
364		The functionality described may be withdrawn in a future version of this volume of
365		IEEE Std 1003.1-200x. Strictly Conforming POSIX Applications and Strictly Conforming XSI
366		Applications shall not use obsolescent features.
367	OF	<b>Output Format Incompletely Specified</b>
368		The functionality described is an XSI extension. The format of the output produced by the utility
369		is not fully specified. It is therefore not possible to post-process this output in a consistent
370		fashion. Typical problems include unknown length of strings and unspecified field delimiters.
371	OH	<b>Optional Header</b>
372		In the SYNOPSIS section of some interfaces in the System Interfaces volume of
373		IEEE Std 1003.1-200x an included header is marked as in the following example:
374	OH	<code>#include &lt;sys/types.h&gt;</code>
375		<code>#include &lt;grp.h&gt;</code>
376		<code>struct group *getgrnam(const char *name);</code>

377 This indicates that the marked header is not required on XSI-conformant systems.

378 PIO **Prioritized Input and Output**  
379 The functionality described is optional. The functionality described is also an extension to the  
380 ISO C standard.

381 Where applicable, functions are marked with the PIO margin legend in the SYNOPSIS section.  
382 Where additional semantics apply to a function, the material is identified by use of the PIO  
383 margin legend.

384 PS **Process Scheduling**  
385 The functionality described is optional. The functionality described is also an extension to the  
386 ISO C standard.

387 Where applicable, functions are marked with the PS margin legend in the SYNOPSIS section.  
388 Where additional semantics apply to a function, the material is identified by use of the PS  
389 margin legend.

390 RS **Raw Sockets**  
391 The functionality described is optional. The functionality described is also an extension to the  
392 ISO C standard.

393 Where applicable, functions are marked with the RS margin legend in the SYNOPSIS section.  
394 Where additional semantics apply to a function, the material is identified by use of the RS  
395 margin legend.

396 RTS **Realtime Signals Extension**  
397 The functionality described is optional. The functionality described is also an extension to the  
398 ISO C standard.

399 Where applicable, functions are marked with the RTS margin legend in the SYNOPSIS section.  
400 Where additional semantics apply to a function, the material is identified by use of the RTS  
401 margin legend.

402 SD **Software Development Utilities**  
403 The functionality described is optional.

404 Where applicable, utilities are marked with the SD margin legend in the SYNOPSIS section.  
405 Where additional semantics apply to a utility, the material is identified by use of the SD margin  
406 legend.

407 SEM **Semaphores**  
408 The functionality described is optional. The functionality described is also an extension to the  
409 ISO C standard.

410 Where applicable, functions are marked with the SEM margin legend in the SYNOPSIS section.  
411 Where additional semantics apply to a function, the material is identified by use of the SEM  
412 margin legend.

413 SHM **Shared Memory Objects**  
414 The functionality described is optional. The functionality described is also an extension to the  
415 ISO C standard.

416 Where applicable, functions are marked with the SHM margin legend in the SYNOPSIS section.  
417 Where additional semantics apply to a function, the material is identified by use of the SHM  
418 margin legend.

419 SIO **Synchronized Input and Output**  
420 The functionality described is optional. The functionality described is also an extension to the  
421 ISO C standard.

422 Where applicable, functions are marked with the SIO margin legend in the SYNOPSIS section.  
423 Where additional semantics apply to a function, the material is identified by use of the SIO  
424 margin legend.

425 SPI **Spin Locks**  
426 The functionality described is optional. The functionality described is also an extension to the  
427 ISO C standard.

428 Where applicable, functions are marked with the SPI margin legend in the SYNOPSIS section.  
429 Where additional semantics apply to a function, the material is identified by use of the SPI  
430 margin legend.

431 SPN **Spawn**  
432 The functionality described is optional. The functionality described is also an extension to the  
433 ISO C standard.

434 Where applicable, functions are marked with the SPN margin legend in the SYNOPSIS section.  
435 Where additional semantics apply to a function, the material is identified by use of the SPN  
436 margin legend.

437 SS **Process Sporadic Server**  
438 The functionality described is optional. The functionality described is also an extension to the  
439 ISO C standard.

440 Where applicable, functions are marked with the SS margin legend in the SYNOPSIS section.  
441 Where additional semantics apply to a function, the material is identified by use of the SS  
442 margin legend.

443 TCT **Thread CPU-Time Clocks**  
444 The functionality described is optional. The functionality described is also an extension to the  
445 ISO C standard.

446 Where applicable, functions are marked with the TCT margin legend in the SYNOPSIS section.  
447 Where additional semantics apply to a function, the material is identified by use of the TCT  
448 margin legend.

449 TEF **Trace Event Filter**  
450 The functionality described is optional. The functionality described is also an extension to the  
451 ISO C standard.

452 Where applicable, functions are marked with the TEF margin legend in the SYNOPSIS section.  
453 Where additional semantics apply to a function, the material is identified by use of the TEF  
454 margin legend.

455 THR **Threads**  
456 The functionality described is optional. The functionality described is also an extension to the  
457 ISO C standard.

458 Where applicable, functions are marked with the THR margin legend in the SYNOPSIS section.  
459 Where additional semantics apply to a function, the material is identified by use of the THR  
460 margin legend.

461 TMO **Timeouts**  
462 The functionality described is optional. The functionality described is also an extension to the  
463 ISO C standard.

464 Where applicable, functions are marked with the TMO margin legend in the SYNOPSIS section.  
465 Where additional semantics apply to a function, the material is identified by use of the TMO  
466 margin legend.

467	TMR	<b>Timers</b>
468		The functionality described is optional. The functionality described is also an extension to the
469		ISO C standard.
470		Where applicable, functions are marked with the TMR margin legend in the SYNOPSIS section.
471		Where additional semantics apply to a function, the material is identified by use of the TMR
472		margin legend.
473	TPI	<b>Thread Priority Inheritance</b>
474		The functionality described is optional. The functionality described is also an extension to the
475		ISO C standard.
476		Where applicable, functions are marked with the TPI margin legend in the SYNOPSIS section.
477		Where additional semantics apply to a function, the material is identified by use of the TPI
478		margin legend.
479	TPP	<b>Thread Priority Protection</b>
480		The functionality described is optional. The functionality described is also an extension to the
481		ISO C standard.
482		Where applicable, functions are marked with the TPP margin legend in the SYNOPSIS section.
483		Where additional semantics apply to a function, the material is identified by use of the TPP
484		margin legend.
485	TPS	<b>Thread Execution Scheduling</b>
486		The functionality described is optional. The functionality described is also an extension to the
487		ISO C standard.
488		Where applicable, functions are marked with the TPS margin legend for the SYNOPSIS section.
489		Where additional semantics apply to a function, the material is identified by use of the TPS
490		margin legend.
491	TRC	<b>Trace</b>
492		The functionality described is optional. The functionality described is also an extension to the
493		ISO C standard.
494		Where applicable, functions are marked with the TRC margin legend in the SYNOPSIS section.
495		Where additional semantics apply to a function, the material is identified by use of the TRC
496		margin legend.
497	TRI	<b>Trace Inherit</b>
498		The functionality described is optional. The functionality described is also an extension to the
499		ISO C standard.
500		Where applicable, functions are marked with the TRI margin legend in the SYNOPSIS section.
501		Where additional semantics apply to a function, the material is identified by use of the TRI
502		margin legend.
503	TRL	<b>Trace Log</b>
504		The functionality described is optional. The functionality described is also an extension to the
505		ISO C standard.
506		Where applicable, functions are marked with the TRL margin legend in the SYNOPSIS section.
507		Where additional semantics apply to a function, the material is identified by use of the TRL
508		margin legend.
509	TSA	<b>Thread Stack Address Attribute</b>
510		The functionality described is optional. The functionality described is also an extension to the
511		ISO C standard.

512 Where applicable, functions are marked with the TSA margin legend for the SYNOPSIS section.  
513 Where additional semantics apply to a function, the material is identified by use of the TSA  
514 margin legend.

515 TSF **Thread-Safe Functions**

516 The functionality described is optional. The functionality described is also an extension to the  
517 ISO C standard.

518 Where applicable, functions are marked with the TSF margin legend in the SYNOPSIS section.  
519 Where additional semantics apply to a function, the material is identified by use of the TSF  
520 margin legend.

521 TSH **Thread Process-Shared Synchronization**

522 The functionality described is optional. The functionality described is also an extension to the  
523 ISO C standard.

524 Where applicable, functions are marked with the TSH margin legend in the SYNOPSIS section.  
525 Where additional semantics apply to a function, the material is identified by use of the TSH  
526 margin legend.

527 TSP **Thread Sporadic Server**

528 The functionality described is optional. The functionality described is also an extension to the  
529 ISO C standard.

530 Where applicable, functions are marked with the TSP margin legend in the SYNOPSIS section.  
531 Where additional semantics apply to a function, the material is identified by use of the TSP  
532 margin legend.

533 TSS **Thread Stack Address Size**

534 The functionality described is optional. The functionality described is also an extension to the  
535 ISO C standard.

536 Where applicable, functions are marked with the TSS margin legend in the SYNOPSIS section.  
537 Where additional semantics apply to a function, the material is identified by use of the TSS  
538 margin legend.

539 TYM **Typed Memory Objects**

540 The functionality described is optional. The functionality described is also an extension to the  
541 ISO C standard.

542 Where applicable, functions are marked with the TYM margin legend in the SYNOPSIS section.  
543 Where additional semantics apply to a function, the material is identified by use of the TYM  
544 margin legend.

545 UP **User Portability Utilities**

546 The functionality described is optional.

547 Where applicable, utilities are marked with the UP margin legend in the SYNOPSIS section.  
548 Where additional semantics apply to a utility, the material is identified by use of the UP margin  
549 legend.

550 XSI **Extension**

551 The functionality described is an XSI extension. Functionality marked XSI is also an extension to  
552 the ISO C standard. Application writers may confidently make use of an extension on all  
553 systems supporting the X/Open System Interfaces Extension.

554 If an entire SYNOPSIS section is shaded and marked XSI, all the functionality described in that  
555 reference page is an extension. See Section 2.1.4 (on page 19).

556 XSR **XSI STREAMS**  
557 The functionality described is optional. The functionality described is also an extension to the  
558 ISO C standard.

559 Where applicable, functions are marked with the XSR margin legend in the SYNOPSIS section.  
560 Where additional semantics apply to a function, the material is identified by use of the XSR  
561 margin legend.

## 562 1.5.2 Margin Code Notation

563 Some of the functionality described in IEEE Std 1003.1-200x depends on support of more than  
564 one option, or independently may depend on several options. The following notation for margin  
565 codes is used to denote the following cases.

### 566 A Feature Dependent on One or Two Options

567 In this case, margin codes have a <space> separator; for example:

568 MF This feature requires support for only the Memory Mapped Files option.

569 MF SHM This feature requires support for both the Memory Mapped Files and the Shared Memory  
570 Objects options; that is, an application which uses this feature is portable only between  
571 implementations that provide both options.

### 572 A Feature Dependent on Either of the Options Denoted

573 In this case, margin codes have a ' | ' separator to denote the logical OR; for example:

574 MF|SHM This feature is dependent on support for either the Memory Mapped Files option or the Shared  
575 Memory Objects option; that is, an application which uses this feature is portable between  
576 implementations that provide any (or all) of the options.

### 577 A Feature Dependent on More than Two Options

578 The following shorthand notations are used:

579 MC1 The MC1 margin code is shorthand for ADV (MF|SHM). Features which are shaded with this  
580 margin code require support of the Advisory Information option and either the Memory  
581 Mapped Files or Shared Memory Objects option.

582 MC2 The MC2 margin code is shorthand for MF|SHM|MPR. Features which are shaded with this  
583 margin code require support of either the Memory Mapped Files, Shared Memory Objects, or  
584 Memory Protection options.

### 585 Large Sections Dependent on an Option

586 Where large sections of text are dependent on support for an option, a lead-in text block is  
587 provided and shaded accordingly; for example:

588 TRC This section describes extensions to support tracing of user applications. This functionality is  
589 dependent on support of the Trace option (and the rest of this section is not further shaded for  
590 this option).

591

**2.1 Implementation Conformance****2.1.1 Requirements**

A *conforming implementation* shall meet all of the following criteria:

1. The system shall support all utilities, functions, and facilities defined within IEEE Std 1003.1-200x that are required for POSIX conformance (see Section 2.1.3 (on page 16)). These interfaces shall support the functional behavior described herein.
2. The system may support one or more options as described under Section 2.1.5 (on page 20). When an implementation claims that an option is supported, all of its constituent parts shall be provided.
3. The system may support the X/Open System Interface Extension (XSI) as described under Section 2.1.4 (on page 19).
4. The system may provide additional utilities, functions, or facilities not required by IEEE Std 1003.1-200x. Non-standard extensions of the utilities, functions, or facilities specified in IEEE Std 1003.1-200x should be identified as such in the system documentation. Non-standard extensions, when used, may change the behavior of utilities, functions, or facilities defined by IEEE Std 1003.1-200x. The conformance document shall define an environment in which an application can be run with the behavior specified by IEEE Std 1003.1-200x. In no case shall such an environment require modification of a Strictly Conforming POSIX Application (see Section 2.2.1 (on page 29)).

**2.1.2 Documentation**

A conformance document with the following information shall be available for an implementation claiming conformance to IEEE Std 1003.1-200x. The conformance document shall have the same structure as IEEE Std 1003.1-200x, with the information presented in the appropriate sections and subsections. Sections and subsections that consist solely of subordinate section titles, with no other information, are not required. The conformance document shall not contain information about extended facilities or capabilities outside the scope of IEEE Std 1003.1-200x.

The conformance document shall contain a statement that indicates the full name, number, and date of the standard that applies. The conformance document may also list international software standards that are available for use by a Conforming POSIX Application. Applicable characteristics where documentation is required by one of these standards, or by standards of government bodies, may also be included.

The conformance document shall describe the limit values found in the headers `<limits.h>` (on page 245) and `<unistd.h>` (on page 398), stating values, the conditions under which those values may change, and the limits of such variations, if any.

The conformance document shall describe the behavior of the implementation for all implementation-defined features defined in IEEE Std 1003.1-200x. This requirement shall be met by listing these features and providing either a specific reference to the system documentation or providing full syntax and semantics of these features. When the value or behavior in the

631 implementation is designed to be variable or customized on each instantiation of the system, the  
 632 implementation provider shall document the nature and permissible ranges of this variation.

633 The conformance document may specify the behavior of the implementation for those features  
 634 where IEEE Std 1003.1-200x states that implementations may vary or where features are  
 635 identified as undefined or unspecified.

636 The conformance document shall not contain documentation other than that specified in the  
 637 preceding paragraphs except where such documentation is specifically allowed or required by  
 638 other provisions of IEEE Std 1003.1-200x.

639 The phrases “shall document” or “shall be documented” in IEEE Std 1003.1-200x mean that  
 640 documentation of the feature shall appear in the conformance document, as described  
 641 previously, unless there is an explicit reference in the conformance document to show where the  
 642 information can be found in the system documentation.

643 The system documentation should also contain the information found in the conformance  
 644 document.

### 645 2.1.3 POSIX Conformance

646 A conforming implementation shall meet the following criteria for POSIX conformance.

#### 647 2.1.3.1 POSIX System Interfaces

648 • The system shall support all the mandatory functions and headers defined in |  
 649 IEEE Std 1003.1-200x, and shall set the symbolic constant `_POSIX_VERSION` to the value |  
 650 `200xxxL`. |

651 • Although all implementations conforming to IEEE Std 1003.1-200x support all the features |  
 652 described below, there may be system-dependent or file system-dependent configuration |  
 653 procedures that can remove or modify any or all of these features. Such configurations |  
 654 should not be made if strict compliance is required.

655 The following symbolic constants shall either be undefined or defined with a value other  
 656 than `-1`. If a constant is undefined, an application should use the `sysconf()`, `pathconf()`, or  
 657 `fpathconf()` functions, or the `getconf` utility, to determine which features are present on the  
 658 system at that time or for the particular pathname in question.

659 — `_POSIX_CHOWN_RESTRICTED`

660 The use of `chown()` is restricted to a process with appropriate privileges, and to changing  
 661 the group ID of a file only to the effective group ID of the process or to one of its  
 662 supplementary group IDs.

663 — `_POSIX_NO_TRUNC`

664 Pathname components longer than `{NAME_MAX}` generate an error.

665 • The following symbolic constants shall be defined as follows: |

666 • `_POSIX_JOB_CONTROL` shall have a value greater than zero. |

667 • `_POSIX_SAVED_IDS` shall have a value greater than zero. |

668 • `_POSIX_VDISABLE` shall have a value other than `-1`. |

669 **Note:** The symbols above represent historical options that are no longer allowed as options, but  
 670 are retained here for backwards-compatibility of applications.

- 671 • The system may support one or more options (see Section 2.1.6 (on page 26)) denoted by the
- 672 following symbolic constants:
- 673 — \_POSIX\_ADVISORY\_INFO
- 674 — \_POSIX\_ASYNCHRONOUS\_IO
- 675 — \_POSIX\_BARRIERS
- 676 — \_POSIX\_CLOCK\_SELECTION
- 677 — \_POSIX\_CPUTIME
- 678 — \_POSIX\_FSYNC
- 679 — \_POSIX\_IPV6
- 680 — \_POSIX\_MAPPED\_FILES
- 681 — \_POSIX\_MEMLOCK
- 682 — \_POSIX\_MEMLOCK\_RANGE
- 683 — \_POSIX\_MEMORY\_PROTECTION
- 684 — \_POSIX\_MESSAGE\_PASSING
- 685 — \_POSIX\_MONOTONIC\_CLOCK
- 686 — \_POSIX\_PRIORITIZED\_IO
- 687 — \_POSIX\_PRIORITY\_SCHEDULING
- 688 — \_POSIX\_RAW\_SOCKETS
- 689 — \_POSIX\_REALTIME\_SIGNALS
- 690 — \_POSIX\_SEMAPHORES
- 691 — \_POSIX\_SHARED\_MEMORY\_OBJECTS
- 692 — \_POSIX\_SPAWN
- 693 — \_POSIX\_SPIN\_LOCKS
- 694 — \_POSIX\_SPORADIC\_SERVER
- 695 — \_POSIX\_SYNCHRONIZED\_IO
- 696 — \_POSIX\_THREAD\_ATTR\_STACKADDR
- 697 — \_POSIX\_THREAD\_CPUTIME
- 698 — \_POSIX\_THREAD\_ATTR\_STACKSIZE
- 699 — \_POSIX\_THREAD\_PRIO\_INHERIT
- 700 — \_POSIX\_THREAD\_PRIO\_PROTECT
- 701 — \_POSIX\_THREAD\_PRIORITY\_SCHEDULING
- 702 — \_POSIX\_THREAD\_PROCESS\_SHARED
- 703 — \_POSIX\_THREAD\_SAFE\_FUNCTIONS
- 704 — \_POSIX\_THREAD\_SPARADIC\_SERVER
- 705 — \_POSIX\_THREADS

706 — \_POSIX\_TIMEOUTS  
 707 — \_POSIX\_TIMERS  
 708 — \_POSIX\_TRACE  
 709 — \_POSIX\_TRACE\_EVENT\_FILTER  
 710 — \_POSIX\_TRACE\_INHERIT  
 711 — \_POSIX\_TRACE\_LOG  
 712 — \_POSIX\_TYPED\_MEMORY\_OBJECTS

713 If any of the symbolic constants \_POSIX\_TRACE\_EVENT\_FILTER, \_POSIX\_TRACE\_LOG, or  
 714 \_POSIX\_TRACE\_INHERIT is defined to have a value other than -1, then the symbolic  
 715 constant \_POSIX\_TRACE shall also be defined to have a value other than -1.

716 XSI • The system may support the XSI extensions (see Section 2.1.5.2 (on page 21)) denoted by the  
 717 following symbolic constants:

718 — \_XOPEN\_CRYPT  
 719 — \_XOPEN\_LEGACY  
 720 — \_XOPEN\_REALTIME  
 721 — \_XOPEN\_REALTIME\_THREADS  
 722 — \_XOPEN\_UNIX

### 723 2.1.3.2 POSIX Shell and Utilities

724 • The system shall provide all the mandatory utilities in the Shell and Utilities volume of  
 725 IEEE Std 1003.1-200x with all the functional behavior described therein.

726 • The system shall support the Large File capabilities described in the Shell and Utilities  
 727 volume of IEEE Std 1003.1-200x.

728 • The system may support one or more options (see Section 2.1.6 (on page 26)) denoted by the  
 729 following symbolic constants. (The literal names below apply to the *getconf* utility.)

730 — POSIX2\_C\_DEV  
 731 — POSIX2\_CHAR\_TERM  
 732 — POSIX2\_FORT\_DEV  
 733 — POSIX2\_FORT\_RUN  
 734 — POSIX2\_LOCALEDEF  
 735 — POSIX2\_PBS  
 736 — POSIX2\_PBS\_ACCOUNTING  
 737 — POSIX2\_PBS\_LOCATE  
 738 — POSIX2\_PBS\_MESSAGE  
 739 — POSIX2\_PBS\_TRACK  
 740 — POSIX2\_SW\_DEV  
 741 — POSIX2\_UPE

- 742           • The system may support the XSI extensions (see Section 2.1.4).

743           Additional language bindings and development utility options may be provided in other related  
744           standards or in a future version of IEEE Std 1003.1-200x. In the former case, additional symbolic  
745           constants of the same general form as shown in this subsection should be defined by the related  
746           standard document and made available to the application without requiring  
747           IEEE Std 1003.1-200x to be updated.

#### 748   **2.1.4   XSI Conformance**

749   XSI   This section describes the criteria for implementations conforming to the X/Open System  
750   Interface extension. This functionality is dependent on the support of the XSI extension (and the  
751   rest of this section is not further shaded).

752           IEEE Std 1003.1-200x describes utilities, functions, and facilities offered to application programs  
753           by the X/Open System Interface (XSI). An XSI-conforming implementation shall meet the  
754           criteria for POSIX conformance and the following requirements.

##### 755   2.1.4.1   XSI System Interfaces

- 756           • The system shall support all the functions and headers defined in IEEE Std 1003.1-200x as  
757           part of the XSI extension denoted by the symbolic constant `_XOPEN_UNIX` and any  
758           extensions marked with the XSI extension marking (see Section 1.5.1 (on page 6)).

- 759           • The system shall support the `mmap()`, `munmap()`, and `msync()` functions.

- 760           • The system shall support the following options defined within IEEE Std 1003.1-200x (see  
761           Section 2.1.6 (on page 26)):

762           — `_POSIX_FSYNC`

763           — `_POSIX_MAPPED_FILES`

764           — `_POSIX_MEMORY_PROTECTION`

765           — `_POSIX_THREAD_ATTR_STACKADDR`

766           — `_POSIX_THREAD_ATTR_STACKSIZE`

767           — `_POSIX_THREAD_PROCESS_SHARED`

768           — `_POSIX_THREAD_SAFE_FUNCTIONS`

769           — `_POSIX_THREADS`

- 770           • The system may support the following XSI Option Groups (see Section 2.1.5.2 (on page 21)) |  
771           defined within IEEE Std 1003.1-200x:

772           — Encryption

773           — Realtime

774           — Advanced Realtime

775           — Realtime Threads

776           — Advanced Realtime Threads

777           — Tracing

778           — XSI STREAMS

779           — Legacy

## 780 2.1.4.2 XSI Shell and Utilities Conformance

781       • The system shall support all the utilities defined in the Shell and Utilities volume of  
782       IEEE Std 1003.1-200x as part of the XSI extension denoted by the XSI marking in the  
783       SYNOPSIS section, and any extensions marked with the XSI extension marking (see Section  
784       1.5.1 (on page 6)) within the text.

785       • The system shall support the User Portability Utilities option.

786       • The system shall support creation of locales (see Chapter 7 (on page 119)).

787       • The C-language Development utility *c99* shall be supported.

788       • The XSI Development Utilities option may be supported. It consists of the following software  
789       development utilities:

790	<i>admin</i>	<i>delta</i>	<i>rmdel</i>	<i>val</i>
791	<i>cflow</i>	<i>get</i>	<i>sact</i>	<i>what</i>
792	<i>ctags</i>	<i>m4</i>	<i>sccs</i>	
793	<i>cxref</i>	<i>prs</i>	<i>unget</i>	

794       • Within the utilities that are provided, functionality marked by the code OF (see Section 1.5.1  
795       (on page 6)) need not be provided.

## 796 2.1.5 Option Groups

797       An Option Group is a group of related functions or options defined within the System Interfaces  
798       volume of IEEE Std 1003.1-200x.

799       If an implementation supports an Option Group, then the system shall support the functional  
800       behavior described herein.

801       If an implementation does not support an Option Group, then the system need not support the  
802       functional behavior described herein.

## 803 2.1.5.1 Subprofiling Considerations

804       Profiling standards supporting functional requirements less than that required in  
805       IEEE Std 1003.1-200x may subset both mandatory and optional functionality required for POSIX  
806       Conformance (see Section 2.1.3 (on page 16)) or XSI Conformance (see Section 2.1.4 (on page  
807       19)). Such profiles shall organize the subsets into Subprofiling Option Groups.

808       The Rationale (Informative) volume of IEEE Std 1003.1-200x, Appendix E, Subprofiling  
809       Considerations (Informative) describes a representative set of such Subprofiling Option Groups  
810       for use by profiles applicable to specialized realtime systems. IEEE Std 1003.1-200x does not  
811       require that the presence of Subprofiling Option Groups be testable at compile-time (as symbols  
812       defined in any header) or at runtime (via *sysconf()* or *getconf()*).

813       A Subprofiling Option Group may provide basic system functionality that other Subprofiling  
814       Option Groups and other options depend upon.<sup>3</sup> If a profile of IEEE Std 1003.1-200x does not

815

816 3. As an example, the File System profiling option group provides underlying support for pathname resolution and file creation  
817 which are needed by any interface in IEEE Std 1003.1-200x that parses a *path* argument. If a profile requires support for the  
818 Device Input and Output profiling option group but does not require support for the File System profiling option group, the  
819 profile must specify how pathname resolution is to behave in that profile, how the *O\_CREAT* flag to *open()* is to be handled (and  
820 the use of the character 'a' in the *mode* argument of *open()* when a filename argument names a file that does not exist), and  
821 specify lots of other details.

822 require an implementation to provide a Subprofiling Option Group that provides features |  
 823 utilized by a required Subprofiling Option Group (or option),<sup>4</sup> the profile shall specify<sup>5</sup> all of the |  
 824 following: |

- 825 • Restricted or altered behavior of interfaces defined in IEEE Std 1003.1-200x that may differ on |  
 826 an implementation of the profile
- 827 • Additional behaviors that may produce undefined or unspecified results
- 828 • Additional implementation-defined behavior that implementations shall be required to |  
 829 document in the profile's conformance document

830 if any of the above is a result of the profile not requiring an interface required by |  
 831 IEEE Std 1003.1-200x. |

832 The following additional rules shall apply to all standard profiles of IEEE Std 1003.1-200x: |

- 833 • Any application that conforms to that profile shall also conform to IEEE Std 1003.1-200x (that |  
 834 is, a profile cannot require restricted, altered, or extended behaviors).
- 835 • Any implementation that conforms to IEEE Std 1003.1-200x (including all options required |  
 836 by the profile) shall also conform to that profile |

### 837 2.1.5.2 XSI Option Groups

838 XSI This section describes Option Groups to support the definition of XSI conformance within the |  
 839 System Interfaces volume of IEEE Std 1003.1-200x. This functionality is dependent on the |  
 840 support of the XSI extension (and the rest of this section is not further shaded). |

841 The following Option Groups are defined.

#### 842 **Encryption**

843 The Encryption Option Group is denoted by the symbolic constant `_XOPEN_CRYPT`. It includes |  
 844 the following functions:

845 `crypt()`, `encrypt()`, `setkey()`

846 These functions are marked CRYPT.

847 Due to export restrictions on the decoding algorithm in some countries, implementations may be |  
 848 restricted in making these functions available. All the functions in the Encryption Option Group |  
 849 may therefore return [ENOSYS] or, alternatively, `encrypt()` shall return [ENOSYS] for the |  
 850 decryption operation.

851 An implementation that claims conformance to this Option Group shall set `_XOPEN_CRYPT` to |  
 852 a value other than `-1`. |

853 \_\_\_\_\_

854 4. As an example, IEEE Std 1003.1-200x requires that implementations claiming to support the Range Memory Locking option also |  
 855 support the Process Memory Locking option. A profile could require that the Range Memory Locking option had to be supplied |  
 856 without requiring that the Process Memory Locking option be supplied as long as the profile specifies everything an application |  
 857 writer or system implementor would have to know to build an application or implementation conforming to the profile.

858 5. Note that the profile could just specify that any use of the features not specified by the profile would produce undefined or |  
 859 unspecified results. |

860 **Realtime**

861 The Realtime Option Group is denoted by the symbolic constant `_XOPEN_REALTIME`.

862 This Option Group includes a set of realtime functions drawn from options within  
863 IEEE Std 1003.1-200x (see Section 2.1.6 (on page 26)).

864 Where entire functions are included in the Option Group, the NAME section is marked with  
865 REALTIME. Where additional semantics have been added to existing pages, the new material is  
866 identified by use of the appropriate margin legend for the underlying option defined within  
867 IEEE Std 1003.1-200x.

868 An implementation that claims conformance to this Option Group shall set |  
869 `_XOPEN_REALTIME` to a value other than `-1`. |

870 This Option Group consists of the set of the following options from within IEEE Std 1003.1-200x  
871 (see Section 2.1.6 (on page 26)):

872 `_POSIX_ASYNCHRONOUS_IO`  
873 `_POSIX_FSYNC`  
874 `_POSIX_MAPPED_FILES`  
875 `_POSIX_MEMLOCK`  
876 `_POSIX_MEMLOCK_RANGE`  
877 `_POSIX_MEMORY_PROTECTION`  
878 `_POSIX_MESSAGE_PASSING`  
879 `_POSIX_PRIORITIZED_IO`  
880 `_POSIX_PRIORITY_SCHEDULING`  
881 `_POSIX_REALTIME_SIGNALS`  
882 `_POSIX_SEMAPHORES`  
883 `_POSIX_SHARED_MEMORY_OBJECTS`  
884 `_POSIX_SYNCHRONIZED_IO`  
885 `_POSIX_TIMERS`

886 If the symbolic constant `_XOPEN_REALTIME` is defined to have a value other than `-1`, then the  
887 following symbolic constants shall be defined by the implementation to have the value `200xxxL`: |

888 `_POSIX_ASYNCHRONOUS_IO`  
889 `_POSIX_MEMLOCK`  
890 `_POSIX_MEMLOCK_RANGE`  
891 `_POSIX_MESSAGE_PASSING`  
892 `_POSIX_PRIORITY_SCHEDULING`  
893 `_POSIX_REALTIME_SIGNALS`  
894 `_POSIX_SEMAPHORES`  
895 `_POSIX_SHARED_MEMORY_OBJECTS`  
896 `_POSIX_SYNCHRONIZED_IO`  
897 `_POSIX_TIMERS`

898 The functionality associated with `_POSIX_MAPPED_FILES`, `_POSIX_MEMORY_PROTECTION`,  
899 and `_POSIX_FSYNC` is always supported on XSI-conformant systems.

900 Support of `_POSIX_PRIORITIZED_IO` on XSI-conformant systems is optional. If this  
901 functionality is supported, then `_POSIX_PRIORITIZED_IO` shall be set to a value other than `-1`.  
902 Otherwise, it shall be undefined.

903 If `_POSIX_PRIORITIZED_IO` is supported, then asynchronous I/O operations performed by  
904 `aio_read()`, `aio_write()`, and `lio_listio()` shall be submitted at a priority equal to the scheduling  
905 priority of the process minus `aiocbp->aio_reqprio`. The implementation shall also document for  
906 which files I/O prioritization is supported.

907       **Advanced Realtime**

908       An implementation that claims conformance to this Option Group shall also support the  
909       Realtime Option Group.

910       Where entire functions are included in the Option Group, the NAME section is marked with  
911       ADVANCED REALTIME. Where additional semantics have been added to existing pages, the  
912       new material is identified by use of the appropriate margin legend for the underlying option  
913       defined within IEEE Std 1003.1-200x.

914       This Option Group consists of the set of the following options from within IEEE Std 1003.1-200x  
915       (see Section 2.1.6 (on page 26)):

```
916        _POSIX_ADVISORY_INFO
917        _POSIX_CLOCK_SELECTION
918        _POSIX_CPUTIME
919        _POSIX_MONOTONIC_CLOCK
920        _POSIX_SPAWN
921        _POSIX_SPORADIC_SERVER
922        _POSIX_TIMEOUTS
923        _POSIX_TYPED_MEMORY_OBJECTS
```

924       If the implementation supports the Advanced Realtime Option Group, then the following  
925       symbolic constants shall be defined by the implementation to have the value 200xxxL:

```
926        _POSIX_ADVISORY_INFO
927        _POSIX_CLOCK_SELECTION
928        _POSIX_CPUTIME
929        _POSIX_MONOTONIC_CLOCK
930        _POSIX_SPAWN
931        _POSIX_SPORADIC_SERVER
932        _POSIX_TIMEOUTS
933        _POSIX_TYPED_MEMORY_OBJECTS
```

934       If the symbolic constant `_POSIX_SPORADIC_SERVER` is defined, then the symbolic constant  
935       `_POSIX_PRIORITY_SCHEDULING` shall also be defined by the implementation to have the  
936       value 200xxxL.

937       If the symbolic constant `_POSIX_CPUTIME` is defined, then the symbolic constant  
938       `_POSIX_TIMERS` shall also be defined by the implementation to have the value 200xxxL.

939       If the symbolic constant `_POSIX_MONOTONIC_CLOCK` is defined, then the symbolic constant  
940       `_POSIX_TIMERS` shall also be defined by the implementation to have the value 200xxxL.

941       If the symbolic constant `_POSIX_CLOCK_SELECTION` is defined, then the symbolic constant  
942       `_POSIX_TIMERS` shall also be defined by the implementation to have the value 200xxxL.

943       **Realtime Threads**

944       The Realtime Threads Option Group is denoted by the symbolic constant  
945       `_XOPEN_REALTIME_THREADS`.

946       This Option Group consists of the set of the following options from within IEEE Std 1003.1-200x  
947       (see Section 2.1.6 (on page 26)):

```
948        _POSIX_THREAD_PRIO_INHERIT
949        _POSIX_THREAD_PRIO_PROTECT
950        _POSIX_THREAD_PRIORITY_SCHEDULING
```

951 Where applicable, whole pages are marked REALTIME THREADS, together with the  
952 appropriate option margin legend for the SYNOPSIS section (see Section 1.5.1 (on page 6)).

953 An implementation that claims conformance to this Option Group shall set  
954 `_XOPEN_REALTIME_THREADS` to a value other than `-1`.

955 If the symbol `_XOPEN_REALTIME_THREADS` is defined to have a value other than `-1`, then the  
956 following options shall also be defined by the implementation to have the value `200xxxL`:

957 `_POSIX_THREAD_PRIO_INHERIT`  
958 `_POSIX_THREAD_PRIO_PROTECT`  
959 `_POSIX_THREAD_PRIORITY_SCHEDULING`

## 960 **Advanced Realtime Threads**

961 An implementation that claims conformance to this Option Group shall also support the  
962 Realtime Threads Option Group.

963 Where entire functions are included in the Option Group, the NAME section is marked with  
964 `ADVANCED_REALTIME_THREADS`. Where additional semantics have been added to existing  
965 pages, the new material is identified by use of the appropriate margin legend for the underlying  
966 option defined within IEEE Std 1003.1-200x.

967 This Option Group consists of the set of the following options from within IEEE Std 1003.1-200x  
968 (see Section 2.1.6 (on page 26)):

969 `_POSIX_BARRIERS`  
970 `_POSIX_SPIN_LOCKS`  
971 `_POSIX_THREAD_CPUTIME`  
972 `_POSIX_THREAD_SPORADIC_SERVER`

973 If the symbolic constant `_POSIX_THREAD_SPORADIC_SERVER` is defined to have the value  
974 `200xxxL`, then the symbolic constant `_POSIX_THREAD_PRIORITY_SCHEDULING` shall also be  
975 defined by the implementation to have the value `200xxxL`.

976 If the symbolic constant `_POSIX_THREAD_CPUTIME` is defined to have the value `200xxxL`,  
977 then the symbolic constant `_POSIX_TIMERS` shall also be defined by the implementation to have  
978 the value `200xxxL`.

979 If the symbolic constant `_POSIX_BARRIERS` is defined to have the value `200xxxL`, then the  
980 symbolic constants `_POSIX_THREADS` and `_POSIX_THREAD_SAFE_FUNCTIONS` shall also  
981 be defined by the implementation to have the value `200xxxL`.

982 If the symbolic constant `_POSIX_SPIN_LOCKS` is defined to have the value `200xxxL`, then the  
983 symbolic constants `_POSIX_THREADS` and `_POSIX_THREAD_SAFE_FUNCTIONS` shall also  
984 be defined by the implementation to have the value `200xxxL`.

985 If the implementation supports the Advanced Realtime Threads Option Group, then the  
986 following symbolic constants shall be defined by the implementation to have the value `200xxxL`:

987 `_POSIX_BARRIERS`  
988 `_POSIX_SPIN_LOCKS`  
989 `_POSIX_THREAD_CPUTIME`  
990 `_POSIX_THREAD_SPORADIC_SERVER`

991 **Tracing**

992 This Option Group includes a set of tracing functions drawn from options within  
 993 IEEE Std 1003.1-200x (see Section 2.1.6 (on page 26)).

994 Where entire functions are included in the Option Group, the NAME section is marked with  
 995 TRACING. Where additional semantics have been added to existing pages, the new material is  
 996 identified by use of the appropriate margin legend for the underlying option defined within  
 997 IEEE Std 1003.1-200x.

998 This Option Group consists of the set of the following options from within IEEE Std 1003.1-200x  
 999 (see Section 2.1.6 (on page 26)):

1000 `_POSIX_TRACE`  
 1001 `_POSIX_TRACE_EVENT_FILTER`  
 1002 `_POSIX_TRACE_LOG`  
 1003 `_POSIX_TRACE_INHERIT`

1004 If the implementation supports the Tracing Option Group, then the following symbolic  
 1005 constants shall be defined by the implementation to have the value 200xxxL:

1006 `_POSIX_TRACE`  
 1007 `_POSIX_TRACE_EVENT_FILTER`  
 1008 `_POSIX_TRACE_LOG`  
 1009 `_POSIX_TRACE_INHERIT`

1010 **XSI STREAMS**

1011 The XSI STREAMS Option Group is denoted by the symbolic constant `_XOPEN_STREAMS`.

1012 This Option Group includes functionality related to STREAMS, a uniform mechanism for  
 1013 implementing networking services and other character-based I/O as described in the System  
 1014 Interfaces volume of IEEE Std 1003.1-200x, Section 2.6, STREAMS.

1015 It includes the following functions:

1016 `fattach()`, `fdetach()`, `getmsg()`, `getpmsg()`, `ioctl()`, `isastream()`, `putmsg()`, `putpmsg()`

1017 and the `<stropts.h>` header.

1018 Where applicable, whole pages are marked STREAMS, together with the appropriate option  
 1019 margin legend for the SYNOPSIS section (see Section 1.5.1 (on page 6)). Where additional  
 1020 semantics have been added to existing pages, the new material is identified by use of the  
 1021 appropriate margin legend for the underlying option defined within IEEE Std 1003.1-200x.

1022 An implementation that claims conformance to this Open Group shall set `_XOPEN_STREAMS`  
 1023 to a value other than `-1`.

1024 **Legacy**

1025 The Legacy Option Group is denoted by the symbolic constant `_XOPEN_LEGACY`.

1026 The Legacy Option Group includes the functions and headers which were mandatory in  
 1027 previous versions of IEEE Std 1003.1-200x but are optional in this version.

1028 These functions and headers are retained in IEEE Std 1003.1-200x because of their widespread  
 1029 use. Application writers should not rely on the existence of these functions or headers in new  
 1030 applications, but should follow the migration path detailed in the APPLICATION USAGE  
 1031 sections of the relevant pages.

1032 Various factors may have contributed to the decision to mark a function or header LEGACY. In  
 1033 all cases, the specific reasons for the withdrawal of a function or header are documented on the  
 1034 relevant pages.

1035 Once a function or header is marked LEGACY, no modifications are made to the specifications  
 1036 of such functions or headers other than to the APPLICATION USAGE sections of the relevant  
 1037 pages.

1038 The functions and headers which form this Option Group are as follows:

1039 *bcmp()*, *bcopy()*, *bzero()*, *ecvt()*, *fcvt()*, *ftime()*, *gcvt()*, *getwd()*, *index()*, *mktemp()*, *rindex()*,  
 1040 *utimes()*, *wcswcs()*

1041 An implementation that claims conformance to this Option Group shall set `_XOPEN_LEGACY` |  
 1042 to a value other than `-1`. |

## 1043 2.1.6 Options

1044 The symbolic constants defined in `<unistd.h>`, **Constants for Options and Option Groups** (on |  
 1045 page 398) reflect implementation options for IEEE Std 1003.1-200x. These symbols can be used |  
 1046 by the application to determine which optional facilities are present on the implementation. The |  
 1047 *sysconf()* function defined in the System Interfaces volume of IEEE Std 1003.1-200x or the *getconf* |  
 1048 utility defined in the Shell and Utilities volume of IEEE Std 1003.1-200x can be used to retrieve |  
 1049 the value of each symbol on each specific implementation to determine whether the option is |  
 1050 supported. |

1051 Where an option is not supported, the associated utilities, functions, or facilities need not be  
 1052 present.

1053 Margin codes are defined for each option (see Section 1.5.1 (on page 6)).

### 1054 2.1.6.1 System Interfaces

1055 Refer to `<unistd.h>`, **Constants for Options and Option Groups** (on page 398) for the list of |  
 1056 options. |

### 1057 2.1.6.2 Shell and Utilities

1058 Each of these symbols shall be considered valid names by the implementation. Refer to |  
 1059 `<unistd.h>`, **Constants for Options and Option Groups** (on page 398). |

1060 The literal names shown below apply only to the *getconf* utility.

#### 1061 CD POSIX2\_C\_DEV

1062 The system supports the C-Language Development Utilities option.

1063 The utilities in the C-Language Development Utilities option are used for the development  
 1064 of C-language applications, including compilation or translation of C source code and  
 1065 complex program generators for simple lexical tasks and processing of context-free  
 1066 grammars.

1067 The utilities listed below may be provided by a conforming system; however, any system  
 1068 claiming conformance to the C-Language Development Utilities option shall provide all of  
 1069 the utilities listed.

1070 *c99*  
 1071 *lex*  
 1072 *yacc*

1073		POSIX2_CHAR_TERM
1074		The system supports the Terminal Characteristics option. This value need not be present on
1075		a system not supporting the User Portability Utilities option.
1076		Where applicable, the dependency is noted within the description of the utility.
1077		This option applies only to systems supporting the User Portability Utilities option. If
1078		supported, then the system supports at least one terminal type capable of all operations
1079		described in IEEE Std 1003.1-200x; see Section 10.2 (on page 181).
1080	FD	POSIX2_FORT_DEV
1081		The system supports the FORTRAN Development Utilities option.
1082		The <i>fort77</i> FORTRAN compiler is the only utility in the FORTRAN Development Utilities
1083		option. This is used for the development of FORTRAN language applications, including
1084		compilation or translation of FORTRAN source code.
1085		The <i>fort77</i> utility may be provided by a conforming system; however, any system claiming
1086		conformance to the FORTRAN Development Utilities option shall provide the <i>fort77</i> utility.
1087	FR	POSIX2_FORT_RUN
1088		The system supports the FORTRAN Runtime Utilities option.
1089		The <i>asa</i> utility is the only utility in the FORTRAN Runtime Utilities option.
1090		The <i>asa</i> utility may be provided by a conforming system; however, any system claiming
1091		conformance to the FORTRAN Runtime Utilities option shall provide the <i>asa</i> utility.
1092		POSIX2_LOCALEDEF
1093		The system supports the Locale Creation Utilities option.
1094		If supported, the system supports the creation of locales as described in the <i>localedef</i> utility.
1095		The <i>localedef</i> utility may be provided by a conforming system; however, any system
1096		claiming conformance to the Locale Creation Utilities option shall provide the <i>localedef</i>
1097		utility.
1098	BE	POSIX2_PBS
1099		The system supports the Batch Environment Services and Utilities option (see the Shell and
1100		Utilities volume of IEEE Std 1003.1-200x, Chapter 3, Batch Environment Services).
1101		<b>Note:</b> The Batch Environment Services and Utilities option is a combination of mandatory and
1102		optional batch services and utilities. The POSIX_PBS symbolic constant implies the
1103		system supports all the mandatory batch services and utilities.
1104		POSIX2_PBS_ACCOUNTING
1105		The system supports the Batch Accounting option.
1106		POSIX2_PBS_CHECKPOINT
1107		The system supports the Batch Checkpoint/Restart option.
1108		POSIX2_PBS_LOCATE
1109		The system supports the Locate Batch Job Request option.
1110		POSIX2_PBS_MESSAGE
1111		The system supports the Batch Job Message Request option.
1112		POSIX2_PBS_TRACK
1113		The system supports the Track Batch Job Request option.
1114	SD	POSIX2_SW_DEV
1115		The system supports the Software Development Utilities option.

1116 The utilities in the Software Development Utilities option are used for the development of  
 1117 applications, including compilation or translation of source code, the creation and  
 1118 maintenance of library archives, and the maintenance of groups of inter-dependent  
 1119 programs.

1120 The utilities listed below may be provided by the conforming system; however, any system  
 1121 claiming conformance to the Software Development Utilities option shall provide all of the  
 1122 utilities listed here.

1123 *ar*  
 1124 *make*  
 1125 *nm*  
 1126 *strip*

1127 UP POSIX2\_UPE

1128 The system supports the User Portability Utilities option.

1129 The utilities in the User Portability Utilities option shall be implemented on all systems that  
 1130 claim conformance to this option. Certain utilities are noted as having features that cannot  
 1131 be implemented on all terminal types; if the POSIX2\_CHAR\_TERM option is supported, the  
 1132 system shall support all such features on at least one terminal type; see Section 10.2 (on  
 1133 page 181).

1134 Some of the utilities are required only on systems that also support the Software  
 1135 Development Utilities option, or the character-at-a-time terminal option (see Section 10.2  
 1136 (on page 181)); such utilities have this noted in their DESCRIPTION sections. All of the  
 1137 other utilities listed are required only on systems that claim conformance to the User  
 1138 Portability Utilities option.

1139	<i>alias</i>	<i>expand</i>	<i>nm</i>	<i>unalias</i>
1140	<i>at</i>	<i>fc</i>	<i>patch</i>	<i>unexpand</i>
1141	<i>batch</i>	<i>fg</i>	<i>ps</i>	<i>uudecode</i>
1142	<i>bg</i>	<i>file</i>	<i>renice</i>	<i>uuencode</i>
1143	<i>crontab</i>	<i>jobs</i>	<i>split</i>	<i>vi</i>
1144	<i>split</i>	<i>man</i>	<i>strings</i>	<i>who</i>
1145	<i>ctags</i>	<i>mesg</i>	<i>tabs</i>	<i>write</i>
1146	<i>df</i>	<i>more</i>	<i>talk</i>	
1147	<i>du</i>	<i>newgrp</i>	<i>time</i>	
1148	<i>ex</i>	<i>nice</i>	<i>tput</i>	

## 1149 2.2 Application Conformance

1150 All applications claiming conformance to IEEE Std 1003.1-200x shall use only language-  
 1151 dependent services for the C programming language described in Section 2.3 (on page 31), shall  
 1152 use only the utilities and facilities defined in the Shell and Utilities volume of  
 1153 IEEE Std 1003.1-200x, and shall fall within one of the following categories.

### 1154 2.2.1 Strictly Conforming POSIX Application

1155 A Strictly Conforming POSIX Application is an application that requires only the facilities  
1156 described in IEEE Std 1003.1-200x. Such an application:

- 1157 1. Shall accept any implementation behavior that results from actions it takes in areas  
1158 described in IEEE Std 1003.1-200x as *implementation-defined* or *unspecified*, or where  
1159 IEEE Std 1003.1-200x indicates that implementations may vary
- 1160 2. Shall not perform any actions that are described as producing *undefined* results
- 1161 3. For symbolic constants, shall accept any value in the range permitted by  
1162 IEEE Std 1003.1-200x, but shall not rely on any value in the range being greater than the  
1163 minimums listed or being less than the maximums listed in IEEE Std 1003.1-200x
- 1164 4. Shall not use facilities designated as *obsolescent*
- 1165 5. Is required to tolerate and permitted to adapt to the presence or absence of optional  
1166 facilities whose availability is indicated by Section 2.1.3 (on page 16)
- 1167 6. For the C programming language, shall not produce any output dependent on any  
1168 behavior described in the ISO/IEC 9899:1999 standard as *unspecified*, *undefined*, or  
1169 *implementation-defined*, unless the System Interfaces volume of IEEE Std 1003.1-200x  
1170 specifies the behavior
- 1171 7. For the C programming language, shall not exceed any minimum implementation limit  
1172 defined in the ISO/IEC 9899:1999 standard, unless the System Interfaces volume of  
1173 IEEE Std 1003.1-200x specifies a higher minimum implementation limit
- 1174 8. For the C programming language, shall define `_POSIX_C_SOURCE` to be 200xxxL before  
1175 any header is included

1176 Within IEEE Std 1003.1-200x, any restrictions placed upon a Conforming POSIX Application  
1177 shall restrict a Strictly Conforming POSIX Application.

### 1178 2.2.2 Conforming POSIX Application

#### 1179 2.2.2.1 ISO/IEC Conforming POSIX Application

1180 An ISO/IEC Conforming POSIX Application is an application that uses only the facilities  
1181 described in IEEE Std 1003.1-200x and approved Conforming Language bindings for any ISO or  
1182 IEC standard. Such an application shall include a statement of conformance that documents all  
1183 options and limit dependencies, and all other ISO or IEC standards used.

#### 1184 2.2.2.2 <National Body> Conforming POSIX Application

1185 A <National Body> Conforming POSIX Application differs from an ISO/IEC Conforming  
1186 POSIX Application in that it also may use specific standards of a single ISO/IEC member body  
1187 referred to here as <National Body>. Such an application shall include a statement of  
1188 conformance that documents all options and limit dependencies, and all other <National Body>  
1189 standards used.

**1190 2.2.3 Conforming POSIX Application Using Extensions**

1191 A Conforming POSIX Application Using Extensions is an application that differs from a  
1192 Conforming POSIX Application only in that it uses non-standard facilities that are consistent  
1193 with IEEE Std 1003.1-200x. Such an application shall fully document its requirements for these  
1194 extended facilities, in addition to the documentation required of a Conforming POSIX  
1195 Application. A Conforming POSIX Application Using Extensions shall be either an ISO/IEC  
1196 Conforming POSIX Application Using Extensions or a <National Body> Conforming POSIX  
1197 Application Using Extensions (see Section 2.2.2.1 (on page 29) and Section 2.2.2.2 (on page 29)).

**1198 2.2.4 Strictly Conforming XSI Application**

1199 A Strictly Conforming XSI Application is an application that requires only the facilities described  
1200 in IEEE Std 1003.1-200x. Such an application:

- 1201 1. Shall accept any implementation behavior that results from actions it takes in areas  
1202 described in IEEE Std 1003.1-200x as *implementation-defined* or *unspecified*, or where  
1203 IEEE Std 1003.1-200x indicates that implementations may vary
- 1204 2. Shall not perform any actions that are described as producing *undefined* results
- 1205 3. For symbolic constants, shall accept any value in the range permitted by  
1206 IEEE Std 1003.1-200x, but shall not rely on any value in the range being greater than the  
1207 minimums listed or being less than the maximums listed in IEEE Std 1003.1-200x
- 1208 4. Shall not use facilities designated as *obsolescent*
- 1209 5. Is required to tolerate and permitted to adapt to the presence or absence of optional  
1210 facilities whose availability is indicated by Section 2.1.4 (on page 19)
- 1211 6. For the C programming language, shall not produce any output dependent on any  
1212 behavior described in the ISO C standard as *unspecified*, *undefined*, or *implementation-*  
1213 *defined*, unless the System Interfaces volume of IEEE Std 1003.1-200x specifies the behavior
- 1214 7. For the C programming language, shall not exceed any minimum implementation limit  
1215 defined in the ISO C standard, unless the System Interfaces volume of  
1216 IEEE Std 1003.1-200x specifies a higher minimum implementation limit
- 1217 8. For the C programming language, shall define `_XOPEN_SOURCE` to be 600 before any  
1218 header is included

1219 Within IEEE Std 1003.1-200x, any restrictions placed upon a Conforming POSIX Application  
1220 shall restrict a Strictly Conforming XSI Application.

**1221 2.2.5 Conforming XSI Application Using Extensions**

1222 A Conforming XSI Application Using Extensions is an application that differs from a Strictly  
1223 Conforming XSI Application only in that it uses non-standard facilities that are consistent with  
1224 IEEE Std 1003.1-200x. Such an application shall fully document its requirements for these  
1225 extended facilities, in addition to the documentation required of a Strictly Conforming XSI  
1226 Application.

**1227 2.3 Language-Dependent Services for the C Programming Language**

1228 Implementors seeking to claim conformance using the ISO C standard shall claim POSIX  
1229 conformance as described in Section 2.1.3 (on page 16).

**1230 2.4 Other Language-Related Specifications**

1231 IEEE Std 1003.1-200x is currently specified in terms of the shell command language and ISO C.  
1232 Bindings to other programming languages are being developed.

1233 If conformance to IEEE Std 1003.1-200x is claimed for implementation of any programming  
1234 language, the implementation of that language shall support the use of external symbols distinct  
1235 to at least 31 bytes in length in the source program text. (That is, identifiers that differ at or  
1236 before the thirty-first byte shall be distinct.) If a national or international standard governing a  
1237 language defines a maximum length that is less than this value, the language-defined maximum  
1238 shall be supported. External symbols that differ only by case shall be distinct when the character  
1239 set in use distinguishes uppercase and lowercase characters and the language permits (or  
1240 requires) uppercase and lowercase characters to be distinct in external symbols.



**Definitions**

1242

1243 For the purposes of IEEE Std 1003.1-200x, the terms and definitions given in Chapter 3 apply.

1244 **Note:** No shading to denote extensions or options occurs in this chapter. Where the terms and  
 1245 definitions given in this chapter are used elsewhere in text related to extensions and options,  
 1246 they are shaded as appropriate.

### 1247 **3.1 Abortive Release**

1248 An abrupt termination of a network connection that may result in the loss of data.

### 1249 **3.2 Absolute Pathname**

1250 A pathname beginning with a single or more than two slashes; see also Section 3.266 (on page  
 1251 69).

1252 **Note:** Pathname Resolution is defined in detail in Section 4.11 (on page 98).

### 1253 **3.3 Access Mode**

1254 A particular form of access permitted to a file. |

### 1255 **3.4 Additional File Access Control Mechanism** |

1256 An implementation-defined mechanism that is layered upon the access control mechanisms |  
 1257 defined here, but which do not grant permissions beyond those defined herein, although they |  
 1258 may further restrict them.

1259 **Note:** File Access Permissions are defined in detail in Section 4.4 (on page 95).

### 1260 **3.5 Address Space**

1261 The memory locations that can be referenced by a process or the threads of a process.

### 1262 **3.6 Advisory Information**

1263 An interface that advises the implementation on (portable) application behavior so that it can  
 1264 optimize the system. |

### 1265 **3.7 Affirmative Response** |

1266 An input string that matches one of the responses acceptable to the *LC\_MESSAGES* category |  
 1267 keyword **yesexpr**, matching an extended regular expression in the current locale.

1268        **Note:**     The *LC\_MESSAGES* category is defined in detail in Section 7.3.6 (on page 148).

### 1269    **3.8    Alert**

1270        To cause the user's terminal to give some audible or visual indication that an error or some other  
1271        event has occurred. When the standard output is directed to a terminal device, the method for  
1272        alerting the terminal user is unspecified. When the standard output is not directed to a terminal  
1273        device, the alert is accomplished by writing the <alert> to standard output (unless the utility  
1274        description indicates that the use of standard output produces undefined results in this case).

### 1275    **3.9    Alert Character (<alert>)**

1276        A character that in the output stream should cause a terminal to alert its user via a visual or  
1277        audible notification. It is the character designated by '\a' in the C language. It is unspecified  
1278        whether this character is the exact sequence transmitted to an output device by the system to  
1279        accomplish the alert function.

### 1280    **3.10   Alias Name**

1281        In the shell command language, a word consisting solely of underscores, digits, and alphabetic  
1282        from the portable character set and any of the following characters: '!', '%', ', ', '@'.

1283        Implementations may allow other characters within alias names as an extension.

1284        **Note:**     The portable character set is defined in detail in Section 6.1 (on page 111).

### 1285    **3.11   Alignment**

1286        A requirement that objects of a particular type be located on storage boundaries with addresses  
1287        that are particular multiples of a byte address.

1288        **Note:**     See also the ISO C standard, Section B3.

### 1289    **3.12   Alternate File Access Control Mechanism**

1290        An implementation-defined mechanism that is independent of the access control mechanisms  
1291        defined herein, and which if enabled on a file may either restrict or extend the permissions of a  
1292        given user. IEEE Std 1003.1-200x defines when such mechanisms can be enabled and when they  
1293        are disabled.

1294        **Note:**     File Access Permissions are defined in detail in Section 4.4 (on page 95).

### 1295    **3.13   Alternate Signal Stack**

1296        Memory associated with a thread, established upon request by the implementation for a thread,  
1297        separate from the thread signal stack, in which signal handlers responding to signals sent to that  
1298        thread may be executed.

**1299 3.14 Ancillary Data** |

1300 Protocol-specific, local system-specific, or optional information. The information can be both |  
1301 local or end-to-end significant, header information, part of a data portion, protocol-specific, and |  
1302 implementation or system-specific. |

**1303 3.15 Angle Brackets** |

1304 The characters '`<`' (left-angle-bracket) and '`>`' (right-angle-bracket). When used in the phrase |  
1305 "enclosed in angle brackets", the symbol '`<`' immediately precedes the object to be enclosed, |  
1306 and '`>`' immediately follows it. When describing these characters in the portable character set, |  
1307 the names `<less-than-sign>` and `<greater-than-sign>` are used. |

**1308 3.16 Application**

1309 A computer program that performs some desired function.

**1310 3.17 Application Address**

1311 Endpoint address of a specific application.

**1312 3.18 Application Program Interface (API)**

1313 The definition of syntax and semantics for providing computer system services. |

**1314 3.19 Appropriate Privileges** |

1315 An implementation-defined means of associating privileges with a process with regard to the |  
1316 function calls, function call options, and the commands that need special privileges. There may |  
1317 be zero or more such means. These means (or lack thereof) are described in the conformance |  
1318 document. |

1319 **Note:** Function calls are defined in the System Interfaces volume of IEEE Std 1003.1-200x, and |  
1320 commands are defined in the Shell and Utilities volume of IEEE Std 1003.1-200x. |

**1321 3.20 Argument** |

1322 In the shell command language, a parameter passed to a utility as the equivalent of a single |  
1323 string in the `argv` array created by one of the `exec` functions. An argument is one of the options, |  
1324 option-arguments, or operands following the command name. |

1325 **Note:** The Utility Argument Syntax is defined in detail in Section 12.1 (on page 197) and the Shell and |  
1326 Utilities volume of IEEE Std 1003.1-200x, Section 2.9.1.1, Command Search and Execution. |

1327 In the C language, an expression in a function call expression or a sequence of preprocessing |  
1328 tokens in a function-like macro invocation. |

**1329 3.21 Arm (a Timer)**

1330 To start a timer measuring the passage of time, enabling notifying a process when the specified  
1331 time or time interval has passed.

**1332 3.22 Asterisk**

1333 The character ' \* '.

**1334 3.23 Async-Cancel-Safe Function**

1335 A function that may be safely invoked by an application while the asynchronous form of  
1336 cancelation is enabled. No function is async-cancel-safe unless explicitly described as such.

**1337 3.24 Asynchronous Events**

1338 Events that occur independently of the execution of the application.

**1339 3.25 Asynchronous Input and Output**

1340 A functionality enhancement to allow an application process to queue data input and output  
1341 commands with asynchronous notification of completion. |

**1342 3.26 Async-Signal-Safe Function**

1343 A function that may be invoked, without restriction, from signal-catching functions. No function  
1344 is async-signal-safe unless explicitly described as such. |

**1345 3.27 Asynchronously-Generated Signal** |

1346 A signal that is not attributable to a specific thread. Examples are signals sent via *kill()*, signals |  
1347 sent from the keyboard, and signals delivered to process groups. Being asynchronous is a  
1348 property of how the signal was generated and not a property of the signal number. All signals  
1349 may be generated asynchronously.

1350 **Note:** The *kill()* function is defined in detail in the System Interfaces volume of IEEE Std 1003.1-200x.

**1351 3.28 Asynchronous I/O Operation** |

1352 An I/O operation that does not of itself cause the thread requesting the I/O to be blocked from |  
1353 further use of the processor.

1354 This implies that the process and the I/O operation may be running concurrently.

1355 **3.29 Asynchronous I/O Completion**

1356 For an asynchronous read or write operation, when a corresponding synchronous read or write  
1357 would have completed and when any associated status fields have been updated.

1358 **3.30 Authentication**

1359 The process of validating a user or process to verify that the user or process is not a counterfeit. |

1360 **3.31 Authorization** |

1361 The process of verifying that a user or process has permission to use a resource in the manner |  
1362 requested.

1363 To ensure security, the user or process would also need to be authenticated before granting  
1364 access.

1365 **3.32 Background Job**

1366 See *Background Process Group* in Section 3.34.

1367 **3.33 Background Process**

1368 A process that is a member of a background process group.

1369 **3.34 Background Process Group (or Background Job)**

1370 Any process group, other than a foreground process group, that is a member of a session that  
1371 has established a connection with a controlling terminal.

1372 **3.35 Backquote**

1373 The character ' ` ', also known as a *grave accent*.

1374 **3.36 Backslash**

1375 The character ' \ ', also known as a *reverse solidus*. |

1376 **3.37 Backspace Character (<backspace>)** |

1377 A character that, in the output stream, should cause printing (or displaying) to occur one column |  
1378 position previous to the position about to be printed. If the position about to be printed is at the  
1379 beginning of the current line, the behavior is unspecified. It is the character designated by ' \b ' |  
1380 in the C language. It is unspecified whether this character is the exact sequence transmitted to an  
1381 output device by the system to accomplish the backspace function. The <backspace> defined

1382 here is not necessarily the ERASE special character.

1383 **Note:** Special Characters are defined in detail in Section 11.1.9 (on page 187).

### 1384 **3.38 Barrier**

1385 A synchronization object that allows multiple threads to synchronize at a particular point in  
1386 their execution.

### 1387 **3.39 Base Character**

1388 One of the set of characters defined in the Latin alphabet. In Western European languages other  
1389 than English, these characters are commonly used with diacritical marks (accents, cedilla, and so  
1390 on) to extend the range of characters in an alphabet.

### 1391 **3.40 Basename**

1392 The final, or only, filename in a pathname. |

### 1393 **3.41 Basic Regular Expression (BRE)** |

1394 A regular expression (see Section 3.316 (on page 76)) used by the majority of utilities that select |  
1395 strings from a set of character strings. |

1396 **Note:** Basic Regular Expressions are described in detail in Section 9.3 (on page 167). |

### 1397 **3.42 Batch Access List** |

1398 A list of user IDs and group IDs of those users and groups authorized to place batch jobs in a |  
1399 batch queue. |

1400 A batch access list is associated with a batch queue. A batch server uses the batch access list of a  
1401 batch queue as one of the criteria in deciding to put a batch job in a batch queue.

### 1402 **3.43 Batch Administrator**

1403 A user that is authorized to modify all the attributes of queues and jobs and to change the status |  
1404 of a batch server. |

### 1405 **3.44 Batch Client** |

1406 A computational entity that utilizes batch services by making requests of batch servers. |

1407 Batch clients often provide the means by which users access batch services, although a batch  
1408 server may act as a batch client by virtue of making requests of another batch server. |

**1409 3.45 Batch Destination** |

1410 The batch server in a batch system to which a batch job should be sent for processing. |

1411 Acceptance of a batch job at a batch destination is the responsibility of a receiving batch server.  
1412 A batch destination may consist of a batch server-specific portion, a network-wide portion, or  
1413 both. The batch server-specific portion is referred to as the *batch queue*. The network-wide  
1414 portion is referred to as a *batch server name*. |

**1415 3.46 Batch Destination Identifier** |

1416 A string that identifies a specific batch destination. |

1417 A string of characters in the portable character set used to specify a particular batch destination. |

1418 **Note:** The portable character set is defined in detail in Section 6.1 (on page 111). |

**1419 3.47 Batch Directive** |

1420 A line from a file that is interpreted by the batch server. The line is usually in the form of a  
1421 comment and is an additional means of passing options to the *qsub* utility. |

1422 **Note:** The *qsub* utility is defined in detail in the Shell and Utilities volume of IEEE Std 1003.1-200x. |

**1423 3.48 Batch Job** |

1424 A set of computational tasks for a computing system. |

1425 Batch jobs are managed by batch servers. |

1426 Once created, a batch job may be executing or pending execution. A batch job that is executing  
1427 has an associated session leader (a process) that initiates and monitors the computational tasks  
1428 of the batch job. |

**1429 3.49 Batch Job Attribute** |

1430 A named data type whose value affects the processing of a batch job. |

1431 The values of the attributes of a batch job affect the processing of that job by the batch server  
1432 that manages the batch job. |

**1433 3.50 Batch Job Identifier** |

1434 A unique name for a batch job. A name that is unique among all other batch job identifiers in a  
1435 batch system and that identifies the batch server to which the batch job was originally  
1436 submitted. |

**1437 3.51 Batch Job Name** |

1438 A label that is an attribute of a batch job. The batch job name is not necessarily unique. |

**1439 3.52 Batch Job Owner** |

1440 The *username@hostname* of the user submitting the batch job, where *username* is a user name (see |  
1441 also Section 3.426 (on page 91)) and *hostname* is a network host name. |

**1442 3.53 Batch Job Priority** |

1443 A value specified by the user that may be used by an implementation to determine the order in |  
1444 which batch jobs are selected to be executed. Job priority has a numeric value in the range -1 024 |  
1445 to 1 023. |

1446 **Note:** The batch job priority is not the execution priority (nice value) of the batch job.

**1447 3.54 Batch Job State** |

1448 An attribute of a batch job which determines the types of requests that the batch server that |  
1449 manages the batch job can accept for the batch job. Valid states include QUEUED, RUNNING, |  
1450 HELD, WAITING, EXITING, and TRANSITING. |

**1451 3.55 Batch Name Service**

1452 A service that assigns batch names that are unique within the batch name space, and that can |  
1453 translate a unique batch name into the location of the named batch entity.

**1454 3.56 Batch Name Space**

1455 The environment within which a batch name is known to be unique. |

**1456 3.57 Batch Node** |

1457 A host containing part or all of a batch system. |

1458 A batch node is a host meeting at least one of the following conditions:

- 1459 • Capable of executing a batch client
- 1460 • Contains a routing batch queue
- 1461 • Contains an execution batch queue

**1462 3.58 Batch Operator**

1463 A user that is authorized to modify some, but not all, of the attributes of jobs and queues, and |  
1464 may change the status of the batch server. |

**1465 3.59 Batch Queue** |

1466 A manageable object that represents a set of batch jobs and is managed by a single batch server. |

1467       **Note:**     A set of batch jobs is called a batch queue largely for historical reasons. Jobs are selected from |  
1468                   the batch queue for execution based on attributes such as priority, resource requirements, and |  
1469                   hold conditions. |  
1470                   See also the Shell and Utilities volume of IEEE Std 1003.1-200x, Section 3.1.2, Batch Queues. |

### 1471 **3.60   Batch Queue Attribute**

1472                   A named data type whose value affects the processing of all batch jobs that are members of the |  
1473                   batch queue. |

1474                   A batch queue has attributes that affect the processing of batch jobs that are members of the |  
1475                   batch queue. |

### 1476 **3.61   Batch Queue Position**

1477                   The place, relative to other jobs in the batch queue, occupied by a particular job in a batch queue. |  
1478                   This is defined in part by submission time and priority; see also Section 3.62. |

### 1479 **3.62   Batch Queue Priority**

1480                   The maximum job priority allowed for any batch job in a given batch queue. |

1481                   The batch queue priority is set and may be changed by users with appropriate privilege. The |  
1482                   priority is bounded in an implementation-defined manner. |

### 1483 **3.63   Batch Rerunability**

1484                   An attribute of a batch job indicating that it may be rerun after an abnormal termination from |  
1485                   the beginning without affecting the validity of the results. |

### 1486 **3.64   Batch Restart**

1487                   The action of resuming the processing of a batch job from the point of the last checkpoint. |  
1488                   Typically, this is done if the batch job has been interrupted because of a system failure. |

### 1489 **3.65   Batch Server**

1490                   A computational entity that provides batch services. |

### 1491 **3.66   Batch Server Name**

1492                   A string of characters in the portable character set used to specify a particular server in a |  
1493                   network. |

1494       **Note:**     The portable character set is defined in detail in Section 6.1 (on page 111). |

**1495 3.67 Batch Service** |

1496 Computational and organizational services performed by a batch system on behalf of batch jobs. |

1497 Batch services are of two types: *requested* and *deferred*.

1498 **Note:** Batch Services are listed in the Shell and Utilities volume of IEEE Std 1003.1-200x, Table 3-5,  
1499 Batch Services Summary.

**1500 3.68 Batch Service Request** |

1501 A solicitation of services from a batch client to a batch server. |

1502 A batch service request may entail the exchange of any number of messages between the batch  
1503 client and the batch server.

1504 When naming specific types of service requests, the term request is qualified by the type of  
1505 request, as in *Queue Batch Job Request* and *Delete Batch Job Request*.

**1506 3.69 Batch Submission**

1507 The process by which a batch client requests that a batch server create a batch job via a *Queue Job*  
1508 *Request* to perform a specified computational task.

**1509 3.70 Batch System**

1510 A collection of one or more batch servers.

**1511 3.71 Batch Target User**

1512 The name of a user on the batch destination batch server.

1513 The target user is the user name under whose account the batch job is to execute on the  
1514 destination batch server.

**1515 3.72 Batch User**

1516 A user who is authorized to make use of batch services. |

**1517 3.73 Bind**

1518 The process of assigning a network address to an endpoint. |

**1519 3.74 Blank Character (<blank>)** |

1520 One of the characters that belong to the **blank** character class as defined via the *LC\_CTYPE* |  
1521 category in the current locale. In the POSIX locale, a <blank> is either a <tab> or a <space>.

**1522 3.75 Blank Line**

1523 A line consisting solely of zero or more <blank>s terminated by a <newline>; see also Section  
1524 3.144 (on page 52).

**1525 3.76 Blocked Process (or Thread)**

1526 A process (or thread) that is waiting for some condition (other than the availability of a  
1527 processor) to be satisfied before it can continue execution.

**1528 3.77 Blocking**

1529 A property of an open file description that causes function calls associated with it to wait for the  
1530 requested action to be performed before returning.

**1531 3.78 Block-Mode Terminal**

1532 A terminal device operating in a mode incapable of the character-at-a-time input and output  
1533 operations described by some of the standard utilities.

1534 **Note:** Output Devices and Terminal Types are defined in detail in Section 10.2 (on page 181).

**1535 3.79 Block Special File**

1536 A file that refers to a device. A block special file is normally distinguished from a character  
1537 special file by providing access to the device in a manner such that the hardware characteristics  
1538 of the device are not visible.

**1539 3.80 Braces**

1540 The characters ‘{’ (left brace) and ‘}’ (right brace), also known as *curly braces*. When used in  
1541 the phrase “enclosed in (curly) braces” the symbol ‘{’ immediately precedes the object to be  
1542 enclosed, and ‘}’ immediately follows it. When describing these characters in the portable  
1543 character set, the names <left-brace> and <right-brace> are used.

**1544 3.81 Brackets**

1545 The characters ‘[’ (left-bracket) and ‘]’ (right-bracket), also known as *square brackets*. When  
1546 used in the phrase “enclosed in (square) brackets” the symbol ‘[’ immediately precedes the  
1547 object to be enclosed, and ‘]’ immediately follows it. When describing these characters in the  
1548 portable character set, the names <left-square-bracket> and <right-square-bracket> are used.

**1549 3.82 Broadcast**

1550 The transfer of data from one endpoint to several endpoints, as described in RFC 919 and  
1551 RFC 922.

**1552 3.83 Built-In Utility (or Built-In)**

1553 A utility implemented within a shell. The utilities referred to as *special built-ins* have special  
1554 qualities. Unless qualified, the term built-in includes the special built-in utilities. *Regular built-ins*  
1555 are not required to be actually built into the shell on the implementation, but they do have  
1556 special command-search qualities.

1557 **Note:** Special Built-In Utilities are defined in detail in the Shell and Utilities volume of  
1558 IEEE Std 1003.1-200x, Section 2.14, Special Built-In Utilities.

1559 Regular Built-In Utilities are defined in detail in the Shell and Utilities volume of  
1560 IEEE Std 1003.1-200x, Section 2.9.1.1, Command Search and Execution.

**1561 3.84 Byte**

1562 An individually addressable unit of data storage that is exactly an octet, used to store a character  
1563 or a portion of a character; see also Section 3.87. A byte is composed of a contiguous sequence of  
1564 8 bits. The least significant bit is called the *low-order* bit; the most significant is called the *high-*  
1565 *order* bit.

1566 **Note:** The definition of byte from the ISO C standard is broader than the above and might  
1567 accommodate hardware architectures with different sized addressable units than octets.

**1568 3.85 Byte Input/Output Functions**

1569 The functions that perform byte-oriented input from streams or byte-oriented output to streams:  
1570 *fgetc()*, *fgets()*, *fprintf()*, *fputc()*, *fputs()*, *fread()*, *fscanf()*, *fwrite()*, *getc()*, *getchar()*, *gets()*, *printf()*,  
1571 *putc()*, *putchar()*, *puts()*, *scanf()*, *ungetc()*, *vfprintf()*, and *vprintf()*.

1572 **Note:** Functions are defined in detail in the System Interfaces volume of IEEE Std 1003.1-200x.

**1573 3.86 Carriage-Return Character (<carriage-return>)**

1574 A character that in the output stream indicates that printing should start at the beginning of the  
1575 same physical line in which the <carriage-return> occurred. It is the character designated by  
1576 '\r' in the C language. It is unspecified whether this character is the exact sequence  
1577 transmitted to an output device by the system to accomplish the movement to the beginning of  
1578 the line.

**1579 3.87 Character**

1580 A sequence of one or more bytes representing a single graphic symbol or control code.

1581 **Note:** This term corresponds to the ISO C standard term multi-byte character, where a single-byte  
1582 character is a special case of a multi-byte character. Unlike the usage in the ISO C standard,  
1583 *character* here has no necessary relationship with storage space, and *byte* is used when storage  
1584 space is discussed.

1585 See the definition of the portable character set in Section 6.1 (on page 111) for a further  
1586 explanation of the graphical representations of (abstract) characters, as opposed to character  
1587 encodings.

**1588 3.88 Character Array**

1589 An array of elements of type **char**. |

**1590 3.89 Character Class**

1591 A named set of characters sharing an attribute associated with the name of the class. The classes |  
1592 and the characters that they contain are dependent on the value of the *LC\_CTYPE* category in the |  
1593 current locale.

1594 **Note:** The *LC\_CTYPE* category is defined in detail in Section 7.3.1 (on page 122).

**1595 3.90 Character Set**

1596 A finite set of different characters used for the representation, organization, or control of data. |

**1597 3.91 Character Special File**

1598 A file that refers to a device. One specific type of character special file is a terminal device file. |

1599 **Note:** The General Terminal Interface is defined in detail in Chapter 11 (on page 183).

**1600 3.92 Character String**

1601 A contiguous sequence of characters terminated by and including the first null byte. |

**1602 3.93 Child Process**

1603 A new process created (by *fork()* or *spawn()*) by a given process. A child process remains the |  
1604 child of the creating process as long as both processes continue to exist.

1605 **Note:** The *fork()* and *spawn()* functions are defined in detail in the System Interfaces volume of |  
1606 IEEE Std 1003.1-200x.

**1607 3.94 Circumflex**

1608 The character '^'. |

**1609 3.95 Clock**

1610 A software or hardware object that can be used to measure the apparent or actual passage of |  
1611 time.

1612 The current value of the time measured by a clock can be queried and, possibly, set to a value |  
1613 within the legal range of the clock.

**1614 3.96 Clock Jump**

1615 The difference between two successive distinct values of a clock, as observed from the  
1616 application via one of the “get time” operations.

**1617 3.97 Clock Tick**

1618 An interval of time; an implementation-defined number of these occur each second. Clock ticks  
1619 are one of the units that may be used to express a value found in type `clock_t`.

**1620 3.98 Coded Character Set**

1621 A set of unambiguous rules that establishes a character set and the one-to-one relationship  
1622 between each character of the set and its bit representation.

**1623 3.99 Codeset**

1624 The result of applying rules that map a numeric code value to each element of a character set. An  
1625 element of a character set may be related to more than one numeric code value but the reverse is  
1626 not true. However, for state-dependent encodings the relationship between numeric code values  
1627 to elements of a character set may be further controlled by state information. The character set  
1628 may contain fewer elements than the total number of possible numeric code values; that is, some  
1629 code values may be unassigned.

1630 **Note:** Character Encoding is defined in detail in Section 6.2 (on page 114).

**1631 3.100 Collating Element**

1632 The smallest entity used to determine the logical ordering of character or wide-character strings;  
1633 see also Section 3.102. A collating element consists of either a single character, or two or more  
1634 characters collating as a single entity. The value of the `LC_COLLATE` category in the current  
1635 locale determines the current set of collating elements.

**1636 3.101 Collation**

1637 The logical ordering of character or wide-character strings according to defined precedence  
1638 rules. These rules identify a collation sequence between the collating elements, and such  
1639 additional rules that can be used to order strings consisting of multiple collating elements.

**1640 3.102 Collation Sequence**

1641 The relative order of collating elements as determined by the setting of the `LC_COLLATE`  
1642 category in the current locale. The collation sequence is used for sorting and is determined from  
1643 the collating weights assigned to each collating element. In the absence of weights, the collation  
1644 sequence is the order in which collating elements are specified between `order_start` and  
1645 `order_end` keywords in the `LC_COLLATE` category.

1646 Multi-level sorting is accomplished by assigning elements one or more collation weights, up to  
 1647 the limit {COLL\_WEIGHTS\_MAX}. On each level, elements may be given the same weight (at  
 1648 the primary level, called an equivalence class; see also Section 3.150 (on page 53)) or be omitted  
 1649 from the sequence. Strings that collate equally using the first assigned weight (primary ordering)  
 1650 are then compared using the next assigned weight (secondary ordering), and so on.

1651 **Note:** {COLL\_WEIGHTS\_MAX} is defined in detail in <limits.h>.

### 1652 3.103 Column Position |

1653 A unit of horizontal measure related to characters in a line. |

1654 It is assumed that each character in a character set has an intrinsic column width independent of  
 1655 any output device. Each printable character in the portable character set has a column width of  
 1656 one. The standard utilities, when used as described in IEEE Std 1003.1-200x, assume that all  
 1657 characters have integral column widths. The column width of a character is not necessarily  
 1658 related to the internal representation of the character (numbers of bits or bytes).

1659 The column position of a character in a line is defined as one plus the sum of the column widths  
 1660 of the preceding characters in the line. Column positions are numbered starting from 1.

### 1661 3.104 Command

1662 A directive to the shell to perform a particular task.

1663 **Note:** Shell Commands are defined in detail in the Shell and Utilities volume of IEEE Std 1003.1-200x,  
 1664 Section 2.9, Shell Commands.

### 1665 3.105 Command Language Interpreter |

1666 An interface that interprets sequences of text input as commands. It may operate on an input  
 1667 stream or it may interactively prompt and read commands from a terminal. It is possible for  
 1668 applications to invoke utilities through a number of interfaces, which are collectively considered  
 1669 to act as command interpreters. The most obvious of these are the *sh* utility and the *system()*  
 1670 function, although *popen()* and the various forms of *exec* may also be considered to behave as  
 1671 interpreters.

1672 **Note:** The *sh* utility is defined in detail in the Shell and Utilities volume of IEEE Std 1003.1-200x.

1673 The *system()*, *popen()*, and *exec* functions are defined in detail in the System Interfaces volume  
 1674 of IEEE Std 1003.1-200x.

### 1675 3.106 Composite Graphic Symbol |

1676 A graphic symbol consisting of a combination of two or more other graphic symbols in a single  
 1677 character position, such as a diacritical mark and a base character. |

### 1678 3.107 Condition Variable |

1679 A synchronization object which allows a thread to suspend execution, repeatedly, until some  
 1680 associated predicate becomes true. A thread whose execution is suspended on a condition

1681 variable is said to be blocked on the condition variable.

### 1682 **3.108 Connection**

1683 An association established between two or more endpoints for the transfer of data

### 1684 **3.109 Connection Mode**

1685 The transfer of data in the context of a connection; see also Section 3.110.

### 1686 **3.110 Connectionless Mode**

1687 The transfer of data other than in the context of a connection; see also Section 3.109 and Section  
1688 3.123 (on page 49).

### 1689 **3.111 Control Character**

1690 A character, other than a graphic character, that affects the recording, processing, transmission,  
1691 or interpretation of text. |

### 1692 **3.112 Control Operator**

1693 In the shell command language, a token that performs a control function. It is one of the |  
1694 following symbols: |

1695 & && ( ) ; ;; newline | ||

1696 The end-of-input indicator used internally by the shell is also considered a control operator.

1697 **Note:** Token Recognition is defined in detail in the Shell and Utilities volume of IEEE Std 1003.1-200x,  
1698 Section 2.3, Token Recognition.

### 1699 **3.113 Controlling Process**

1700 The session leader that established the connection to the controlling terminal. If the terminal  
1701 subsequently ceases to be a controlling terminal for this session, the session leader ceases to be  
1702 the controlling process. |

### 1703 **3.114 Controlling Terminal**

1704 A terminal that is associated with a session. Each session may have at most one controlling |  
1705 terminal associated with it, and a controlling terminal is associated with exactly one session. |  
1706 Certain input sequences from the controlling terminal cause signals to be sent to all processes in  
1707 the process group associated with the controlling terminal.

1708 **Note:** The General Terminal Interface is defined in detail in Chapter 11 (on page 183).

**1709 3.115 Conversion Descriptor**

1710 A per-process unique value used to identify an open codeset conversion.

**1711 3.116 Core File**

1712 A file of unspecified format that may be generated when a process terminates abnormally. |

**1713 3.117 CPU Time (Execution Time)** |

1714 The time spent executing a process or thread, including the time spent executing system services |  
1715 on behalf of that process or thread. If the Threads option is supported, then the value of the |  
1716 CPU-time clock for a process is implementation-defined. With this definition the sum of all the |  
1717 execution times of all the threads in a process might not equal the process execution time, even |  
1718 in a single-threaded process, because implementations may differ in how they account for time |  
1719 during context switches or for other reasons.

**1720 3.118 CPU-Time Clock**

1721 A clock that measures the execution time of a particular process or thread.

**1722 3.119 CPU-Time Timer**

1723 A timer attached to a CPU-time clock.

**1724 3.120 Current Job**

1725 In the context of job control, the job that will be used as the default for the *fg* or *bg* utilities. There  
1726 is at most one current job; see also Section 3.203 (on page 60).

**1727 3.121 Current Working Directory**

1728 See *Working Directory* in Section 3.436 (on page 92).

**1729 3.122 Cursor Position**

1730 The line and column position on the screen denoted by the terminal's cursor.

**1731 3.123 Datagram**

1732 A unit of data transferred from one endpoint to another in connectionless mode service.

**1733 3.124 Data Segment**

1734 Memory associated with a process, that can contain dynamically allocated data. |

**1735 3.125 Deferred Batch Service** |

1736 A service that is performed as a result of events that are asynchronous with respect to requests. |

1737 **Note:** Once a batch job has been created, it is subject to deferred services.

**1738 3.126 Device**

1739 A computer peripheral or an object that appears to the application as such.

**1740 3.127 Device ID**

1741 A non-negative integer used to identify a device.

**1742 3.128 Directory**

1743 A file that contains directory entries. No two directory entries in the same directory have the  
1744 same name.

**1745 3.129 Directory Entry (or Link)**

1746 An object that associates a filename with a file. Several directory entries can associate names  
1747 with the same file.

**1748 3.130 Directory Stream**

1749 A sequence of all the directory entries in a particular directory. An open directory stream may be  
1750 implemented using a file descriptor.

**1751 3.131 Disarm (a Timer)**

1752 To stop a timer from measuring the passage of time, disabling any future process notifications  
1753 (until the timer is armed again).

**1754 3.132 Display**

1755 To output to the user's terminal. If the output is not directed to a terminal, the results are  
1756 undefined.

1757 **3.133 Display Line**

1758 A line of text on a physical device or an emulation thereof. Such a line will have a maximum  
1759 number of characters which can be presented.

1760 **Note:** This may also be written as “line on the display”.

1761 **3.134 Dollar Sign**

1762 The character ‘\$’.

1763 **3.135 Dot**

1764 In the context of naming files, the filename consisting of a single dot character (‘.’).

1765 **Note:** In the context of shell special built-in utilities, see *dot* in the Shell and Utilities volume of  
1766 IEEE Std 1003.1-200x, Section 2.14, Special Built-In Utilities.

1767 Pathname Resolution is defined in detail in Section 4.11 (on page 98).

1768 **3.136 Dot-Dot**

1769 The filename consisting solely of two dot characters (“..”).

1770 **Note:** Pathname Resolution is defined in detail in Section 4.11 (on page 98).

1771 **3.137 Double-Quote**

1772 The character ‘”’, also known as *quotation-mark*.

1773 **Note:** The *double* adjective in this term refers to the two strokes in the character glyph.  
1774 IEEE Std 1003.1-200x never uses the term double-quote to refer to two apostrophes or  
1775 quotation marks.

1776 **3.138 Downshifting**

1777 The conversion of an uppercase character that has a single-character lowercase representation  
1778 into this lowercase representation.

1779 **3.139 Driver**

1780 A module that controls data transferred to and received from devices.

1781 **Note:** Drivers are traditionally written to be a part of the system implementation, although they are  
1782 frequently written separately from the writing of the implementation. A driver may contain  
1783 processor-specific code, and therefore be non-portable.

**1784 3.140 Effective Group ID**

1785 An attribute of a process that is used in determining various permissions, including file access  
1786 permissions; see also Section 3.188 (on page 58).

**1787 3.141 Effective User ID**

1788 An attribute of a process that is used in determining various permissions, including file access  
1789 permissions; see also Section 3.425 (on page 91).

**1790 3.142 Eight-Bit Transparency**

1791 The ability of a software component to process 8-bit characters without modifying or utilizing  
1792 any part of the character in a way that is inconsistent with the rules of the current coded  
1793 character set.

**1794 3.143 Empty Directory**

1795 A directory that contains, at most, directory entries for dot and dot-dot, and has exactly one link |  
1796 to it, in dot-dot. No other links to the directory may exist. It is unspecified whether an |  
1797 implementation can ever consider the root directory to be empty.

**1798 3.144 Empty Line**

1799 A line consisting of only a <newline>; see also Section 3.75 (on page 43).

**1800 3.145 Empty String (or Null String)**

1801 A string whose first byte is a null byte.

**1802 3.146 Empty Wide-Character String**

1803 A wide-character string whose first element is a null wide-character code. |

**1804 3.147 Encoding Rule** |

1805 The rules used to convert between wide-character codes and multi-byte character codes. |

1806 **Note:** Stream Orientation and Encoding Rules are defined in detail in the System Interfaces volume  
1807 of IEEE Std 1003.1-200x, Section 2.5.2, Stream Orientation and Encoding Rules.

**1808 3.148 Entire Regular Expression** |

1809 The concatenated set of one or more basic regular expressions or extended regular expressions |  
1810 that make up the pattern specified for string selection.

1811           **Note:**     Regular Expressions are defined in detail in Chapter 9 (on page 165).

### 1812 **3.149 Epoch**

1813           The time zero hours, zero minutes, zero seconds, on January 1, 1970 Coordinated Universal Time  
1814           (UTC).

1815           **Note:**     See also Seconds Since the Epoch defined in Section 4.14 (on page 100).

### 1816 **3.150 Equivalence Class**

1817           A set of collating elements with the same primary collation weight.

1818           Elements in an equivalence class are typically elements that naturally group together, such as all  
1819           accented letters based on the same base letter.

1820           The collation order of elements within an equivalence class is determined by the weights  
1821           assigned on any subsequent levels after the primary weight.

### 1822 **3.151 Era**

1823           A locale-specific method for counting and displaying years.

1824           **Note:**     The *LC\_TIME* category is defined in detail in Section 7.3.5 (on page 142).

### 1825 **3.152 Event Management**

1826           The mechanism that enables applications to register for and be made aware of external events  
1827           such as data becoming available for reading.

### 1828 **3.153 Executable File**

1829           A regular file acceptable as a new process image file by the equivalent of the *exec* family of  
1830           functions, and thus usable as one form of a utility. The standard utilities described as compilers  
1831           can produce executable files, but other unspecified methods of producing executable files may  
1832           also be provided. The internal format of an executable file is unspecified, but a conforming  
1833           application cannot assume an executable file is a text file.

### 1834 **3.154 Execute**

1835           To perform command search and execution actions, as defined in the Shell and Utilities volume  
1836           of IEEE Std 1003.1-200x; see also Section 3.200 (on page 60).

1837           **Note:**     Command Search and Execution is defined in detail in the Shell and Utilities volume of  
1838           IEEE Std 1003.1-200x, Section 2.9.1.1, Command Search and Execution.

1839 **3.155 Execution Time**1840 See *CPU Time* in Section 3.117 (on page 49).1841 **3.156 Execution Time Monitoring**1842 A set of execution time monitoring primitives that allow online measuring of thread and process  
1843 execution times.1844 **3.157 Expand**

1845 In the shell command language, when not qualified, the act of applying word expansions.

1846 **Note:** Word Expansions are defined in detail in the Shell and Utilities volume of  
1847 IEEE Std 1003.1-200x, Section 2.6, Word Expansions.1848 **3.158 Extended Regular Expression (ERE)**1849 A regular expression (see also Section 3.316 (on page 76)) that is an alternative to the Basic  
1850 Regular Expression using a more extensive syntax, occasionally used by some utilities.1851 **Note:** Extended Regular Expressions are described in detail in Section 9.4 (on page 171).1852 **3.159 Extended Security Controls**1853 Implementation-defined security controls allowed by the file access permission and appropriate  
1854 privilege (see also Section 3.19 (on page 35)) mechanisms, through which an implementation can  
1855 support different security policies from those described in IEEE Std 1003.1-200x.1856 **Note:** See also Extended Security Controls defined in Section 4.3 (on page 95).

1857 File Access Permissions are defined in detail in Section 4.4 (on page 95).

1858 **3.160 Feature Test Macro**

1859 A macro used to determine whether a particular set of features is included from a header.

1860 **Note:** See also the System Interfaces volume of IEEE Std 1003.1-200x, Section 2.2, The Compilation  
1861 Environment.1862 **3.161 Field**1863 In the shell command language, a unit of text that is the result of parameter expansion,  
1864 arithmetic expansion, command substitution, or field splitting. During command processing, the  
1865 resulting fields are used as the command name and its arguments.1866 **Note:** Parameter Expansion is defined in detail in the Shell and Utilities volume of  
1867 IEEE Std 1003.1-200x, Section 2.6.2, Parameter Expansion.1868 Arithmetic Expansion is defined in detail in the Shell and Utilities volume of  
1869 IEEE Std 1003.1-200x, Section 2.6.4, Arithmetic Expansion.

1870 Command Substitution is defined in detail in the Shell and Utilities volume of  
1871 IEEE Std 1003.1-200x, Section 2.6.3, Command Substitution.

1872 Field Splitting is defined in detail in the Shell and Utilities volume of IEEE Std 1003.1-200x,  
1873 Section 2.6.5, Field Splitting.

1874 For further information on command processing, see the Shell and Utilities volume of  
1875 IEEE Std 1003.1-200x, Section 2.9.1, Simple Commands.

### 1876 **3.162 FIFO Special File (or FIFO)** |

1877 A type of file with the property that data written to such a file is read on a first-in-first-out basis. |

1878 **Note:** Other characteristics of FIFOs are described in the System Interfaces volume of  
1879 IEEE Std 1003.1-200x, *lseek()*, *open()*, *read()*, and *write()*.

### 1880 **3.163 File** |

1881 An object that can be written to, or read from, or both. A file has certain attributes, including |  
1882 access permissions and type. File types include regular file, character special file, block special  
1883 file, FIFO special file, symbolic link, socket, and directory. Other types of files may be supported  
1884 by the implementation.

### 1885 **3.164 File Description**

1886 See *Open File Description* in Section 3.253 (on page 67). |

### 1887 **3.165 File Descriptor** |

1888 A per-process unique, non-negative integer used to identify an open file for the purpose of file |  
1889 access. The value of a file descriptor is from zero to {OPEN\_MAX}. A process can have no more  
1890 than {OPEN\_MAX} file descriptors open simultaneously. File descriptors may also be used to  
1891 implement message catalog descriptors and directory streams; see also Section 3.253 (on page  
1892 67).

1893 **Note:** {OPEN\_MAX} is defined in detail in <limits.h>.

### 1894 **3.166 File Group Class** |

1895 The property of a file indicating access permissions for a process related to the group |  
1896 identification of a process. A process is in the file group class of a file if the process is not in the  
1897 file owner class and if the effective group ID or one of the supplementary group IDs of the  
1898 process matches the group ID associated with the file. Other members of the class may be  
1899 implementation-defined. |

### 1900 **3.167 File Mode** |

1901 An object containing the *file mode bits* and file type of a file. |

1902 **Note:** File mode bits and file types are defined in detail in <sys/stat.h>.

**1903 3.168 File Mode Bits**

1904 A file's file permission bits, set-user-ID-on-execution bit (S\_ISUID), and set-group-ID-on-  
1905 execution bit (S\_ISGID).

1906 **Note:** File Mode Bits are defined in detail in <sys/stat.h>.

**1907 3.169 Filename**

1908 A name consisting of 1 to {NAME\_MAX} bytes used to name a file. The characters composing  
1909 the name may be selected from the set of all character values excluding the slash character and  
1910 the null byte. The filenames dot and dot-dot have special meaning. A filename is sometimes  
1911 referred to as a *pathname component*.

1912 **Note:** Pathname Resolution is defined in detail in Section 4.11 (on page 98).

**1913 3.170 Filename Portability**

1914 Filenames should be constructed from the portable filename character set because the use of  
1915 other characters can be confusing or ambiguous in certain contexts. (For example, the use of a  
1916 colon (':') in a pathname could cause ambiguity if that pathname were included in a *PATH*  
1917 definition.)

**1918 3.171 File Offset**

1919 The byte position in the file where the next I/O operation begins. Each open file description  
1920 associated with a regular file, block special file, or directory has a file offset. A character special  
1921 file that does not refer to a terminal device may have a file offset. There is no file offset specified  
1922 for a pipe or FIFO.

**1923 3.172 File Other Class**

1924 The property of a file indicating access permissions for a process related to the user and group  
1925 identification of a process. A process is in the file other class of a file if the process is not in the  
1926 file owner class or file group class.

**1927 3.173 File Owner Class**

1928 The property of a file indicating access permissions for a process related to the user  
1929 identification of a process. A process is in the file owner class of a file if the effective user ID of  
1930 the process matches the user ID of the file.

**1931 3.174 File Permission Bits**

1932 Information about a file that is used, along with other information, to determine whether a  
1933 process has read, write, or execute/search permission to a file. The bits are divided into three  
1934 parts: owner, group, and other. Each part is used with the corresponding file class of processes.  
1935 These bits are contained in the file mode.

1936 **Note:** File modes are defined in detail in `<sys/stat.h>`.

1937 File Access Permissions are defined in detail in Section 4.4 (on page 95).

### 1938 **3.175 File Serial Number**

1939 A per-file system unique identifier for a file.

### 1940 **3.176 File System**

1941 A collection of files and certain of their attributes. It provides a name space for file serial  
1942 numbers referring to those files.

### 1943 **3.177 File Type**

1944 See *File* in Section 3.163 (on page 55).

### 1945 **3.178 Filter**

1946 A command whose operation consists of reading data from standard input or a list of input files  
1947 and writing data to standard output. Typically, its function is to perform some transformation  
1948 on the data stream.

### 1949 **3.179 First Open (of a File)**

1950 When a process opens a file that is not currently an open file within any process.

### 1951 **3.180 Flow Control**

1952 The mechanism employed by a communications provider that constrains a sending entity to  
1953 wait until the receiving entities can safely receive additional data without loss.

### 1954 **3.181 Foreground Job**

1955 See *Foreground Process Group* in Section 3.183.

### 1956 **3.182 Foreground Process**

1957 A process that is a member of a foreground process group. |

### 1958 **3.183 Foreground Process Group (or Foreground Job)** |

1959 A process group whose member processes have certain privileges, denied to processes in |  
1960 background process groups, when accessing their controlling terminal. Each session that has |

1961 established a connection with a controlling terminal has at most one process group of the session  
1962 as the foreground process group of that controlling terminal.

1963 **Note:** The General Terminal Interface is defined in detail in Chapter 11.

### 1964 **3.184 Foreground Process Group ID**

1965 The process group ID of the foreground process group. |

### 1966 **3.185 Form-Feed Character (<form-feed>)**

1967 A character that in the output stream indicates that printing should start on the next page of an  
1968 output device. It is the character designated by '`\f`' in the C language. If the <form-feed> is not  
1969 the first character of an output line, the result is unspecified. It is unspecified whether this  
1970 character is the exact sequence transmitted to an output device by the system to accomplish the  
1971 movement to the next page. |

### 1972 **3.186 Graphic Character**

1973 A member of the **graph** character class of the current locale. |

1974 **Note:** The **graph** character class is defined in detail in Section 7.3.1 (on page 122).

### 1975 **3.187 Group Database**

1976 A system database of implementation-defined format that contains at least the following  
1977 information for each group ID: |

- 1978 • Group name
- 1979 • Numerical group ID
- 1980 • List of users allowed in the group

1981 The list of users allowed in the group is used by the *newgrp* utility.

1982 **Note:** The *newgrp* utility is defined in detail in the Shell and Utilities volume of IEEE Std 1003.1-200x.

### 1983 **3.188 Group ID**

1984 A non-negative integer, which can be contained in an object of type **gid\_t**, that is used to identify  
1985 a group of system users. Each system user is a member of at least one group. When the identity  
1986 of a group is associated with a process, a group ID value is referred to as a real group ID, an  
1987 effective group ID, one of the supplementary group IDs, or a saved set-group-ID. |

### 1988 **3.189 Group Name**

1989 A string that is used to identify a group; see also Section 3.187. To be portable across conforming  
1990 systems, the value is composed of characters from the portable filename character set. The  
1991 hyphen should not be used as the first character of a portable group name.

**1992 3.190 Hard Limit**

1993 A system resource limitation that may be reset to a lesser or greater limit by a privileged process.  
1994 A non-privileged process is restricted to only lowering its hard limit. |

**1995 3.191 Hard Link**

1996 The relationship between two directory entries that represent the same file; see also Section 3.129 |  
1997 (on page 50). The result of an execution of the *ln* utility (without the *-s* option) or the *link()* |  
1998 function. This term is contrasted against symbolic link; see also Section 3.372 (on page 83). |

**1999 3.192 Home Directory**

2000 The directory specified by the *HOME* environment variable. |

**2001 3.193 Host Byte Order**

2002 The arrangement of bytes in any **int** type when using a specific machine architecture. |

2003 **Note:** Two common methods of byte ordering are big-endian and little-endian. Big-endian is a  
2004 format for storage of binary data in which the most significant byte is placed first, with the rest  
2005 in descending order. Little-endian is a format for storage or transmission of binary data in  
2006 which the least significant byte is placed first, with the rest in ascending order. See also Section |  
2007 4.8 (on page 97). |

**2008 3.194 Incomplete Line**

2009 A sequence of one or more non-*<newline>*s at the end of the file.

**2010 3.195 Inf**

2011 A value representing +infinity or a value representing -infinity that can be stored in a floating  
2012 type. Not all systems support the Inf values.

**2013 3.196 Instrumented Application**

2014 An application that contains at least one call to the trace point function *posix\_trace\_event()*. Each  
2015 process of an instrumented application has a mapping of trace event names to trace event type  
2016 identifiers. This mapping is used by the trace stream that is created for that process. |

**2017 3.197 Interactive Shell**

2018 A processing mode of the shell that is suitable for direct user interaction. |

2019 **3.198 Internationalization**

2020 The provision within a computer program of the capability of making itself adaptable to the  
2021 requirements of different native languages, local customs, and coded character sets.

2022 **3.199 Interprocess Communication**

2023 A functionality enhancement to add a high-performance, deterministic interprocess  
2024 communication facility for local communication.

2025 **3.200 Invoke**

2026 To perform command search and execution actions, except that searching for shell functions and  
2027 special built-in utilities is suppressed; see also Section 3.154 (on page 53).

2028 **Note:** Command Search and Execution is defined in detail in the Shell and Utilities volume of  
2029 IEEE Std 1003.1-200x, Section 2.9.1.1, Command Search and Execution.

2030 **3.201 Job**

2031 A set of processes, comprising a shell pipeline, and any processes descended from it, that are all  
2032 in the same process group.

2033 **Note:** See also the Shell and Utilities volume of IEEE Std 1003.1-200x, Section 2.9.2, Pipelines.

2034 **3.202 Job Control**

2035 A facility that allows users selectively to stop (suspend) the execution of processes and continue  
2036 (resume) their execution at a later point. The user typically employs this facility via the  
2037 interactive interface jointly supplied by the terminal I/O driver and a command interpreter.

2038 **3.203 Job Control Job ID**

2039 A handle that is used to refer to a job. The job control job ID can be any of the forms shown in the  
2040 following table:

2041

Table 3-1 Job Control Job ID Formats

2042

2043

2044

2045

2046

2047

2048

2049

Job Control Job ID	Meaning
%%	Current job.
%+	Current job.
%-	Previous job.
% <i>n</i>	Job number <i>n</i> .
% <i>string</i>	Job whose command begins with <i>string</i> .
%? <i>string</i>	Job whose command contains <i>string</i> .

2050 **3.204 Last Close (of a File)**

2051 When a process closes a file, resulting in the file not being an open file within any process.

2052 **3.205 Line**

2053 A sequence of zero or more non-&lt;newline&gt;s plus a terminating &lt;newline&gt;.

2054 **3.206 Linger**

2055 A period of time before terminating a connection, to allow outstanding data to be transferred.

2056 **3.207 Link**2057 See *Directory Entry* in Section 3.129 (on page 50).2058 **3.208 Link Count**

2059 The number of directory entries that refer to a particular file.

2060 **3.209 Local Customs**2061 The conventions of a geographical area or territory for such things as date, time, and currency  
2062 formats.2063 **3.210 Local Interprocess Communication (Local IPC)**

2064 The transfer of data between processes in the same system. |

2065 **3.211 Locale** |2066 The definition of the subset of a user's environment that depends on language and cultural  
2067 conventions. |

2068           **Note:**     Locales are defined in detail in Chapter 7 (on page 119).

### 2069   **3.212   Localization**

2070           The process of establishing information within a computer system specific to the operation of  
2071           particular native languages, local customs, and coded character sets.

### 2072   **3.213   Login**

2073           The unspecified activity by which a user gains access to the system. Each login is associated  
2074           with exactly one login name.

### 2075   **3.214   Login Name**

2076           A user name that is associated with a login.

### 2077   **3.215   Map**

2078           To create an association between a page-aligned range of the address space of a process and  
2079           some memory object, such that a reference to an address in that range of the address space  
2080           results in a reference to the associated memory object. The mapped memory object is not  
2081           necessarily memory-resident. |

### 2082   **3.216   Marked Message**

2083           A STREAMS message on which a certain flag is set. Marking a message gives the application |  
2084           protocol-specific information. An application can use *ioctl()* to determine whether a given |  
2085           message is marked. |

2086           **Note:**     The *ioctl()* function is defined in detail in the System Interfaces volume of  
2087           IEEE Std 1003.1-200x.

### 2088   **3.217   Matched**

2089           A state applying to a sequence of zero or more characters when the characters in the sequence |  
2090           correspond to a sequence of characters defined by a basic regular expression or extended regular |  
2091           expression pattern. |

2092           **Note:**     Regular Expressions are defined in detail in Chapter 9 (on page 165).

### 2093   **3.218   Memory Mapped Files**

2094           A facility to allow applications to access files as part of the address space. |

**2095 3.219 Memory Object** |

2096 One of: |

- 2097 • A file (see Section 3.163 (on page 55))
- 2098 • A shared memory object (see Section 3.340 (on page 79))
- 2099 • A typed memory object (see Section 3.418 (on page 90))

2100 When used in conjunction with *mmap()*, a memory object appears in the address space of the  
2101 calling process.

2102 **Note:** The *mmap()* function is defined in detail in the System Interfaces volume of  
2103 IEEE Std 1003.1-200x.

**2104 3.220 Memory-Resident**

2105 The process of managing the implementation in such a way as to provide an upper bound on  
2106 memory access times.

**2107 3.221 Message**

2108 In the context of programmatic message passing, information that can be transferred between  
2109 processes or threads by being added to and removed from a message queue. A message consists  
2110 of a fixed-size message buffer.

**2111 3.222 Message Catalog**

2112 In the context of providing natural language messages to the user, a file or storage area  
2113 containing program messages, command prompts, and responses to prompts for a particular  
2114 native language, territory, and codeset.

**2115 3.223 Message Catalog Descriptor**

2116 In the context of providing natural language messages to the user, a per-process unique value  
2117 used to identify an open message catalog. A message catalog descriptor may be implemented  
2118 using a file descriptor.

**2119 3.224 Message Queue**

2120 In the context of programmatic message passing, an object to which messages can be added and  
2121 removed. Messages may be removed in the order in which they were added or in priority order. |

**2122 3.225 Mode** |

2123 A collection of attributes that specifies a file's type and its access permissions. |

2124 **Note:** File Access Permissions are defined in detail in Section 4.4 (on page 95).

**2125 3.226 Monotonic Clock**

2126 A clock whose value cannot be set via `clock_settime()` and which cannot have negative clock  
2127 jumps.

**2128 3.227 Mount Point**

2129 Either the system root directory or a directory for which the `st_dev` field of structure `stat` differs  
2130 from that of its parent directory.

2131 **Note:** The `stat` structure is defined in detail in `<sys/stat.h>`.

**2132 3.228 Multi-Character Collating Element**

2133 A sequence of two or more characters that collate as an entity. For example, in some coded  
2134 character sets, an accented character is represented by a non-spacing accent, followed by the  
2135 letter. Other examples are the Spanish elements *ch* and *ll*.

**2136 3.229 Mutex**

2137 A synchronization object used to allow multiple threads to serialize their access to shared data.  
2138 The name derives from the capability it provides; namely, mutual-exclusion. The thread that has  
2139 locked a mutex becomes its owner and remains the owner until that same thread unlocks the  
2140 mutex.

**2141 3.230 Name**

2142 In the shell command language, a word consisting solely of underscores, digits, and alphabets  
2143 from the portable character set. The first character of a name is not a digit.

2144 **Note:** The portable character set is defined in detail in Section 6.1 (on page 111).

**2145 3.231 Named STREAM**

2146 A STREAMS-based file descriptor that is attached to a name in the file system name space. All  
2147 subsequent operations on the named STREAM act on the STREAM that was associated with the  
2148 file descriptor until the name is disassociated from the STREAM.

**2149 3.232 NaN (Not a Number)**

2150 A set of values that may be stored in a floating type but that are neither Inf nor valid floating-  
2151 point numbers. Not all systems support NaN values.

**2152 3.233 Native Language**

2153 A computer user's spoken or written language, such as American English, British English,  
2154 Danish, Dutch, French, German, Italian, Japanese, Norwegian, or Swedish.

**2155 3.234 Negative Response** |

2156 An input string that matches one of the responses acceptable to the *LC\_MESSAGES* category |  
2157 keyword **noexpr**, matching an extended regular expression in the current locale. |

2158 **Note:** The *LC\_MESSAGES* category is defined in detail in Section 7.3.6 (on page 148). |

**2159 3.235 Network** |

2160 A collection of interconnected hosts. |

2161 **Note:** The term network in IEEE Std 1003.1-200x is used to refer to the network of hosts. The term |  
2162 batch system is used to refer to the network of batch servers. |

**2163 3.236 Network Address** |

2164 A network-visible identifier used to designate specific endpoints in a network. Specific |  
2165 endpoints on host systems have addresses, and host systems may also have addresses. |

**2166 3.237 Network Byte Order** |

2167 The way of representing any **int** type such that, when transmitted over a network via a network |  
2168 endpoint, the **int** type is transmitted as an appropriate number of octets with the most |  
2169 significant octet first, followed by any other octets in descending order of significance. |

2170 **Note:** This order is more commonly known as big-endian ordering. See also Section 4.8 (on page 97). |

**2171 3.238 Newline Character (<newline>)** |

2172 A character that in the output stream indicates that printing should start at the beginning of the |  
2173 next line. It is the character designated by '`\n`' in the C language. It is unspecified whether this |  
2174 character is the exact sequence transmitted to an output device by the system to accomplish the |  
2175 movement to the next line. |

**2176 3.239 Nice Value** |

2177 A number used as advice to the system to alter process scheduling. Numerically smaller values |  
2178 give a process additional preference when scheduling a process to run. Numerically larger |  
2179 values reduce the preference and make a process less likely to run. Typically, a process with a |  
2180 smaller nice value runs to completion more quickly than an equivalent process with a higher |  
2181 nice value. The symbol {NZERO} specifies the default nice value of the system. |

**2182 3.240 Non-Blocking** |

2183 A property of an open file description that causes function calls involving it to return without |  
2184 delay when it is detected that the requested action associated with the function call cannot be |  
2185 completed without unknown delay. |

2186 **Note:** The exact semantics are dependent on the type of file associated with the open file description. |  
2187 For data reads from devices such as ttys and FIFOs, this property causes the read to return |

2188 immediately when no data was available. Similarly, for writes, it causes the call to return |  
2189 immediately when the thread would otherwise be delayed in the write operation; for example, |  
2190 because no space was available. For networking, it causes functions not to await protocol |  
2191 events (for example, acknowledgements) to occur. See also the System Interfaces volume of |  
2192 IEEE Std 1003.1-200x, Section 2.10.7, Socket I/O Mode. |

### 2193 **3.241 Non-Spacing Characters**

2194 A character, such as a character representing a diacritical mark in the ISO/IEC 6937:1994  
2195 standard coded character set, which is used in combination with other characters to form  
2196 composite graphic symbols.

### 2197 **3.242 NUL**

2198 A character with all bits set to zero.

### 2199 **3.243 Null Byte**

2200 A byte with all bits set to zero.

### 2201 **3.244 Null Pointer**

2202 The value that is obtained by converting the number 0 into a pointer; for example, (**void \***) 0. The  
2203 C language guarantees that this value does not match that of any legitimate pointer, so it is used  
2204 by many functions that return pointers to indicate an error.

### 2205 **3.245 Null String**

2206 See *Empty String* in Section 3.145 (on page 52).

### 2207 **3.246 Null Wide-Character Code**

2208 A wide-character code with all bits set to zero.

### 2209 **3.247 Number Sign**

2210 The character ' # ', also known as *hash sign*. |

### 2211 **3.248 Object File**

2212 A regular file containing the output of a compiler, formatted as input to a linkage editor for |  
2213 linking with other object files into an executable form. The methods of linking are unspecified |  
2214 and may involve the dynamic linking of objects at runtime. The internal format of an object file |  
2215 is unspecified, but a conforming application cannot assume an object file is a text file. |

**2216 3.249 Octet**

2217 Unit of data representation that consists of eight contiguous bits.

**2218 3.250 Offset Maximum**

2219 An attribute of an open file description representing the largest value that can be used as a file  
2220 offset.

**2221 3.251 Opaque Address**

2222 An address such that the entity making use of it requires no details about its contents or format.

**2223 3.252 Open File**

2224 A file that is currently associated with a file descriptor. |

**2225 3.253 Open File Description** |

2226 A record of how a process or group of processes is accessing a file. Each file descriptor refers to |  
2227 exactly one open file description, but an open file description can be referred to by more than |  
2228 one file descriptor. A file offset, file status, and file access modes are attributes of an open file |  
2229 description. |

**2230 3.254 Operand** |

2231 An argument to a command that is generally used as an object supplying information to a utility |  
2232 necessary to complete its processing. Operands generally follow the options in a command line. |

2233 **Note:** Utility Argument Syntax is defined in detail in Section 12.1 (on page 197).

**2234 3.255 Operator** |

2235 In the shell command language, either a control operator or a redirection operator. |

**2236 3.256 Option** |

2237 An argument to a command that is generally used to specify changes in the utility's default |  
2238 behavior. |

2239 **Note:** Utility Argument Syntax is defined in detail in Section 12.1 (on page 197).

**2240 3.257 Option-Argument** |

2241 A parameter that follows certain options. In some cases an option-argument is included within |  
2242 the same argument string as the option—in most cases it is the next argument. |

2243           **Note:**     Utility Argument Syntax is defined in detail in Section 12.1 (on page 197).

### 2244   **3.258 Orientation**

2245           A stream has one of three orientations: unoriented, byte-oriented, or wide-oriented.

2246           **Note:**     For further information, see the System Interfaces volume of IEEE Std 1003.1-200x, Section  
2247                        2.5.2, Stream Orientation and Encoding Rules.

### 2248   **3.259 Orphaned Process Group**

2249           A process group in which the parent of every member is either itself a member of the group or is  
2250           not a member of the group's session.

### 2251   **3.260 Page**

2252           The granularity of process memory mapping or locking.

2253           Physical memory and memory objects can be mapped into the address space of a process on  
2254           page boundaries and in integral multiples of pages. Process address space can be locked into  
2255           memory (made memory-resident) on page boundaries and in integral multiples of pages.

### 2256   **3.261 Page Size**

2257           The size, in bytes, of the system unit of memory allocation, protection, and mapping. On systems  
2258           that have segment rather than page-based memory architectures, the term page means a  
2259           segment.

### 2260   **3.262 Parameter**

2261           In the shell command language, an entity that stores values. There are three types of parameters:  
2262           variables (named parameters), positional parameters, and special parameters. Parameter  
2263           expansion is accomplished by introducing a parameter with the '\$' character.

2264           **Note:**     See also the Shell and Utilities volume of IEEE Std 1003.1-200x, Section 2.5, Parameters and  
2265                        Variables.

2266           In the C language, an object declared as part of a function declaration or definition that acquires  
2267           a value on entry to the function, or an identifier following the macro name in a function-like  
2268           macro definition.

### 2269   **3.263 Parent Directory**

2270           When discussing a given directory, the directory that both contains a directory entry for the  
2271           given directory and is represented by the pathname dot-dot in the given directory.

2272           When discussing other types of files, a directory containing a directory entry for the file under  
2273           discussion.

2274 This concept does not apply to dot and dot-dot.

### 2275 **3.264 Parent Process**

2276 The process which created (or inherited) the process under discussion.

### 2277 **3.265 Parent Process ID**

2278 An attribute of a new process identifying the parent of the process. The parent process ID of a  
2279 process is the process ID of its creator, for the lifetime of the creator. After the creator's lifetime  
2280 has ended, the parent process ID is the process ID of an implementation-defined system process. |

### 2281 **3.266 Pathname**

2282 A character string that is used to identify a file. In the context of IEEE Std 1003.1-200x, a |  
2283 pathname consists of, at most, {PATH\_MAX} bytes, including the terminating null byte. It has an |  
2284 optional beginning slash, followed by zero or more filenames separated by slashes. A pathname  
2285 may optionally contain one or more trailing slashes. Multiple successive slashes are considered  
2286 to be the same as one slash.

2287 **Note:** Pathname Resolution is defined in detail in Section 4.11 (on page 98).

### 2288 **3.267 Pathname Component**

2289 See *Filename* in Section 3.169 (on page 56).

### 2290 **3.268 Path Prefix**

2291 A pathname, with an optional ending slash, that refers to a directory. |

### 2292 **3.269 Pattern**

2293 A sequence of characters used either with regular expression notation or for pathname |  
2294 expansion, as a means of selecting various character strings or pathnames, respectively. |

2295 **Note:** Regular Expressions are defined in detail in Chapter 9 (on page 165).

2296 See also the Shell and Utilities volume of IEEE Std 1003.1-200x, Section 2.6.6, Pathname  
2297 Expansion.

2298 The syntaxes of the two types of patterns are similar, but not identical; IEEE Std 1003.1-200x  
2299 always indicates the type of pattern being referred to in the immediate context of the use of the  
2300 term.

### 2301 **3.270 Period**

2302 The character ' . '. The term period is contrasted with dot (see also Section 3.135 (on page 51)),  
2303 which is used to describe a specific directory entry. |

2304 **3.271 Permissions** |

2305 Attributes of an object that determine the privilege necessary to access or manipulate the object. |

2306 **Note:** File Access Permissions are defined in detail in Section 4.4 (on page 95). |

2307 **3.272 Persistence** |

2308 A mode for semaphores, shared memory, and message queues requiring that the object and its |  
2309 state (including data, if any) are preserved after the object is no longer referenced by any process. |

2310 Persistence of an object does not imply that the state of the object is maintained across a system |  
2311 crash or a system reboot. |

2312 **3.273 Pipe** |

2313 An object accessed by one of the pair of file descriptors created by the *pipe()* function. Once |  
2314 created, the file descriptors can be used to manipulate it, and it behaves identically to a FIFO |  
2315 special file when accessed in this way. It has no name in the file hierarchy. |

2316 **Note:** The *pipe()* function is defined in detail in the System Interfaces volume of |  
2317 IEEE Std 1003.1-200x. |

2318 **3.274 Polling** |

2319 A scheduling scheme whereby the local process periodically checks until the prespecified events |  
2320 (for example, read, write) have occurred. |

2321 **3.275 Portable Character Set** |

2322 The collection of characters that are required to be present in all locales supported by |  
2323 conforming systems. |

2324 **Note:** The portable character set is defined in detail in Section 6.1 (on page 111). |

2325 This term is contrasted against the smaller *portable filename character set*; see also Section 3.276. |

2326 **3.276 Portable Filename Character Set** |

2327 The set of characters from which portable filenames are constructed. |

2328       A B C D E F G H I J K L M N O P Q R S T U V W X Y Z  
2329       a b c d e f g h i j k l m n o p q r s t u v w x y z  
2330       0 1 2 3 4 5 6 7 8 9 . \_ -

2331 The last three characters are the period, underscore, and hyphen characters, respectively. |

2332 **3.277 Positional Parameter** |

2333 In the shell command language, a parameter denoted by a single digit or one or more digits in |  
2334 curly braces. |

2335           **Note:**     For further information, see the Shell and Utilities volume of IEEE Std 1003.1-200x, Section  
2336                    2.5.1, Positional Parameters.

### 2337 **3.278 Preallocation** |

2338            The reservation of resources in a system for a particular use. |

2339            Preallocation does not imply that the resources are immediately allocated to that use, but merely  
2340            indicates that they are guaranteed to be available in bounded time when needed.

### 2341 **3.279 Preempted Process (or Thread)**

2342            A running thread whose execution is suspended due to another thread becoming runnable at a  
2343            higher priority. |

### 2344 **3.280 Previous Job** |

2345            In the context of job control, the job that will be used as the default for the *fg* or *bg* utilities if the  
2346            current job exits. There is at most one previous job; see also Section 3.203 (on page 60). |

### 2347 **3.281 Printable Character** |

2348            One of the characters included in the **print** character classification of the *LC\_CTYPE* category in  
2349            the current locale. |

2350            **Note:**     The *LC\_CTYPE* category is defined in detail in Section 7.3.1 (on page 122).

### 2351 **3.282 Printable File** |

2352            A text file consisting only of the characters included in the **print** and **space** character  
2353            classifications of the *LC\_CTYPE* category and the <backspace>, all in the current locale. |

2354            **Note:**     The *LC\_CTYPE* category is defined in detail in Section 7.3.1 (on page 122).

### 2355 **3.283 Priority**

2356            A non-negative integer associated with processes or threads whose value is constrained to a  
2357            range defined by the applicable scheduling policy. Numerically higher values represent higher  
2358            priorities.

### 2359 **3.284 Priority Band**

2360            The queuing order applied to normal priority STREAMS messages. High priority STREAMS  
2361            messages are not grouped by priority bands. The only differentiation made by the STREAMS  
2362            mechanism is between zero and non-zero bands, but specific protocol modules may differentiate  
2363            between priority bands.

2364 **3.285 Priority Inversion**

2365 A condition in which a thread that is not voluntarily suspended (waiting for an event or time  
2366 delay) is not running while a lower priority thread is running. Such blocking of the higher  
2367 priority thread is often caused by contention for a shared resource.

2368 **3.286 Priority Scheduling**

2369 A performance and determinism improvement facility to allow applications to determine the  
2370 order in which threads that are ready to run are granted access to processor resources.

2371 **3.287 Priority-Based Scheduling**

2372 Scheduling in which the selection of a running thread is determined by the priorities of the  
2373 runnable processes or threads.

2374 **3.288 Privilege**

2375 See *Appropriate Privileges* in Section 3.19 (on page 35). |

2376 **3.289 Process** |

2377 An address space with one or more threads executing within that address space, and the |  
2378 required system resources for those threads.

2379 **Note:** Many of the system resources defined by IEEE Std 1003.1-200x are shared among all of the  
2380 threads within a process. These include the process ID, the parent process ID, process group ID,  
2381 session membership, real, effective, and saved-set user ID, real, effective, and saved-set group  
2382 ID, supplementary group IDs, current working directory, root directory, file mode creation  
2383 mask, and file descriptors.

2384 **3.290 Process Group**

2385 A collection of processes that permits the signaling of related processes. Each process in the  
2386 system is a member of a process group that is identified by a process group ID. A newly created  
2387 process joins the process group of its creator. |

2388 **3.291 Process Group ID** |

2389 The unique positive integer identifier representing a process group during its lifetime. |

2390 **Note:** See also Process Group ID Reuse defined in Section 4.12 (on page 99).

2391 **3.292 Process Group Leader**

2392 A process whose process ID is the same as its process group ID. |

### 2393 3.293 Process Group Lifetime

2394 A period of time that begins when a process group is created and ends when the last remaining  
 2395 process in the group leaves the group, due either to the end of the last process' lifetime or to the  
 2396 last remaining process calling the *setsid()* or *setpgid()* functions.

2397 **Note:** The *setsid()* and *setpgid()* functions are defined in detail in the System Interfaces volume of  
 2398 IEEE Std 1003.1-200x.

### 2399 3.294 Process ID

2400 The unique positive integer identifier representing a process during its lifetime.

2401 **Note:** See also Process ID Reuse defined in Section 4.12 (on page 99).

### 2402 3.295 Process Lifetime

2403 The period of time that begins when a process is created and ends when its process ID is  
 2404 returned to the system. After a process is created with a *fork()* function, it is considered active.  
 2405 At least one thread of control and address space exist until it terminates. It then enters an  
 2406 inactive state where certain resources may be returned to the system, although some resources,  
 2407 such as the process ID, are still in use. When another process executes a *wait()*, *waitid()*, or  
 2408 *waitpid()* function for an inactive process, the remaining resources are returned to the system.  
 2409 The last resource to be returned to the system is the process ID. At this time, the lifetime of the  
 2410 process ends.

2411 **Note:** The *fork()*, *wait()*, *waitid()*, and *waitpid()* functions are defined in detail in the System  
 2412 Interfaces volume of IEEE Std 1003.1-200x.

### 2413 3.296 Process Memory Locking

2414 A performance improvement facility to bind application programs into the high-performance  
 2415 random access memory of a computer system. This avoids potential latencies introduced by the  
 2416 operating system in storing parts of a program that were not recently referenced on secondary  
 2417 memory devices.

### 2418 3.297 Process Termination

2419 There are two kinds of process termination:

- 2420 1. Normal termination occurs by a return from *main()* or when requested with the *exit()* or  
 2421 *\_exit()* functions.
- 2422 2. Abnormal termination occurs when requested by the *abort()* function or when some  
 2423 signals are received.

2424 **Note:** The *\_exit()*, *abort()*, and *exit()* functions are defined in detail in the System Interfaces volume  
 2425 of IEEE Std 1003.1-200x.

**2426 3.298 Process-To-Process Communication**

2427 The transfer of data between processes.

**2428 3.299 Process Virtual Time**

2429 The measurement of time in units elapsed by the system clock while a process is executing.

**2430 3.300 Program**

2431 A prepared sequence of instructions to the system to accomplish a defined task. The term  
2432 program in IEEE Std 1003.1-200x encompasses applications written in the Shell Command  
2433 Language, complex utility input languages (for example, *awk*, *lex*, *sed*, and so on), and high-level  
2434 languages.

**2435 3.301 Protocol**

2436 A set of semantic and syntactic rules for exchanging information.

**2437 3.302 Pseudo-Terminal**

2438 A facility that provides an interface that is identical to the terminal subsystem. A pseudo-  
2439 terminal is composed of two devices: the *master device* and a *slave device*. The slave device  
2440 provides processes with an interface that is identical to the terminal interface, although there  
2441 need not be hardware behind that interface. Anything written on the master device is presented  
2442 to the slave as an input and anything written on the slave device is presented as an input on the  
2443 master side.

**2444 3.303 Radix Character**

2445 The character that separates the integer part of a number from the fractional part.

**2446 3.304 Read-Only File System**

2447 A file system that has implementation-defined characteristics restricting modifications.

2448 **Note:** File Times Update is described in detail in Section 4.7 (on page 96).

**2449 3.305 Read-Write Lock**

2450 Multiple readers, single writer (read-write) locks allow many threads to have simultaneous  
2451 read-only access to data while allowing only one thread to have write access at any given time.  
2452 They are typically used to protect data that is read-only more frequently than it is changed.

2453 Read-write locks can be used to synchronize threads in the current process and other processes if  
2454 they are allocated in memory that is writable and shared among the cooperating processes and  
2455 have been initialized for this behavior.

2456 **3.306 Real Group ID**

2457 The attribute of a process that, at the time of process creation, identifies the group of the user  
2458 who created the process; see also Section 3.188 (on page 58).

2459 **3.307 Real Time**

2460 Time measured as total units elapsed by the system clock without regard to which thread is  
2461 executing.

2462 **3.308 Realtime Signal Extension**

2463 A determinism improvement facility to enable asynchronous signal notifications to an  
2464 application to be queued without impacting compatibility with the existing signal functions.

2465 **3.309 Real User ID**

2466 The attribute of a process that, at the time of process creation, identifies the user who created the  
2467 process; see also Section 3.425 (on page 91).

2468 **3.310 Record**

2469 A collection of related data units or words which is treated as a unit. |

2470 **3.311 Redirection** |

2471 In the shell command language, a method of associating files with the input or output of |  
2472 commands.

2473 **Note:** For further information, see the Shell and Utilities volume of IEEE Std 1003.1-200x, Section 2.7,  
2474 Redirection.

2475 **3.312 Redirection Operator**

2476 In the shell command language, a token that performs a redirection function. It is one of the  
2477 following symbols:

2478 < > >| << >> <& >& <<- <>

2479 **3.313 Reentrant Function**

2480 A function whose effect, when called by two or more threads, is guaranteed to be as if the  
2481 threads each executed the function one after another in an undefined order, even if the actual  
2482 execution is interleaved.

**2483 3.314 Referenced Shared Memory Object**

2484 A shared memory object that is open or has one or more mappings defined on it.

**2485 3.315 Refresh**

2486 To ensure that the information on the user's terminal screen is up-to-date. |

**2487 3.316 Regular Expression** |

2488 A pattern that selects specific strings from a set of character strings. |

2489 **Note:** Regular Expressions are described in detail in Chapter 9 (on page 165).

**2490 3.317 Region** |

2491 In the context of the address space of a process, a sequence of addresses. |

2492 In the context of a file, a sequence of offsets.

**2493 3.318 Regular File**

2494 A file that is a randomly accessible sequence of bytes, with no further structure imposed by the  
2495 system. |

**2496 3.319 Relative Pathname** |

2497 A pathname not beginning with a slash. |

2498 **Note:** Pathname Resolution is defined in detail in Section 4.11 (on page 98).

**2499 3.320 Relocatable File**

2500 A file holding code or data suitable for linking with other object files to create an executable or a  
2501 shared object file.

**2502 3.321 Relocation**

2503 The process of connecting symbolic references with symbolic definitions. For example, when a  
2504 program calls a function, the associated call instruction transfers control to the proper  
2505 destination address at execution.

**2506 3.322 Requested Batch Service**

2507 A service that is either rejected or performed prior to a response from the service to the  
2508 requester.

**2509 3.323 (Time) Resolution**

2510 The minimum time interval that a clock can measure or whose passage a timer can detect.

**2511 3.324 Root Directory**

2512 A directory, associated with a process, that is used in pathname resolution for pathnames that  
2513 begin with a slash.

**2514 3.325 Runnable Process (or Thread)**

2515 A thread that is capable of being a running thread, but for which no processor is available.

**2516 3.326 Running Process (or Thread)**

2517 A thread currently executing on a processor. On multi-processor systems there may be more  
2518 than one such thread in a system at a time.

**2519 3.327 Saved Resource Limits**

2520 An attribute of a process that provides some flexibility in the handling of unrepresentable  
2521 resource limits, as described in the *exec* family of functions and *setrlimit()*.

2522 **Note:** The *exec* and *setrlimit()* functions are defined in detail in the System Interfaces volume of  
2523 IEEE Std 1003.1-200x.

**2524 3.328 Saved Set-Group-ID**

2525 An attribute of a process that allows some flexibility in the assignment of the effective group ID  
2526 attribute, as described in the *exec* family of functions and *setgid()*.

2527 **Note:** The *exec* and *setgid()* functions are defined in detail in the System Interfaces volume of  
2528 IEEE Std 1003.1-200x.

**2529 3.329 Saved Set-User-ID**

2530 An attribute of a process that allows some flexibility in the assignment of the effective user ID  
2531 attribute, as described in the *exec* family of functions and *setuid()*.

2532 **Note:** The *exec* and *setuid()* functions are defined in detail in the System Interfaces volume of  
2533 IEEE Std 1003.1-200x.

**2534 3.330 Scheduling**

2535 The application of a policy to select a runnable process or thread to become a running process or  
2536 thread, or to alter one or more of the thread lists.

**2537 3.331 Scheduling Allocation Domain**

2538 The set of processors on which an individual thread can be scheduled at any given time. |

**2539 3.332 Scheduling Contention Scope** |

2540 A property of a thread that defines the set of threads against which that thread competes for |  
2541 resources.

2542 For example, in a scheduling decision, threads sharing scheduling contention scope compete for  
2543 processor resources. In IEEE Std 1003.1-200x, a thread has scheduling contention scope of either  
2544 PTHREAD\_SCOPE\_SYSTEM or PTHREAD\_SCOPE\_PROCESS. |

**2545 3.333 Scheduling Policy** |

2546 A set of rules that is used to determine the order of execution of processes or threads to achieve |  
2547 some goal.

2548 **Note:** Scheduling Policy is defined in detail in Section 4.13 (on page 99).

**2549 3.334 Screen**

2550 A rectangular region of columns and lines on a terminal display. A screen may be a portion of a |  
2551 physical display device or may occupy the entire physical area of the display device.

**2552 3.335 Scroll** |

2553 To move the representation of data vertically or horizontally relative to the terminal screen. |  
2554 There are two types of scrolling:

- 2555 1. The cursor moves with the data.
- 2556 2. The cursor remains stationary while the data moves.

**2557 3.336 Semaphore** |

2558 A minimum synchronization primitive to serve as a basis for more complex synchronization |  
2559 mechanisms to be defined by the application program.

2560 **Note:** Semaphores are defined in detail in Section 4.15 (on page 100).

**2561 3.337 Session** |

2562 A collection of process groups established for job control purposes. Each process group is a |  
2563 member of a session. A process is considered to be a member of the session of which its process  
2564 group is a member. A newly created process joins the session of its creator. A process can alter  
2565 its session membership; see *setsid()*. There can be multiple process groups in the same session.

2566 **Note:** The *setsid()* function is defined in detail in the System Interfaces volume of  
2567 IEEE Std 1003.1-200x.

**2568 3.338 Session Leader** |

2569 A process that has created a session. |

2570 **Note:** For further information, see the *setsid()* function defined in the System Interfaces volume of  
2571 IEEE Std 1003.1-200x.

**2572 3.339 Session Lifetime**

2573 The period between when a session is created and the end of the lifetime of all the process  
2574 groups that remain as members of the session.

**2575 3.340 Shared Memory Object**

2576 An object that represents memory that can be mapped concurrently into the address space of  
2577 more than one process.

**2578 3.341 Shell**

2579 A program that interprets sequences of text input as commands. It may operate on an input  
2580 stream or it may interactively prompt and read commands from a terminal. |

**2581 3.342 Shell, the** |

2582 The Shell Command Language Interpreter; a specific instance of a shell. |

2583 **Note:** For further information, see the *sh* utility defined in the Shell and Utilities volume of  
2584 IEEE Std 1003.1-200x.

**2585 3.343 Shell Script** |

2586 A file containing shell commands. If the file is made executable, it can be executed by specifying  
2587 its name as a simple command. Execution of a shell script causes a shell to execute the  
2588 commands within the script. Alternatively, a shell can be requested to execute the commands in  
2589 a shell script by specifying the name of the shell script as the operand to the *sh* utility. |

2590 **Note:** Simple Commands are defined in detail in the Shell and Utilities volume of  
2591 IEEE Std 1003.1-200x, Section 2.9.1, Simple Commands.

2592 The *sh* utility is defined in detail in the Shell and Utilities volume of IEEE Std 1003.1-200x.

**2593 3.344 Signal**

2594 A mechanism by which a process or thread may be notified of, or affected by, an event occurring  
2595 in the system. Examples of such events include hardware exceptions and specific actions by  
2596 processes. The term signal is also used to refer to the event itself.

**2597 3.345 Signal Stack**

2598 Memory established for a thread, in which signal handlers catching signals sent to that thread  
2599 are executed.

**2600 3.346 Single-Quote**

2601 The character ' ' , also known as *apostrophe*.

**2602 3.347 Slash**

2603 The character ' / ' , also known as *solidus*.

**2604 3.348 Socket**

2605 A file of a particular type that is used as a communications endpoint for process-to-process  
2606 communication as described in the System Interfaces volume of IEEE Std 1003.1-200x.

**2607 3.349 Socket Address**

2608 An address associated with a socket or remote endpoint, including an address family identifier  
2609 and addressing information specific to that address family. The address may include multiple  
2610 parts, such as a network address associated with a host system and an identifier for a specific  
2611 endpoint.

**2612 3.350 Soft Limit**

2613 A resource limitation established for each process that the process may set to any value less than  
2614 or equal to the hard limit. |

**2615 3.351 Source Code** |

2616 When dealing with the Shell Command Language, input to the command language interpreter. |  
2617 The term shell script is synonymous with this meaning.

2618 When dealing with an ISO/IEC-conforming programming language, source code is input to a  
2619 compiler conforming to that ISO/IEC standard.

2620 Source code also refers to the input statements prepared for the following standard utilities:  
2621 *awk, bc, ed, lex, localedef, make, sed, and yacc*.

2622 Source code can also refer to a collection of sources meeting any or all of these meanings.

2623 **Note:** The *awk, bc, ed, lex, localedef, make, sed, and yacc* utilities are defined in detail in the Shell and  
2624 Utilities volume of IEEE Std 1003.1-200x.

2625 **3.352 Space Character (<space>)** |

2626 The character defined in the portable character set as <space>. The <space> is a member of the  
 2627 **space** character class of the current locale, but represents the single character, and not all of the  
 2628 possible members of the class; see also Section 3.431 (on page 92). |

2629 **3.353 Spawn** |

2630 A process creation primitive useful for systems that have difficulty with *fork()* and as an efficient  
 2631 replacement for *fork()/exec*. |

2632 **3.354 Special Built-In** |

2633 See *Built-In Utility* in Section 3.83 (on page 44). |

2634 **3.355 Special Parameter** |

2635 In the shell command language, a parameter named by a single character from the following list: |

2636 \* @ # ? ! - \$ 0

2637 **Note:** For further information, see the Shell and Utilities volume of IEEE Std 1003.1-200x, Section  
 2638 2.5.2, Special Parameters. |

2639 **3.356 Spin Lock** |

2640 A synchronization object used to allow multiple threads to serialize their access to shared data. |

2641 **3.357 Sporadic Server** |

2642 A scheduling policy for threads and processes that reserves a certain amount of execution  
 2643 capacity for processing aperiodic events at a given priority level. |

2644 **3.358 Standard Error** |

2645 An output stream usually intended to be used for diagnostic messages. |

2646 **3.359 Standard Input** |

2647 An input stream usually intended to be used for primary data input. |

2648 **3.360 Standard Output** |

2649 An output stream usually intended to be used for primary data output. |

### 2650 **3.361 Standard Utilities**

2651 The utilities described in the Shell and Utilities volume of IEEE Std 1003.1-200x. |

### 2652 **3.362 Stream**

2653 Appearing in lowercase, a stream is a file access object that allows access to an ordered sequence  
 2654 of characters, as described by the ISO C standard. Such objects can be created by the *fdopen()*,  
 2655 *fopen()*, or *popen()* functions, and are associated with a file descriptor. A stream provides the  
 2656 additional services of user-selectable buffering and formatted input and output; see also Section  
 2657 3.363.

2658 **Note:** For further information, see the System Interfaces volume of IEEE Std 1003.1-200x, Section 2.5,  
 2659 Standard I/O Streams.

2660 The *fdopen()*, *fopen()*, or *popen()* functions are defined in detail in the System Interfaces volume  
 2661 of IEEE Std 1003.1-200x.

### 2662 **3.363 STREAM**

2663 Appearing in uppercase, STREAM refers to a full duplex connection between a process and an  
 2664 open device or pseudo-device. It optionally includes one or more intermediate processing  
 2665 modules that are interposed between the process end of the STREAM and the device driver (or  
 2666 pseudo-device driver) end of the STREAM; see also Section 3.362.

2667 **Note:** For further information, see the System Interfaces volume of IEEE Std 1003.1-200x, Section 2.6,  
 2668 STREAMS.

### 2669 **3.364 STREAM End**

2670 The STREAM end is the driver end of the STREAM and is also known as the downstream end of  
 2671 the STREAM.

### 2672 **3.365 STREAM Head**

2673 The STREAM head is the beginning of the STREAM and is at the boundary between the system  
 2674 and the application process. This is also known as the upstream end of the STREAM.

### 2675 **3.366 STREAMS Multiplexor**

2676 A driver with multiple STREAMS connected to it. Multiplexing with STREAMS connected above  
 2677 is referred to as N-to-1, or *upper multiplexing*. Multiplexing with STREAMS connected below is  
 2678 referred to as 1-to-N or *lower multiplexing*.

### 2679 **3.367 String**

2680 A contiguous sequence of bytes terminated by and including the first null byte. |

**2681 3.368 Subshell** |

2682 A shell execution environment, distinguished from the main or current shell execution |  
2683 environment.

2684 **Note:** For further information, see the Shell and Utilities volume of IEEE Std 1003.1-200x, Section 2.12,  
2685 Shell Execution Environment.

**2686 3.369 Successfully Transferred** |

2687 For a write operation to a regular file, when the system ensures that all data written is readable |  
2688 on any subsequent open of the file (even one that follows a system or power failure) in the  
2689 absence of a failure of the physical storage medium.

2690 For a read operation, when an image of the data on the physical storage medium is available to  
2691 the requesting process.

**2692 3.370 Supplementary Group ID**

2693 An attribute of a process used in determining file access permissions. A process has up to  
2694 {NGROUPS\_MAX} supplementary group IDs in addition to the effective group ID. The  
2695 supplementary group IDs of a process are set to the supplementary group IDs of the parent  
2696 process when the process is created.

**2697 3.371 Suspended Job**

2698 A job that has received a SIGSTOP, SIGTSTP, SIGTTIN, or SIGTTOU signal that caused the  
2699 process group to stop. A suspended job is a background job, but a background job is not  
2700 necessarily a suspended job. |

**2701 3.372 Symbolic Link** |

2702 A type of file with the property that when the file is encountered during pathname resolution, a |  
2703 string stored by the file is used to modify the pathname resolution. The stored string has a length  
2704 of {SYMLINK\_MAX} bytes or fewer.

2705 **Note:** Pathname Resolution is defined in detail in Section 4.11 (on page 98).

**2706 3.373 Synchronized Input and Output**

2707 A determinism and robustness improvement mechanism to enhance the data input and output  
2708 mechanisms, so that an application can ensure that the data being manipulated is physically  
2709 present on secondary mass storage devices.

**2710 3.374 Synchronized I/O Completion**

2711 The state of an I/O operation that has either been successfully transferred or diagnosed as  
2712 unsuccessful. |

**2713 3.375 Synchronized I/O Data Integrity Completion** |

2714 For read, when the operation has been completed or diagnosed if unsuccessful. The read is |  
2715 complete only when an image of the data has been successfully transferred to the requesting |  
2716 process. If there were any pending write requests affecting the data to be read at the time that |  
2717 the synchronized read operation was requested, these write requests are successfully transferred |  
2718 prior to reading the data.

2719 For write, when the operation has been completed or diagnosed if unsuccessful. The write is |  
2720 complete only when the data specified in the write request is successfully transferred and all file |  
2721 system information required to retrieve the data is successfully transferred.

2722 File attributes that are not necessary for data retrieval (access time, modification time, status |  
2723 change time) need not be successfully transferred prior to returning to the calling process.

**2724 3.376 Synchronized I/O File Integrity Completion**

2725 Identical to a synchronized I/O data integrity completion with the addition that all file attributes |  
2726 relative to the I/O operation (including access time, modification time, status change time) are |  
2727 successfully transferred prior to returning to the calling process.

**2728 3.377 Synchronized I/O Operation**

2729 An I/O operation performed on a file that provides the application assurance of the integrity of |  
2730 its data and files. |

**2731 3.378 Synchronous I/O Operation** |

2732 An I/O operation that causes the thread requesting the I/O to be blocked from further use of the |  
2733 processor until that I/O operation completes. |

2734 **Note:** A synchronous I/O operation does not imply synchronized I/O data integrity completion or |  
2735 synchronized I/O file integrity completion.

**2736 3.379 Synchronously-Generated Signal** |

2737 A signal that is attributable to a specific thread. |

2738 For example, a thread executing an illegal instruction or touching invalid memory causes a |  
2739 synchronously-generated signal. Being synchronous is a property of how the signal was |  
2740 generated and not a property of the signal number.

**2741 3.380 System**

2742 An implementation of IEEE Std 1003.1-200x.

**2743 3.381 System Crash**

2744 An interval initiated by an unspecified circumstance that causes all processes (possibly other  
2745 than special system processes) to be terminated in an undefined manner, after which any  
2746 changes to the state and contents of files created or written to by an application prior to the  
2747 interval are undefined, except as required elsewhere in IEEE Std 1003.1-200x.

**2748 3.382 System Console**

2749 An implementation-defined device that receives messages sent by the *syslog()* function, and the  
2750 *fmtmsg()* function when the MM\_CONSOLE flat is set.

2751 **Note:** The *syslog()* and *fmtmsg()* functions are defined in detail in the System Interfaces volume of  
2752 IEEE Std 1003.1-200x.

**2753 3.383 System Databases**

2754 An implementation provides two system databases.

2755 The *group database* contains the following information for each group:

- 2756 1. Group name
- 2757 2. Numerical group ID
- 2758 3. List of all users allowed in the group

2759 The *user database* contains the following information for each user:

- 2760 1. User name
- 2761 2. Numerical user ID
- 2762 3. Numerical group ID
- 2763 4. Initial working directory
- 2764 5. Initial user program

2765 If the initial user program field is null, the system default is used. If the initial working directory  
2766 field is null, the interpretation of that field is implementation-defined. These databases may  
2767 contain other fields that are unspecified by IEEE Std 1003.1-200x.

**2768 3.384 System Documentation**

2769 All documentation provided with an implementation except for the conformance document.  
2770 Electronically distributed documents for an implementation are considered part of the system  
2771 documentation.

**2772 3.385 System Process**

2773 An implementation-defined object, other than a process executing an application, that has a  
2774 process ID.

**2775 3.386 System Reboot**

2776 An implementation-defined sequence of events that may result in the loss of transitory data; that  
2777 is, data that is not saved in permanent storage. For example, message queues, shared memory,  
2778 semaphores, and processes.

**2779 3.387 System Trace Event**

2780 A trace event that is generated by the implementation, in response either to a system-initiated  
2781 action or to an application-requested action, except for a call to *posix\_trace\_event()*. When  
2782 supported by the implementation, a system-initiated action generates a process-independent  
2783 system trace event and an application-requested action generates a process-dependent system  
2784 trace event. For a system trace event not defined by IEEE Std 1003.1-200x, the associated trace  
2785 event type identifier is derived from the implementation-defined name for this trace event, and  
2786 the associated data is of implementation-defined content and length.

**2787 3.388 System-Wide**

2788 Pertaining to events occurring in all processes existing in an implementation at a given point in  
2789 time.

**2790 3.389 Tab Character (<tab>)**

2791 A character that in the output stream indicates that printing or displaying should start at the  
2792 next horizontal tabulation position on the current line. It is the character designated by '`\t`' in  
2793 the C language. If the current position is at or past the last defined horizontal tabulation  
2794 position, the behavior is unspecified. It is unspecified whether this character is the exact  
2795 sequence transmitted to an output device by the system to accomplish the tabulation.

**2796 3.390 Terminal (or Terminal Device)**

2797 A character special file that obeys the specifications of the general terminal interface.

2798 **Note:** The General Terminal Interface is defined in detail in Chapter 11 (on page 183).

**2799 3.391 Text Column**

2800 A roughly rectangular block of characters capable of being laid out side-by-side next to other  
2801 text columns on an output page or terminal screen. The widths of text columns are measured in  
2802 column positions.

**2803 3.392 Text File**

2804 A file that contains characters organized into one or more lines. The lines do not contain NUL  
2805 characters and none can exceed `{LINE_MAX}` bytes in length, including the `<newline>`.  
2806 Although IEEE Std 1003.1-200x does not distinguish between text files and binary files (see the  
2807 ISO C standard), many utilities only produce predictable or meaningful output when operating  
2808 on text files. The standard utilities that have such restrictions always specify *text files* in their

2809 STDIN or INPUT FILES sections. |

### 2810 **3.393 Thread** |

2811 A single flow of control within a process. Each thread has its own thread ID, scheduling priority |  
2812 and policy, *errno* value, thread-specific key/value bindings, and the required system resources to |  
2813 support a flow of control. Anything whose address may be determined by a thread, including |  
2814 but not limited to static variables, storage obtained via *malloc()*, directly addressable storage |  
2815 obtained through implementation-defined functions, and automatic variables, are accessible to |  
2816 all threads in the same process.

2817 **Note:** The *malloc()* function is defined in detail in the System Interfaces volume of |  
2818 IEEE Std 1003.1-200x.

### 2819 **3.394 Thread ID**

2820 Each thread in a process is uniquely identified during its lifetime by a value of type **pthread\_t** |  
2821 called a thread ID. |

### 2822 **3.395 Thread List** |

2823 An ordered set of runnable threads that all have the same ordinal value for their priority. |

2824 The ordering of threads on the list is determined by a scheduling policy or policies. The set of |  
2825 thread lists includes all runnable threads in the system.

### 2826 **3.396 Thread-Safe**

2827 A function that may be safely invoked concurrently by multiple threads. Each function defined |  
2828 in the System Interfaces volume of IEEE Std 1003.1-200x is thread-safe unless explicitly stated |  
2829 otherwise. Examples are any “pure” function, a function which holds a mutex locked while it is |  
2830 accessing static storage, or objects shared among threads. |

### 2831 **3.397 Thread-Specific Data Key** |

2832 A process global handle of type **pthread\_key\_t** which is used for naming thread-specific data. |

2833 Although the same key value may be used by different threads, the values bound to the key by |  
2834 *pthread\_setspecific()* and accessed by *pthread\_getspecific()* are maintained on a per-thread basis |  
2835 and persist for the life of the calling thread.

2836 **Note:** The *pthread\_getspecific()* and *pthread\_setspecific()* functions are defined in detail in the System |  
2837 Interfaces volume of IEEE Std 1003.1-200x.

### 2838 **3.398 Tilde**

2839 The character '~'.

**2840 3.399 Timeouts**

2841 A method of limiting the length of time an interface will block; see also Section 3.76 (on page 43).

**2842 3.400 Timer**

2843 A mechanism that can notify a thread when the time as measured by a particular clock has  
2844 reached or passed a specified value, or when a specified amount of time has passed.

**2845 3.401 Timer Overrun**

2846 A condition that occurs each time a timer, for which there is already an expiration signal queued  
2847 to the process, expires.

**2848 3.402 Token**

2849 In the shell command language, a sequence of characters that the shell considers as a single unit  
2850 when reading input. A token is either an operator or a word.

2851 **Note:** The rules for reading input are defined in detail in the Shell and Utilities volume of  
2852 IEEE Std 1003.1-200x, Section 2.3, Token Recognition.

**2853 3.403 Trace Analyzer Process**

2854 A process that extracts trace events from a trace stream to retrieve information about the  
2855 behavior of an application.

**2856 3.404 Trace Controller Process**

2857 A process that creates a trace stream for tracing a process.

**2858 3.405 Trace Event**

2859 A data object that represents an action executed by the system, and that is recorded in a trace  
2860 stream.

**2861 3.406 Trace Event Type**

2862 A data object type that defines a class of trace event.

**2863 3.407 Trace Event Type Mapping**

2864 A one-to-one mapping between trace event types and trace event names.

**2865 3.408 Trace Filter**

2866 A filter that allows the trace controller process to specify those trace event types that are to be  
2867 ignored; that is, not generated.

**2868 3.409 Trace Generation Version**

2869 A data object that is an implementation-defined character string, generated by the trace system  
2870 and describing the origin and version of the trace system. |

**2871 3.410 Trace Log** |

2872 The flushed image of a trace stream, if the trace stream is created with a trace log. |

**2873 3.411 Trace Point**

2874 An action that may cause a trace event to be generated. |

**2875 3.412 Trace Stream** |

2876 An opaque object that contains trace events plus internal data needed to interpret those trace  
2877 events. |

**2878 3.413 Trace Stream Identifier**

2879 A handle to manage tracing operations in a trace stream.

**2880 3.414 Trace System**

2881 A system that allows both system and user trace events to be generated into a trace stream.  
2882 These trace events can be retrieved later. |

**2883 3.415 Traced Process** |

2884 A process for which at least one trace stream has been created. A traced process is also called a  
2885 target process. |

**2886 3.416 Tracing Status of a Trace Stream**

2887 A status that describes the state of an active trace stream. The tracing status of a trace stream can  
2888 be retrieved from the trace stream attributes. An active trace stream can be in one of two states:  
2889 running or suspended.

**2890 3.417 Typed Memory Name Space**

2891 A system-wide name space that contains the names of the typed memory objects present in the  
2892 system. It is configurable for a given implementation.

**2893 3.418 Typed Memory Object**

2894 A combination of a typed memory pool and a typed memory port. The entire contents of the  
2895 pool are accessible from the port. The typed memory object is identified through a name that  
2896 belongs to the typed memory name space.

**2897 3.419 Typed Memory Pool**

2898 An extent of memory with the same operational characteristics. Typed memory pools may be  
2899 contained within each other.

**2900 3.420 Typed Memory Port**

2901 A hardware access path to one or more typed memory pools.

**2902 3.421 Unbind**

2903 Remove the association between a network address and an endpoint.

**2904 3.422 Unit Data**

2905 See *Datagram* in Section 3.123 (on page 49).

**2906 3.423 Upshifting**

2907 The conversion of a lowercase character that has a single-character uppercase representation  
2908 into this uppercase representation. |

**2909 3.424 User Database** |

2910 A system database of implementation-defined format that contains at least the following |  
2911 information for each user ID:

- 2912 • User name
- 2913 • Numerical user ID
- 2914 • Initial numerical group ID
- 2915 • Initial working directory
- 2916 • Initial user program

2917 The initial numerical group ID is used by the *newgrp* utility. Any other circumstances under  
2918 which the initial values are operative are implementation-defined.

2919 If the initial user program field is null, an implementation-defined program is used.

2920 If the initial working directory field is null, the interpretation of that field is implementation-  
2921 defined.

2922 **Note:** The *newgrp* utility is defined in detail in the Shell and Utilities volume of IEEE Std 1003.1-200x.

### 2923 **3.425 User ID**

2924 A non-negative integer that is used to identify a system user. When the identity of a user is  
2925 associated with a process, a user ID value is referred to as a real user ID, an effective user ID, or a  
2926 saved set-user-ID.

### 2927 **3.426 User Name**

2928 A string that is used to identify a user; see also Section 3.424 (on page 90). To be portable across  
2929 systems conforming to IEEE Std 1003.1-200x, the value is composed of characters from the  
2930 portable filename character set. The hyphen should not be used as the first character of a  
2931 portable user name.

### 2932 **3.427 User Trace Event**

2933 A trace event that is generated explicitly by the application as a result of a call to  
2934 *posix\_trace\_event()*.

### 2935 **3.428 Utility**

2936 A program, excluding special built-in utilities provided as part of the Shell Command Language,  
2937 that can be called by name from a shell to perform a specific task, or related set of tasks.

2938 **Note:** For further information on special built-in utilities, see the Shell and Utilities volume of  
2939 IEEE Std 1003.1-200x, Section 2.14, Special Built-In Utilities.

### 2940 **3.429 Variable**

2941 In the shell command language, a named parameter.

2942 **Note:** For further information, see the Shell and Utilities volume of IEEE Std 1003.1-200x, Section 2.5,  
2943 Parameters and Variables.

### 2944 **3.430 Vertical-Tab Character (<vertical-tab>)**

2945 A character that in the output stream indicates that printing should start at the next vertical  
2946 tabulation position. It is the character designated by '`\v`' in the C language. If the current  
2947 position is at or past the last defined vertical tabulation position, the behavior is unspecified. It is  
2948 unspecified whether this character is the exact sequence transmitted to an output device by the  
2949 system to accomplish the tabulation.

2950 **3.431 White Space** |

2951 A sequence of one or more characters that belong to the **space** character class as defined via the |  
 2952 *LC\_CTYPE* category in the current locale. |

2953 In the POSIX locale, white space consists of one or more <blank>s (<space>s and <tab>s), |  
 2954 <newline>s, <carriage-return>s, <form-feed>s, and <vertical-tab>s. |

2955 **3.432 Wide-Character Code (C Language)** |

2956 An integer value corresponding to a single graphic symbol or control code. |

2957 **Note:** C Language Wide-Character Codes are defined in detail in Section 6.3 (on page 115). |

2958 **3.433 Wide-Character Input/Output Functions** |

2959 The functions that perform wide-oriented input from streams or wide-oriented output to |  
 2960 streams: *fgetwc()*, *fputwc()*, *fputws()*, *fwprintf()*, *fwscanf()*, *getwc()*, *getwchar()*, *getws()*, *putwc()*, |  
 2961 *putwchar()*, *ungetwc()*, *vfwprintf()*, *vwprintf()*, *wprintf()*, and *wscanf()*. |

2962 **Note:** These functions are defined in detail in the System Interfaces volume of IEEE Std 1003.1-200x. |

2963 **3.434 Wide-Character String** |

2964 A contiguous sequence of wide-character codes terminated by and including the first null wide- |  
 2965 character code. |

2966 **3.435 Word** |

2967 In the shell command language, a token other than an operator. In some cases a word is also a |  
 2968 portion of a word token: in the various forms of parameter expansion, such as  $\${name-word}$ , and |  
 2969 variable assignment, such as *name=word*, the word is the portion of the token depicted by *word*. |  
 2970 The concept of a word is no longer applicable following word expansions—only fields remain. |

2971 **Note:** For further information, see the Shell and Utilities volume of IEEE Std 1003.1-200x, Section |  
 2972 2.6.2, Parameter Expansion and the Shell and Utilities volume of IEEE Std 1003.1-200x, Section |  
 2973 2.6, Word Expansions. |

2974 **3.436 Working Directory (or Current Working Directory)** |

2975 A directory, associated with a process, that is used in pathname resolution for pathnames that |  
 2976 do not begin with a slash. |

2977 **3.437 Worldwide Portability Interface** |

2978 Functions for handling characters in a codeset-independent manner. |

2979 **3.438 Write**

2980 To output characters to a file, such as standard output or standard error. Unless otherwise  
2981 stated, standard output is the default output destination for all uses of the term write; see the  
2982 distinction between display and write in Section 3.132 (on page 50).

2983 **3.439 XSI**

2984 The X/Open System Interface is the core application programming interface for C and *sh*  
2985 programming for systems conforming to the Single UNIX Specification. This is a superset of the  
2986 mandatory requirements for conformance to IEEE Std 1003.1-200x. |

2987 **3.440 XSI-Conformant** |

2988 A system which allows an application to be built using a set of services that are consistent across |  
2989 all systems that conform to IEEE Std 1003.1-200x and that support the XSI extension. |

2990 **Note:** See also Chapter 2 (on page 15).

2991 **3.441 Zombie Process**

2992 A process that has terminated and that is deleted when its exit status has been reported to  
2993 another process which is waiting for that process to terminate.

2994 **3.442  $\pm 0$** 

2995 The algebraic sign provides additional information about any variable that has the value zero  
2996 when the representation allows the sign to be determined. |



# General Concepts

2998

2999 For the purposes of IEEE Std 1003.1-200x, the general concepts given in Chapter 4 apply.

3000 **Note:** No shading to denote extensions or options occurs in this chapter. Where the terms and  
 3001 definitions given in this chapter are used elsewhere in text related to extensions and options,  
 3002 they are shaded as appropriate.

## 3003 4.1 Concurrent Execution

3004 Functions that suspend the execution of the calling thread shall not cause the execution of other  
 3005 threads to be indefinitely suspended.

## 3006 4.2 Directory Protection

3007 If a directory is writable and the mode bit `S_ISVTX` is set on the directory, a process may remove  
 3008 or rename files within that directory only if one or more of the following is true:

- 3009 • The effective user ID of the process is the same as that of the owner ID of the file.
- 3010 • The effective user ID of the process is the same as that of the owner ID of the directory.
- 3011 • The process has appropriate privileges.

3012 If the `S_ISVTX` bit is set on a non-directory file, the behavior is unspecified.

## 3013 4.3 Extended Security Controls

3014 An implementation may provide implementation-defined extended security controls (see  
 3015 Section 3.159 (on page 54)). These permit an implementation to provide security mechanisms to  
 3016 implement different security policies than those described in IEEE Std 1003.1-200x. These  
 3017 mechanisms shall not alter or override the defined semantics of any of the interfaces in  
 3018 IEEE Std 1003.1-200x.

## 3019 4.4 File Access Permissions

3020 The standard file access control mechanism uses the file permission bits, as described below.

3021 Implementations may provide *additional* or *alternate* file access control mechanisms, or both. An  
 3022 additional access control mechanism shall only further restrict the access permissions defined by  
 3023 the file permission bits. An alternate file access control mechanism shall:

- 3024 • Specify file permission bits for the file owner class, file group class, and file other class of that  
 3025 file, corresponding to the access permissions.
- 3026 • Be enabled only by explicit user action, on a per-file basis by the file owner or a user with the  
 3027 appropriate privilege.
- 3028 • Be disabled for a file after the file permission bits are changed for that file with `chmod()`. The  
 3029 disabling of the alternate mechanism need not disable any additional mechanisms supported

3030 by an implementation.

3031 Whenever a process requests file access permission for read, write, or execute/search, if no |  
3032 additional mechanism denies access, access shall be determined as follows: |

3033 • If a process has the appropriate privilege:

3034 — If read, write, or directory search permission is requested, access shall be granted. |

3035 — If execute permission is requested, access shall be granted if execute permission is |  
3036 granted to at least one user by the file permission bits or by an alternate access control |  
3037 mechanism; otherwise, access shall be denied. |

3038 • Otherwise:

3039 — The file permission bits of a file contain read, write, and execute/search permissions for |  
3040 the file owner class, file group class, and file other class.

3041 — Access shall be granted if an alternate access control mechanism is not enabled and the |  
3042 requested access permission bit is set for the class (file owner class, file group class, or file |  
3043 other class) to which the process belongs, or if an alternate access control mechanism is |  
3044 enabled and it allows the requested access; otherwise, access shall be denied. |

## 3045 4.5 File Hierarchy

3046 Files in the system are organized in a hierarchical structure in which all of the non-terminal |  
3047 nodes are directories and all of the terminal nodes are any other type of file. Since multiple |  
3048 directory entries may refer to the same file, the hierarchy is properly described as a *directed* |  
3049 *graph*.

## 3050 4.6 Filenames

3051 For a filename to be portable across implementations conforming to IEEE Std 1003.1-200x, it |  
3052 shall consist only of the portable filename character set as defined in Section 3.276 (on page 70). |

3053 The hyphen character shall not be used as the first character of a portable filename. Uppercase |  
3054 and lowercase letters shall retain their unique identities between conforming implementations. |  
3055 In the case of a portable pathname, the slash character may also be used.

## 3056 4.7 File Times Update

3057 Each file has three distinct associated time values: *st\_atime*, *st\_mtime*, and *st\_ctime*. The *st\_atime* |  
3058 field is associated with the times that the file data is accessed; *st\_mtime* is associated with the |  
3059 times that the file data is modified; and *st\_ctime* is associated with the times that the file status is |  
3060 changed. These values are returned in the file characteristics structure, as described in |  
3061 `<sys/stat.h>`.

3062 Each function or utility in IEEE Std 1003.1-200x that reads or writes data or changes file status |  
3063 indicates which of the appropriate time-related fields shall be “marked for update”. If an |  
3064 implementation of such a function or utility marks for update a time-related field not specified |  
3065 by IEEE Std 1003.1-200x, this shall be documented, except that any changes caused by pathname |  
3066 resolution need not be documented. For the other functions or utilities in IEEE Std 1003.1-200x |  
3067 (those that are not explicitly required to read or write file data or change file status, but that in |  
3068 some implementations happen to do so), the effect is unspecified.

3069 An implementation may update fields that are marked for update immediately, or it may update  
3070 such fields periodically. At an update point in time, any marked fields shall be set to the current  
3071 time and the update marks shall be cleared. All fields that are marked for update shall be  
3072 updated when the file ceases to be open by any process, or when a *stat()*, *fstat()*, or *lstat()* is  
3073 performed on the file. Other times at which updates are done are unspecified. Marks for update,  
3074 and updates themselves, are not done for files on read-only file systems; see Section 3.304 (on  
3075 page 74).

## 3076 **4.8 Host and Network Byte Orders**

3077 When data is transmitted over the network, it is sent as a sequence of octets (8-bit unsigned  
3078 values). If an entity (such as an address or a port number) can be larger than 8 bits, it needs to be  
3079 stored in several octets. The convention is that all such values are stored with 8 bits in each octet,  
3080 and with the first (lowest-addressed) octet holding the most-significant bits. This is called  
3081 “network byte order”.

3082 Network byte order may not be convenient for processing actual values. For this, it is more  
3083 sensible for values to be stored as ordinary integers. This is known as “host byte order”. In host  
3084 byte order:

- 3085 • The most significant bit might not be stored in the first byte in address order.
- 3086 • Bits might not be allocated to bytes in any obvious order at all.

3087 8-bit values stored in **uint8\_t** objects do not require conversion to or from host byte order, as  
3088 they have the same representation. 16 and 32-bit values can be converted using the *htonl()*,  
3089 *htons()*, *ntohl()*, and *ntohs()* functions. When reading data that is to be converted to host byte  
3090 order, it should either be received directly into a **uint16\_t** or **uint32\_t** object or should be copied  
3091 from an array of bytes using *memcpy()* or similar. Passing the data through other types could  
3092 cause the byte order to be changed. Similar considerations apply when sending data.

## 3093 **4.9 Measurement of Execution Time**

3094 The mechanism used to measure execution time shall be implementation-defined. The  
3095 implementation shall also define to whom the CPU time that is consumed by interrupt handlers  
3096 and system services on behalf of the operating system will be charged. See Section 3.117 (on  
3097 page 49).

## 3098 4.10 Memory Synchronization

3099 Applications shall ensure that access to any memory location by more than one thread of control  
 3100 (threads or processes) is restricted such that no thread of control can read or modify a memory  
 3101 location while another thread of control may be modifying it. Such access is restricted using  
 3102 functions that synchronize thread execution and also synchronize memory with respect to other  
 3103 threads. The following functions synchronize memory with respect to other threads:

3104	<i>fork()</i>	<i>pthread_mutex_timedlock()</i>	<i>pthread_rwlock_tryrdlock()</i>
3105	<i>pthread_barrier_wait()</i>	<i>pthread_mutex_trylock()</i>	<i>pthread_rwlock_trywrlock()</i>
3106	<i>pthread_cond_broadcast()</i>	<i>pthread_mutex_unlock()</i>	<i>pthread_rwlock_unlock()</i>
3107	<i>pthread_cond_signal()</i>	<i>pthread_spin_lock()</i>	<i>pthread_rwlock_wrlock()</i>
3108	<i>pthread_cond_timedwait()</i>	<i>pthread_spin_trylock()</i>	<i>sem_post()</i>
3109	<i>pthread_cond_wait()</i>	<i>pthread_spin_unlock()</i>	<i>sem_trywait()</i>
3110	<i>pthread_create()</i>	<i>pthread_rwlock_rdlock()</i>	<i>sem_wait()</i>
3111	<i>pthread_join()</i>	<i>pthread_rwlock_timedrdlock()</i>	<i>wait()</i>
3112	<i>pthread_mutex_lock()</i>	<i>pthread_rwlock_timedwrlock()</i>	<i>waitpid()</i>

3113 Unless explicitly stated otherwise, if one of the above functions returns an error, it is unspecified  
 3114 whether the invocation causes memory to be synchronized.

3115 Applications may allow more than one thread of control to read a memory location  
 3116 simultaneously.

## 3117 4.11 Pathname Resolution

3118 Pathname resolution is performed for a process to resolve a pathname to a particular file in a file  
 3119 hierarchy. There may be multiple pathnames that resolve to the same file.

3120 Each filename in the pathname is located in the directory specified by its predecessor (for  
 3121 example, in the pathname fragment **a/b**, file **b** is located in directory **a**). Pathname resolution  
 3122 shall fail if this cannot be accomplished. If the pathname begins with a slash, the predecessor of  
 3123 the first filename in the pathname shall be taken to be the root directory of the process (such  
 3124 pathnames are referred to as *absolute pathnames*). If the pathname does not begin with a slash, the  
 3125 predecessor of the first filename of the pathname shall be taken to be the current working  
 3126 directory of the process (such pathnames are referred to as *relative pathnames*).

3127 The interpretation of a pathname component is dependent on the value of {NAME\_MAX} and  
 3128 \_POSIX\_NO\_TRUNC associated with the path prefix of that component. If any pathname  
 3129 component is longer than {NAME\_MAX}, the implementation shall consider this an error.

3130 A pathname that contains at least one non-slash character and that ends with one or more  
 3131 trailing slashes shall be resolved as if a single dot character ( '.' ) were appended to the  
 3132 pathname.

3133 If a symbolic link is encountered during pathname resolution, the behavior shall depend on  
 3134 whether the pathname component is at the end of the pathname and on the function being  
 3135 performed. If all of the following are true, then pathname resolution is complete:

- 3136 1. This is the last pathname component of the pathname.
- 3137 2. The pathname has no trailing slash.
- 3138 3. The function is required to act on the symbolic link itself, or certain arguments direct that  
 3139 the function act on the symbolic link itself.

3140 In all other cases, the system shall prefix the remaining pathname, if any, with the contents of the  
 3141 symbolic link. If the combined length exceeds {PATH\_MAX}, and the implementation considers  
 3142 this to be an error, *errno* shall be set to [ENAMETOOLONG] and an error indication shall be  
 3143 returned. Otherwise, the resolved pathname shall be the resolution of the pathname just created.  
 3144 If the resulting pathname does not begin with a slash, the predecessor of the first filename of the  
 3145 pathname is taken to be the directory containing the symbolic link.

3146 If the system detects a loop in the pathname resolution process, it shall set *errno* to [ELOOP] and  
 3147 return an error indication. The same may happen if during the resolution process more symbolic  
 3148 links were followed than the implementation allows. This implementation-defined limit shall  
 3149 not be smaller than {SYMLOOP\_MAX}.

3150 The special filename dot shall refer to the directory specified by its predecessor. The special |  
 3151 filename dot-dot shall refer to the parent directory of its predecessor directory. As a special case, |  
 3152 in the root directory, dot-dot may refer to the root directory itself.

3153 A pathname consisting of a single slash shall resolve to the root directory of the process. A null |  
 3154 pathname shall not be successfully resolved. A pathname that begins with two successive |  
 3155 slashes may be interpreted in an implementation-defined manner, although more than two |  
 3156 leading slashes shall be treated as a single slash.

## 3157 4.12 Process ID Reuse

3158 A process group ID shall not be reused by the system until the process group lifetime ends.

3159 A process ID shall not be reused by the system until the process lifetime ends. In addition, if  
 3160 there exists a process group whose process group ID is equal to that process ID, the process ID  
 3161 shall not be reused by the system until the process group lifetime ends. A process that is not a  
 3162 system process shall not have a process ID of 1.

## 3163 4.13 Scheduling Policy

3164 A scheduling policy affects process or thread ordering:

- 3165 • When a process or thread is a running thread and it becomes a blocked thread
- 3166 • When a process or thread is a running thread and it becomes a preempted thread
- 3167 • When a process or thread is a blocked thread and it becomes a runnable thread
- 3168 • When a running thread calls a function that can change the priority or scheduling policy of a  
 3169 process or thread
- 3170 • In other scheduling policy-defined circumstances

3171 Conforming implementations shall define the manner in which each of the scheduling policies |  
 3172 may modify the priorities or otherwise affect the ordering of processes or threads at each of the |  
 3173 occurrences listed above. Additionally, conforming implementations shall define in what other |  
 3174 circumstances and in what manner each scheduling policy may modify the priorities or affect |  
 3175 the ordering of processes or threads.

## 3176 4.14 Seconds Since the Epoch

3177 A value that approximates the number of seconds that have elapsed since the Epoch. A  
 3178 Coordinated Universal Time name (specified in terms of seconds (*tm\_sec*), minutes (*tm\_min*),  
 3179 hours (*tm\_hour*), days since January 1 of the year (*tm\_yday*), and calendar year minus 1900  
 3180 (*tm\_year*)) is related to a time represented as seconds since the Epoch, according to the  
 3181 expression below.

3182 If the year is <1970 or the value is negative, the relationship is undefined. If the year is ≥1970 and  
 3183 the value is non-negative, the value is related to a Coordinated Universal Time name according  
 3184 to the C-language expression, where *tm\_sec*, *tm\_min*, *tm\_hour*, *tm\_yday*, and *tm\_year* are all  
 3185 integer types:

$$\begin{aligned}
 &tm\_sec + tm\_min*60 + tm\_hour*3600 + tm\_yday*86400 + \\
 & (tm\_year-70)*31536000 + ((tm\_year-69)/4)*86400 - \\
 & ((tm\_year-1)/100)*86400 + ((tm\_year+299)/400)*86400
 \end{aligned}$$

3189 The relationship between the actual time of day and the current value for seconds since the  
 3190 Epoch is unspecified.

3191 How any changes to the value of seconds since the Epoch are made to align to a desired  
 3192 relationship with the current actual time are made is implementation-defined. As represented in  
 3193 seconds since the Epoch, each and every day shall be accounted for by exactly 86 400 seconds.

3194 **Note:** The last three terms of the expression add in a day for each year that follows a leap year  
 3195 starting with the first leap year since the Epoch. The first term adds a day every 4 years |  
 3196 starting in 1973, the second subtracts a day back out every 100 years starting in 2001, and the |  
 3197 third adds a day back in every 400 years starting in 2001. The divisions in the formula are |  
 3198 integer divisions; that is, the remainder is discarded leaving only the integer quotient. |

## 3199 4.15 Semaphore

3200 A minimum synchronization primitive to serve as a basis for more complex synchronization  
 3201 mechanisms to be defined by the application program.

3202 For the semaphores associated with the Semaphores option, a semaphore is represented as a  
 3203 shareable resource that has a non-negative integer value. When the value is zero, there is a  
 3204 (possibly empty) set of threads awaiting the availability of the semaphore.

3205 For the semaphores associated with the X/Open System Interface Extension (XSI), a semaphore  
 3206 is a positive integer (0 through 32767). The *semget()* function can be called to create a set or array  
 3207 of semaphores. A semaphore set can contain one or more semaphores up to an implementation-  
 3208 defined value.

### 3209 Semaphore Lock Operation

3210 An operation that is applied to a semaphore. If, prior to the operation, the value of the  
 3211 semaphore is zero, the semaphore lock operation shall cause the calling thread to be blocked and  
 3212 added to the set of threads awaiting the semaphore; otherwise, the value shall be decremented. |

3213 **Semaphore Unlock Operation**

3214 An operation that is applied to a semaphore. If, prior to the operation, there are any threads in  
 3215 the set of threads awaiting the semaphore, then some thread from that set shall be removed from  
 3216 the set and becomes unblocked; otherwise, the semaphore value shall be incremented. |

3217 **4.16 Thread-Safety**

3218 Refer to the System Interfaces volume of IEEE Std 1003.1-200x, Section 2.9, Threads.

3219 **4.17 Tracing**

3220 The trace system allows a traced process to have a selection of events created for it. Traces  
 3221 consist of streams of trace event types.

3222 A trace event type is identified on the one hand by a trace event type name, also referenced as a  
 3223 trace event name, and on the other hand by a trace event type identifier. A trace event name is a  
 3224 human-readable string. A trace event type identifier is an opaque identifier used by the trace |  
 3225 system. There shall be a one-to-one relationship between trace event type identifiers and trace |  
 3226 event names for a given trace stream and also for a given traced process. The trace event type |  
 3227 identifier shall be generated automatically from a trace event name by the trace system either |  
 3228 when a trace controller process invokes *posix\_trace\_trid\_eventid\_open()* or when an instrumented |  
 3229 application process invokes *posix\_trace\_eventid\_open()*. Trace event type identifiers are used to |  
 3230 filter trace event types, to allow interpretation of user data, and to identify the kind of trace point  
 3231 that generated a trace event.

3232 Each trace event shall be of a particular trace event type, and associated with a trace event type |  
 3233 identifier. The execution of a trace point shall generate a trace event if a trace stream has been |  
 3234 created and started for the process that executed the trace point and if the corresponding trace |  
 3235 event type identifier is not ignored by filtering. |

3236 A generated trace event shall be recorded in a trace stream, and optionally also in a trace log if a  
 3237 trace log is associated with the trace stream, except that:

- 3238 • For a trace stream, if no resources are available for the event, the event is lost.
- 3239 • For a trace log, if no resources are available for the event, or a flush operation does not  
 3240 succeed, the event is lost.

3241 A trace event recorded in an active trace stream may be retrieved by an application having the  
 3242 appropriate privileges.

3243 A trace event recorded in a trace log may be retrieved by an application having the appropriate  
 3244 privileges after opening the trace log as a pre-recorded trace stream, with the function  
 3245 *posix\_trace\_open()*.

3246 When a trace event is reported it is possible to retrieve the following:

- 3247 • A trace event type identifier
- 3248 • A timestamp
- 3249 • The process ID of the traced process, if the trace event is process-dependent
- 3250 • Any optional trace event data including its length

- 3251           • If the Threads option is supported, the thread ID, if the trace event is process-dependent  
 3252           • The program address at which the trace point was invoked
- 3253           Trace events may be mapped from trace event types to trace event names. One such mapping |  
 3254           shall be associated with each trace stream. An active trace stream is associated with a traced |  
 3255           process, and also with its children if the Trace Inherit option is supported and also the |  
 3256           inheritance policy is set to `_POSIX_TRACE_INHERIT`. Therefore each traced process has a |  
 3257           mapping of the trace event names to trace event type identifiers that have been defined for that |  
 3258           process. |
- 3259           Traces can be recorded into either trace streams or trace logs. |
- 3260           The implementation and format of a trace stream are unspecified. A trace stream need not be |  
 3261           and generally is not persistent. A trace stream may be either active or pre-recorded:
- 3262           • An active trace stream is a trace stream that has been created and has not yet been shut |  
 3263           down. It can be of one of the two following classes:
- 3264                1. An active trace stream without a trace log that was created with the *posix\_trace\_create()* |  
 3265                function
- 3266                2. If the Trace Log option is supported, an active trace stream with a trace log that was |  
 3267                created with the *posix\_trace\_create\_withlog()* function
- 3268           • A pre-recorded trace stream is a trace stream that was opened from a trace log object using |  
 3269           the *posix\_trace\_open()* function.
- 3270           An active trace stream can loop. This behavior means that when the resources allocated by the |  
 3271           trace system for the trace stream are exhausted, the trace system reuses the resources associated |  
 3272           with the oldest recorded trace events to record new trace events.
- 3273           If the Trace Log option is supported, an active trace stream with a trace log can be flushed. This |  
 3274           operation causes the trace system to write trace events from the trace stream to the associated |  
 3275           trace log, following the defined policies or using an explicit function call. After this operation, |  
 3276           the trace system may reuse the resources associated with the flushed trace events.
- 3277           An active trace stream with or without a trace log can be cleared. This operation shall cause all |  
 3278           the resources associated with this trace stream to be reinitialized. The trace stream shall behave |  
 3279           as if it was returning from its creation, except that the mapping of trace event type identifiers to |  
 3280           trace event names shall not be cleared. If a trace log was associated with this trace stream, the |  
 3281           trace log shall also be reinitialized. |
- 3282           A trace log shall be recorded when the *posix\_trace\_shutdown()* operation is invoked or during |  
 3283           tracing, depending on the tracing strategy which is defined by a log policy. After the trace |  
 3284           stream has been shut down, the trace information can be retrieved from the associated trace log |  
 3285           using the same interface used to retrieve information from an active trace stream.
- 3286           For a traced process, if the Trace Inherit option is supported and the trace stream's inheritance |  
 3287           attribute is `_POSIX_TRACE_INHERIT`, the initial targeted traced process shall be traced together |  
 3288           with all of its future children. The *posix\_pid* member of each trace event in a trace stream shall be |  
 3289           the process ID of the traced process. |
- 3290           Each trace point may be an implementation-defined action such as a context switch, or an |  
 3291           application-programmed action such as a call to a specific operating system service (for |  
 3292           example, *fork()*) or a call to *posix\_trace\_event()*.
- 3293           Trace points may be filtered. The operation of the filter is to filter out (ignore) selected trace |  
 3294           events. By default, no trace events are filtered.

3295 The results of the tracing operations can be analyzed and monitored by a trace controller process  
3296 or a trace analyzer process.

3297 Only the trace controller process has control of the trace stream it has created. The control of the  
3298 operation of a trace stream is done using its corresponding trace stream identifier. The trace  
3299 controller process is able to:

- 3300 • Initialize the attributes of a trace stream
- 3301 • Create the trace stream
- 3302 • Start and stop tracing
- 3303 • Know the mapping of the traced process
- 3304 • If the Trace Event Filter option is supported, filter the type of trace events to be recorded
- 3305 • Shut the trace stream down

3306 A traced process may also be a trace controller process. Only the trace controller process can |  
3307 control its trace stream(s). A trace stream created by a trace controller process shall be shut |  
3308 down if its controller process terminates or executes another file. |

3309 A trace controller process may also be a trace analyzer process. Trace analysis can be done  
3310 concurrently with the traced process or can be done off-line, in the same or in a different  
3311 platform.

## 3312 **4.18 Treatment of Error Conditions for Mathematical Functions**

3313 For all the functions in the `<math.h>` header, an application wishing to check for error situations  
3314 should set `errno` to 0 and call `feclearexcept(FE_ALL_EXCEPT)` before calling the function. On  
3315 return, if `errno` is non-zero or `fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW |`  
3316 `FE_UNDERFLOW)` is non-zero, an error has occurred.

3317 The following error conditions are defined for all functions in the `<math.h>` header.

### 3318 **4.18.1 Domain Error**

3319 A *domain error* shall occur if an input argument is outside the domain over which the  
3320 mathematical function is defined. The description of each function lists any required domain  
3321 errors; an implementation may define additional domain errors, provided that such errors are  
3322 consistent with the mathematical definition of the function.

3323 On a domain error, the function shall return an implementation-defined value; if the integer |  
3324 expression (`math_errhandling & MATH_ERRNO`) is non-zero, `errno` shall be set to [EDOM]; if |  
3325 the integer expression (`math_errhandling & MATH_ERREXCEPT`) is non-zero, the “invalid” |  
3326 floating-point exception shall be raised. |

3327 **4.18.2 Pole Error**

3328 A *pole error* occurs if the mathematical result of the function is an exact infinity (for example,  
3329  $\log(0.0)$ ).

3330 On a pole error, the function shall return the value of the macro HUGE\_VAL, HUGE\_VALF, or  
3331 HUGE\_VALL according to the return type, with the same sign as the correct value of the  
3332 function; if the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero, *errno* shall  
3333 be set to [ERANGE]; if the integer expression (math\_errhandling & MATH\_ERREXCEPT) is  
3334 non-zero, the “divide-by-zero” floating-point exception shall be raised.

3335 **4.18.3 Range Error**

3336 A *range error* shall occur if the finite mathematical result of the function cannot be represented in  
3337 an object of the specified type, due to extreme magnitude.

3338 **4.18.3.1 Result Overflows**

3339 A floating result overflows if the magnitude of the mathematical result is finite but so large that  
3340 the mathematical result cannot be represented without extraordinary roundoff error in an object  
3341 of the specified type. If a floating result overflows and default rounding is in effect, then the  
3342 function shall return the value of the macro HUGE\_VAL, HUGE\_VALF, or HUGE\_VALL  
3343 according to the return type, with the same sign as the correct value of the function; if the integer  
3344 expression (math\_errhandling & MATH\_ERRNO) is non-zero, *errno* shall be set to [ERANGE]; if  
3345 the integer expression (math\_errhandling & MATH\_ERREXCEPT) is non-zero, the “overflow”  
3346 floating-point exception shall be raised.

3347 **4.18.3.2 Result Underflows**

3348 The result underflows if the magnitude of the mathematical result is so small that the  
3349 mathematical result cannot be represented, without extraordinary roundoff error, in an object of  
3350 the specified type. If the result underflows, the function shall return an implementation-defined  
3351 value whose magnitude is no greater than the smallest normalized positive number in the  
3352 specified type; if the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero,  
3353 whether *errno* is set to [ERANGE] is implementation-defined; if the integer expression  
3354 (math\_errhandling & MATH\_ERREXCEPT) is non-zero, whether the “underflow” floating-point  
3355 exception is raised is implementation-defined.

3356 **4.19 Treatment of NaN Arguments for the Mathematical Functions**

3357 For functions called with a NaN argument, no errors shall occur and a NaN shall be returned,  
3358 except where stated otherwise.

3359 If a function with one or more NaN arguments returns a NaN result, the result should be the  
3360 same as one of the NaN arguments (after possible type conversion), except perhaps for the sign.

3361 On implementations that support the IEC 60559:1989 standard floating point, functions with  
3362 signaling NaN argument(s) shall be treated as if the function were called with an argument that  
3363 is a required domain error and shall return a quiet NaN result, except where stated otherwise.

3364 **Note:** The function might never see the signaling NaN, since it might trigger when the arguments are  
3365 evaluated during the function call.

3366 On implementations that support the IEC 60559:1989 standard floating point, for those  
3367 functions that do not have a documented domain error, the following shall apply:

3368           These functions shall fail if:  
 3369           Domain Error    Any argument is a signaling NaN.  
 3370           Either, the integer expression (`math_errhandling & MATH_ERRNO`) is non-zero and *errno*  
 3371           shall be set to [EDOM], or the integer expression (`math_errhandling & MATH_ERREXCEPT`)  
 3372           is non-zero and the invalid floating-point exception shall be raised.

## 3373 4.20 Utility

3374           A utility program shall be either an executable file, such as might be produced by a compiler or  
 3375           linker system from computer source code, or a file of shell source code, directly interpreted by  
 3376           the shell. The program may have been produced by the user, provided by the system  
 3377           implementor, or acquired from an independent distributor.

3378           The system may implement certain utilities as shell functions (see the Shell and Utilities volume  
 3379           of IEEE Std 1003.1-200x, Section 2.9.5, Function Definition Command) or built-in utilities, but  
 3380           only an application that is aware of the command search order described in the Shell and  
 3381           Utilities volume of IEEE Std 1003.1-200x, Section 2.9.1.1, Command Search and Execution or of  
 3382           performance characteristics can discern differences between the behavior of such a function or  
 3383           built-in utility and that of an executable file.

## 3384 4.21 Variable Assignment

3385           In the shell command language, a word consisting of the following parts:

3386            `varname=value`

3387           When used in a context where assignment is defined to occur and at no other time, the *value*  
 3388           (representing a word or field) shall be assigned as the value of the variable denoted by *varname*.

3389           **Note:**    For further information, see the Shell and Utilities volume of IEEE Std 1003.1-200x, Section  
 3390           2.9.1, Simple Commands.

3391           The *varname* and *value* parts shall meet the requirements for a name and a word, respectively, |  
 3392           except that they are delimited by the embedded unquoted equals-sign, in addition to other |  
 3393           delimiters. |

3394           **Note:**    Additional delimiters are described in the Shell and Utilities volume of IEEE Std 1003.1-200x,  
 3395           Section 2.3, Token Recognition.

3396           When a variable assignment is done, the variable shall be created if it did not already exist. If  
 3397           *value* is not specified, the variable shall be given a null value.

3398           **Note:**    An alternative form of variable assignment:

3399            `symbol=value`

3400           (where *symbol* is a valid word delimited by an equals-sign, but not a valid name) produces  
 3401           unspecified results. The form `symbol=value` is used by the KornShell `name[expression]=value`  
 3402           syntax.



**File Format Notation**

3404

3405 The STDIN, STDOUT, STDERR, INPUT FILES, and OUTPUT FILES sections of the utility  
 3406 descriptions use a syntax to describe the data organization within the files, when that  
 3407 organization is not otherwise obvious. The syntax is similar to that used by the System Interfaces  
 3408 volume of IEEE Std 1003.1-200x *printf()* function, as described in this chapter. When used in  
 3409 STDIN or INPUT FILES sections of the utility descriptions, this syntax describes the format that  
 3410 could have been used to write the text to be read, not a format that could be used by the System  
 3411 Interfaces volume of IEEE Std 1003.1-200x *scanf()* function to read the input file.

3412 The description of an individual record is as follows:

3413       "*format*", [*arg1*, *arg2*, . . . , *argn*]

3414 The *format* is a character string that contains three types of objects defined below:

- 3415       1. *Characters* that are not *escape sequences* or *conversion specifications*, as described below, shall  
 3416       be copied to the output.
- 3417       2. *Escape Sequences* represent non-graphic characters.
- 3418       3. *Conversion Specifications* specify the output format of each argument; (see below).

3419 The following characters have the following special meaning in the format string:

3420       ' ' (An empty character position.) Represents one or more <blank>s.

3421       Δ Represents exactly one <space>.

3422 Table 5-1 lists escape sequences and associated actions on display devices capable of the action.

3423

Table 5-1 Escape Sequences and Associated Actions

3424

3425

3426

3427

3428

3429

3430

3431

3432

3433

3434

3435

3436

3437

3438

3439

Escape Sequence	Represents Character	Terminal Action
'\\'	backslash	Print the character '\\ '.
'\a'	alert	Attempt to alert the user through audible or visible notification.
'\b'	backspace	Move the printing position to one column before the current position, unless the current position is the start of a line.
'\f'	form-feed	Move the printing position to the initial printing position of the next logical page.
'\n'	newline	Move the printing position to the start of the next line.
'\r'	carriage-return	Move the printing position to the start of the current line.
'\t'	tab	Move the printing position to the next tab position on the current line. If there are no more tab positions remaining on the line, the behavior is undefined.
'\v'	vertical-tab	Move the printing position to the start of the next vertical tab position. If there are no more vertical tab positions left on the page, the behavior is undefined.

3440

3441

Each conversion specification is introduced by the percent-sign character ('%'). After the character '%', the following shall appear in sequence:

3442

3443

*flags* Zero or more *flags*, in any order, that modify the meaning of the conversion specification.

3444

3445

3446

3447

*field width* An optional string of decimal digits to specify a minimum *field width*. For an output field, if the converted value has fewer bytes than the field width, it shall be padded on the left (or right, if the left-adjustment flag ('-'), described below, has been given) to the field width.

3448

3449

3450

3451

3452

3453

3454

*precision* Gives the minimum number of digits to appear for the *d*, *o*, *i*, *u*, *x*, or *X* conversion specifiers (the field is padded with leading zeros), the number of digits to appear after the radix character for the *e* and *f* conversion specifiers, the maximum number of significant digits for the *g* conversion specifier; or the maximum number of bytes to be written from a string in the *s* conversion specifier. The precision shall take the form of a period ('.') followed by a decimal digit string; a null digit string is treated as zero.

3455

3456

3457

*conversion specifier characters*

A conversion specifier character (see below) that indicates the type of conversion to be applied.

3458

The *flag* characters and their meanings are:

3459

3460

3461

3462

3463

3464

3465

3466

3467

- The result of the conversion shall be left-justified within the field.

+ The result of a signed conversion shall always begin with a sign ('+' or '-').

<space> If the first character of a signed conversion is not a sign, a <space> shall be prefixed to the result. This means that if the <space> and '+' flags both appear, the <space> flag shall be ignored.

# The value shall be converted to an alternative form. For *c*, *d*, *i*, *u*, and *s* conversion specifiers, the behavior is undefined. For the *o* conversion specifier, it shall increase the precision to force the first digit of the result to be a zero. For *x* or *X* conversion specifiers, a non-zero result has 0x or 0X prefixed to it, respectively. For

3468		e, E, f, g, and G conversion specifiers, the result shall always contain a radix character, even if no digits follow the radix character. For g and G conversion specifiers, trailing zeros shall not be removed from the result as they usually are.
3469		
3470		
3471	0	For d, i, o, u, x, X, e, E, f, g, and G conversion specifiers, leading zeros (following any indication of sign or base) shall be used to pad to the field width; no space padding is performed. If the '0' and '-' flags both appear, the '0' flag shall be ignored. For d, i, o, u, x, and X conversion specifiers, if a precision is specified, the '0' flag shall be ignored. For other conversion specifiers, the behavior is undefined.
3472		
3473		
3474		
3475		
3476		
3477		Each conversion specifier character shall result in fetching zero or more arguments. The results are undefined if there are insufficient arguments for the format. If the format is exhausted while arguments remain, the excess arguments shall be ignored.
3478		
3479		
3480		The <i>conversion specifiers</i> and their meanings are:
3481	d,i,o,u,x,X	The integer argument shall be written as signed decimal (d or i), unsigned octal (o), unsigned decimal (u), or unsigned hexadecimal notation (x and X). The d and i specifiers shall convert to signed decimal in the style "[−]dddd". The x conversion specifier shall use the numbers and letters "0123456789abcdef" and the X conversion specifier shall use the numbers and letters "0123456789ABCDEF". The <i>precision</i> component of the argument shall specify the minimum number of digits to appear. If the value being converted can be represented in fewer digits than the specified minimum, it shall be expanded with leading zeros. The default precision shall be 1. The result of converting a zero value with a precision of 0 shall be no characters. If both the field width and precision are omitted, the implementation may precede, follow, or precede and follow numeric arguments of types d, i, and u with <blank>s; arguments of type o (octal) may be preceded with leading zeros.
3482		
3483		
3484		
3485		
3486		
3487		
3488		
3489		
3490		
3491		
3492		
3493		
3494	f	The floating-point number argument shall be written in decimal notation in the style [−]ddd.ddd, where the number of digits after the radix character (shown here as a decimal point) shall be equal to the <i>precision</i> specification. The <i>LC_NUMERIC</i> locale category shall determine the radix character to use in this format. If the <i>precision</i> is omitted from the argument, six digits shall be written after the radix character; if the <i>precision</i> is explicitly 0, no radix character shall appear.
3495		
3496		
3497		
3498		
3499		
3500	e,E	The floating-point number argument shall be written in the style [−]d.ddde±dd (the symbol '±' indicates either a plus or minus sign), where there is one digit before the radix character (shown here as a decimal point) and the number of digits after it is equal to the precision. The <i>LC_NUMERIC</i> locale category shall determine the radix character to use in this format. When the precision is missing, six digits shall be written after the radix character; if the precision is 0, no radix character shall appear. The E conversion specifier shall produce a number with E instead of e introducing the exponent. The exponent shall always contain at least two digits. However, if the value to be written requires an exponent greater than two digits, additional exponent digits shall be written as necessary.
3501		
3502		
3503		
3504		
3505		
3506		
3507		
3508		
3509		
3510	g,G	The floating-point number argument shall be written in style f or e (or in style F or E in the case of a G conversion specifier), with the precision specifying the number of significant digits. The style used depends on the value converted: style e (or E) shall be used only if the exponent resulting from the conversion is less than −4 or greater than or equal to the precision. Trailing zeros are removed from the result. A radix character shall appear only if it is followed by a digit.
3511		
3512		
3513		
3514		
3515		

3516 c The integer argument shall be converted to an **unsigned char** and the resulting  
 3517 byte shall be written.

3518 s The argument shall be taken to be a string and bytes from the string shall be  
 3519 written until the end of the string or the number of bytes indicated by the *precision*  
 3520 specification of the argument is reached. If the precision is omitted from the  
 3521 argument, it shall be taken to be infinite, so all bytes up to the end of the string  
 3522 shall be written.

3523 % Write a '%' character; no argument is converted.

3524 In no case does a nonexistent or insufficient *field width* cause truncation of a field; if the result of  
 3525 a conversion is wider than the field width, the field is simply expanded to contain the conversion  
 3526 result. The term *field width* should not be confused with the term *precision* used in the description  
 3527 of %s.

### 3528 Examples

3529 To represent the output of a program that prints a date and time in the form Sunday, July 3,  
 3530 10:02, where *weekday* and *month* are strings:

3531 "%s, %s %d, %d: %.2d\n" <weekday>, <month>, <day>, <hour>, <min>

3532 To show ' $\pi$ ' written to 5 decimal places:

3533 "pi = %.5f\n", <value of  $\pi$ >

3534 To show an input file format consisting of five colon-separated fields:

3535 "%s: %s: %s: %s: %s\n", <arg1>, <arg2>, <arg3>, <arg4>, <arg5>

3537 **6.1 Portable Character Set**

3538 Conforming implementations shall support one or more coded character sets. Each supported  
 3539 locale shall include the *portable character set*, which is the set of symbolic names for characters in  
 3540 Table 6-1. This is used to describe characters within the text of IEEE Std 1003.1-200x. The first  
 3541 eight entries in Table 6-1 are defined in the ISO/IEC 6429:1992 standard and the rest of the  
 3542 characters are defined in the ISO/IEC 10646-1:2000 standard.

3543 **Table 6-1** Portable Character Set

3544

3545

3546

3547

3548

3549

3550

3551

3552

3553

3554

3555

3556

3557

3558

3559

3560

3561

3562

3563

3564

3565

3566

3567

3568

3569

3570

3571

3572

3573

3574

3575

3576

Symbolic Name	Glyph	UCS	Description
<NUL>		<U0000>	NULL (NUL)
<alert>		<U0007>	BELL (BEL)
<backspace>		<U0008>	BACKSPACE (BS)
<tab>		<U0009>	CHARACTER TABULATION (HT)
<carriage-return>		<U000D>	CARRIAGE RETURN (CR)
<newline>		<U000A>	LINE FEED (LF)
<vertical-tab>		<U000B>	LINE TABULATION (VT)
<form-feed>		<U000C>	FORM FEED (FF)
<space>		<U0020>	SPACE
<exclamation-mark>	!	<U0021>	EXCLAMATION MARK
<quotation-mark>	"	<U0022>	QUOTATION MARK
<number-sign>	#	<U0023>	NUMBER SIGN
<dollar-sign>	\$	<U0024>	DOLLAR SIGN
<percent-sign>	%	<U0025>	PERCENT SIGN
<ampersand>	&	<U0026>	AMPERSAND
<apostrophe>	'	<U0027>	APOSTROPHE
<left-parenthesis>	(	<U0028>	LEFT PARENTHESIS
<right-parenthesis>	)	<U0029>	RIGHT PARENTHESIS
<asterisk>	*	<U002A>	ASTERISK
<plus-sign>	+	<U002B>	PLUS SIGN
<comma>	,	<U002C>	COMMA
<hyphen-minus>	-	<U002D>	HYPHEN-MINUS
<hyphen>	-	<U002D>	HYPHEN-MINUS
<full-stop>	.	<U002E>	FULL STOP
<period>	.	<U002E>	FULL STOP
<slash>	/	<U002F>	SOLIDUS
<solidus>	/	<U002F>	SOLIDUS
<zero>	0	<U0030>	DIGIT ZERO
<one>	1	<U0031>	DIGIT ONE
<two>	2	<U0032>	DIGIT TWO
<three>	3	<U0033>	DIGIT THREE

3577  
3578  
3579  
3580  
3581  
3582  
3583  
3584  
3585  
3586  
3587  
3588  
3589  
3590  
3591  
3592  
3593  
3594  
3595  
3596  
3597  
3598  
3599  
3600  
3601  
3602  
3603  
3604  
3605  
3606  
3607  
3608  
3609  
3610  
3611  
3612  
3613  
3614  
3615  
3616  
3617  
3618  
3619  
3620  
3621  
3622  
3623  
3624  
3625

Symbolic Name	Glyph	UCS	Description
<four>	4	<U0034>	DIGIT FOUR
<five>	5	<U0035>	DIGIT FIVE
<six>	6	<U0036>	DIGIT SIX
<seven>	7	<U0037>	DIGIT SEVEN
<eight>	8	<U0038>	DIGIT EIGHT
<nine>	9	<U0039>	DIGIT NINE
<colon>	:	<U003A>	COLON
<semicolon>	;	<U003B>	SEMICOLON
<less-than-sign>	<	<U003C>	LESS-THAN SIGN
<equals-sign>	=	<U003D>	EQUALS SIGN
<greater-than-sign>	>	<U003E>	GREATER-THAN SIGN
<question-mark>	?	<U003F>	QUESTION MARK
<commercial-at>	@	<U0040>	<U0040>
<A>	A	<U0041>	LATIN CAPITAL LETTER A
<B>	B	<U0042>	LATIN CAPITAL LETTER B
<C>	C	<U0043>	LATIN CAPITAL LETTER C
<D>	D	<U0044>	LATIN CAPITAL LETTER D
<E>	E	<U0045>	LATIN CAPITAL LETTER E
<F>	F	<U0046>	LATIN CAPITAL LETTER F
<G>	G	<U0047>	LATIN CAPITAL LETTER G
<H>	H	<U0048>	LATIN CAPITAL LETTER H
<I>	I	<U0049>	LATIN CAPITAL LETTER I
<J>	J	<U004A>	LATIN CAPITAL LETTER J
<K>	K	<U004B>	LATIN CAPITAL LETTER K
<L>	L	<U004C>	LATIN CAPITAL LETTER L
<M>	M	<U004D>	LATIN CAPITAL LETTER M
<N>	N	<U004E>	LATIN CAPITAL LETTER N
<O>	O	<U004F>	LATIN CAPITAL LETTER O
<P>	P	<U0050>	LATIN CAPITAL LETTER P
<Q>	Q	<U0051>	LATIN CAPITAL LETTER Q
<R>	R	<U0052>	LATIN CAPITAL LETTER R
<S>	S	<U0053>	LATIN CAPITAL LETTER S
<T>	T	<U0054>	LATIN CAPITAL LETTER T
<U>	U	<U0055>	LATIN CAPITAL LETTER U
<V>	V	<U0056>	LATIN CAPITAL LETTER V
<W>	W	<U0057>	LATIN CAPITAL LETTER W
<X>	X	<U0058>	LATIN CAPITAL LETTER X
<Y>	Y	<U0059>	LATIN CAPITAL LETTER Y
<Z>	Z	<U005A>	LATIN CAPITAL LETTER Z
<left-square-bracket>	[	<U005B>	LEFT SQUARE BRACKET
<backslash>	\	<U005C>	REVERSE SOLIDUS
<reverse-solidus>	\	<U005C>	REVERSE SOLIDUS
<right-square-bracket>	]	<U005D>	RIGHT SQUARE BRACKET
<circumflex-accent>	^	<U005E>	CIRCUMFLEX ACCENT
<circumflex>	^	<U005E>	CIRCUMFLEX ACCENT
<low-line>	_	<U005F>	LOW LINE
<underscore>	_	<U005F>	LOW LINE

3626

3627

3628

3629

3630

3631

3632

3633

3634

3635

3636

3637

3638

3639

3640

3641

3642

3643

3644

3645

3646

3647

3648

3649

3650

3651

3652

3653

3654

3655

3656

3657

3658

3659

3660

Symbolic Name	Glyph	UCS	Description
<grave-accent>	`	<U0060>	GRAVE ACCENT
<a>	a	<U0061>	LATIN SMALL LETTER A
<b>	b	<U0062>	LATIN SMALL LETTER B
<c>	c	<U0063>	LATIN SMALL LETTER C
<d>	d	<U0064>	LATIN SMALL LETTER D
<e>	e	<U0065>	LATIN SMALL LETTER E
<f>	f	<U0066>	LATIN SMALL LETTER F
<g>	g	<U0067>	LATIN SMALL LETTER G
<h>	h	<U0068>	LATIN SMALL LETTER H
<i>	i	<U0069>	LATIN SMALL LETTER I
<j>	j	<U006A>	LATIN SMALL LETTER J
<k>	k	<U006B>	LATIN SMALL LETTER K
<l>	l	<U006C>	LATIN SMALL LETTER L
<m>	m	<U006D>	LATIN SMALL LETTER M
<n>	n	<U006E>	LATIN SMALL LETTER N
<o>	o	<U006F>	LATIN SMALL LETTER O
<p>	p	<U0070>	LATIN SMALL LETTER P
<q>	q	<U0071>	LATIN SMALL LETTER Q
<r>	r	<U0072>	LATIN SMALL LETTER R
<s>	s	<U0073>	LATIN SMALL LETTER S
<t>	t	<U0074>	LATIN SMALL LETTER T
<u>	u	<U0075>	LATIN SMALL LETTER U
<v>	v	<U0076>	LATIN SMALL LETTER V
<w>	w	<U0077>	LATIN SMALL LETTER W
<x>	x	<U0078>	LATIN SMALL LETTER X
<y>	y	<U0079>	LATIN SMALL LETTER Y
<z>	z	<U007A>	LATIN SMALL LETTER Z
<left-brace>	{	<U007B>	LEFT CURLY BRACKET
<left-curly-bracket>	{	<U007B>	LEFT CURLY BRACKET
<vertical-line>		<U007C>	VERTICAL LINE
<right-brace>	}	<U007D>	RIGHT CURLY BRACKET
<right-curly-bracket>	}	<U007D>	RIGHT CURLY BRACKET
<tilde>	~	<U007E>	TILDE

3661

3662

3663

IEEE Std 1003.1-200x uses character names other than the above, but only in an informative way; for example, in examples to illustrate the use of characters beyond the portable character set with the facilities of IEEE Std 1003.1-200x.

3664

3665

3666

3667

3668

Table 6-1 (on page 111) defines the characters in the portable character set and the corresponding symbolic character names used to identify each character in a character set description file. The table contains more than one symbolic character name for characters whose traditional name differs from the chosen name. Characters defined in Table 6-2 (on page 116) may also be used in character set description files.

3669

3670

IEEE Std 1003.1-200x places only the following requirements on the encoded values of the characters in the portable character set:

3671

3672

3673

3674

3675

- If the encoded values associated with each member of the portable character set are not invariant across all locales supported by the implementation, if an application accesses any pair of locales where the character encodings differ, or accesses data from an application running in a locale which has different encodings from the application's current locale, the results are unspecified.

- 3676           • The encoded values associated with the digits 0 to 9 shall be such that the value of each  
3677 character after 0 shall be one greater than the value of the previous character.
- 3678           • A null character, NUL, which has all bits set to zero, shall be in the set of characters.
- 3679           • The encoded values associated with the members of the portable character set are each  
3680 represented in a single byte. Moreover, if the value is stored in an object of C-language type  
3681 **char**, it is guaranteed to be positive (except the NUL, which is always zero).
- 3682           Conforming implementations shall support certain character and character set attributes, as  
3683 defined in Section 7.2 (on page 120).

## 3684 6.2 Character Encoding

3685           The POSIX locale contains the characters in Table 6-1 (on page 111), which have the properties  
3686 listed in Section 7.3.1 (on page 122). In other locales, the presence, meaning, and representation  
3687 of any additional characters is locale-specific.

3688           In locales other than the POSIX locale, a character may have a state-dependent encoding. There  
3689 are two types of these encodings:

- 3690           • A single-shift encoding (where each character not in the initial shift state is preceded by a  
3691 shift code) can be defined if each shift-code and character sequence is considered a multi-  
3692 byte character. This is done using the concatenated-constant format in a character set  
3693 description file, as described in Section 6.4 (on page 115). If the implementation supports a  
3694 character encoding of this type, all of the standard utilities in the Shell and Utilities volume of  
3695 IEEE Std 1003.1-200x shall support it. Use of a single-shift encoding with any of the functions  
3696 in the System Interfaces volume of IEEE Std 1003.1-200x that do not specifically mention the  
3697 effects of state-dependent encoding is implementation-defined.
- 3698           • A locking-shift encoding (where the state of the character is determined by a shift code that  
3699 may affect more than the single character following it) cannot be defined with the current  
3700 character set description file format. Use of a locking-shift encoding with any of the standard  
3701 utilities in the Shell and Utilities volume of IEEE Std 1003.1-200x or with any of the functions  
3702 in the System Interfaces volume of IEEE Std 1003.1-200x that do not specifically mention the  
3703 effects of state-dependent encoding is implementation-defined.

3704           While in the initial shift state, all characters in the portable character set shall retain their usual  
3705 interpretation and shall not alter the shift state. The interpretation for subsequent bytes in the  
3706 sequence shall be a function of the current shift state. A byte with all bits zero shall be  
3707 interpreted as the null character independent of shift state. Thus a byte with all bits zero shall  
3708 never occur in the second or subsequent bytes of a character.

3709           The maximum allowable number of bytes in a character in the current locale shall be indicated  
3710 by {MB\_CUR\_MAX}, defined in the `<stdlib.h>` header and by the `<mb_cur_max>` value in a  
3711 character set description file; see Section 6.4 (on page 115). The implementation's maximum  
3712 number of bytes in a character shall be defined by the C-language macro {MB\_LEN\_MAX}.

### 3713 6.3 C Language Wide-Character Codes

3714 In the shell, the standard utilities are written so that the encodings of characters are described by  
 3715 the locale's *LC\_CTYPE* definition (see Section 7.3.1 (on page 122)) and there is no differentiation  
 3716 between characters consisting of single octets (8-bit bytes) or multiple bytes. However, in the C  
 3717 language, a differentiation is made. To ease the handling of variable length characters, the C  
 3718 language has introduced the concept of wide-character codes.

3719 All wide-character codes in a given process consist of an equal number of bits. This is in contrast  
 3720 to characters, which can consist of a variable number of bytes. The byte or byte sequence that  
 3721 represents a character can also be represented as a wide-character code. Wide-character codes  
 3722 thus provide a uniform size for manipulating text data. A wide-character code having all bits  
 3723 zero is the null wide-character code (see Section 3.246 (on page 66)), and terminates wide-  
 3724 character strings (see Section 3.432 (on page 92)). The wide-character value for each member of  
 3725 the portable character set shall equal its value when used as the lone character in an integer  
 3726 character constant. Wide-character codes for other characters are locale and implementation-  
 3727 defined. State shift bytes shall not have a wide-character code representation.

### 3728 6.4 Character Set Description File

3729 Implementations shall provide a character set description file for at least one coded character set  
 3730 supported by the implementation. These files are referred to elsewhere in IEEE Std 1003.1-200x  
 3731 as *charmap* files. It is implementation-defined whether or not users or applications can provide  
 3732 additional character set description files.

3733 IEEE Std 1003.1-200x does not require that multiple character sets or codesets be supported.  
 3734 Although multiple charmap files are supported, it is the responsibility of the implementation to  
 3735 provide the file or files; if only one is provided, only that one is accessible using the *localedef*  
 3736 utility's *-f* option.

3737 Each character set description file, except those that use the ISO/IEC 10646-1:2000 standard  
 3738 position values as the encoding values, shall define characteristics for the coded character set  
 3739 and the encoding for the characters specified in Table 6-1 (on page 111), and may define  
 3740 encoding for additional characters supported by the implementation. Other information about  
 3741 the coded character set may also be in the file. Coded character set character values shall be  
 3742 defined using symbolic character names followed by character encoding values.

3743 Each symbolic name specified in Table 6-1 (on page 111) shall be included in the file and shall be  
 3744 mapped to a unique coding value, except as noted below. The glyphs '{', '}', '\_-', '/',  
 3745 '\', '.', and '^' have more than one symbolic name; all symbolic names for each such glyph  
 3746 shall be included, each with identical encoding. If some or all of the control characters identified  
 3747 in Table 6-2 (on page 116) are supported by the implementation, the symbolic names and their  
 3748 corresponding encoding values shall be included in the file. Some of the encodings associated  
 3749 with the symbolic names in Table 6-2 (on page 116) may be the same as characters found in Table  
 3750 6-1 (on page 111); both names shall be provided for each encoding.

3751

Table 6-2 Control Character Set

3752	<ACK>	<DC2>	<ENQ>	<FS>	<IS4>	<SOH>
3753	<BEL>	<DC3>	<EOT>	<GS>	<LF>	<STX>
3754	<BS>	<DC4>	<ESC>	<HT>	<NAK>	<SUB>
3755	<CAN>	<DEL>	<ETB>	<IS1>	<RS>	<SYN>
3756	<CR>	<DLE>	<ETX>	<IS2>	<SI>	<US>
3757	<DC1>	<EM>	<FF>	<IS3>	<SO>	<VT>

3758 The following declarations can precede the character definitions. Each shall consist of the  
 3759 symbol shown in the following list, starting in column 1, including the surrounding brackets,  
 3760 followed by one or more <blank>s, followed by the value to be assigned to the symbol.

3761 <code\_set\_name> The name of the coded character set for which the character set  
 3762 description file is defined. The characters of the name shall be taken from  
 3763 the set of characters with visible glyphs defined in Table 6-1 (on page  
 3764 111).

3765 <mb\_cur\_max> The maximum number of bytes in a multi-byte character. This shall  
 3766 default to 1.

3767 <mb\_cur\_min> An unsigned positive integer value that defines the minimum number of  
 3768 XSI bytes in a character for the encoded character set. On XSI-conformant  
 3769 systems, <mb\_cur\_min> shall always be 1.

3770 <escape\_char> The character used to indicate that the characters following shall be  
 3771 interpreted in a special way, as defined later in this section. This shall  
 3772 default to backslash ('\''), which is the character used in all the following  
 3773 text and examples, unless otherwise noted.

3774 <comment\_char> The character that, when placed in column 1 of a charmap line, is used to  
 3775 indicate that the line shall be ignored. The default character shall be the  
 3776 number sign ('#').

3777 The character set mapping definitions shall be all the lines immediately following an identifier  
 3778 line containing the string "CHARMAP" starting in column 1, and preceding a trailer line  
 3779 containing the string "END CHARMAP" starting in column 1. Empty lines and lines containing a  
 3780 <comment\_char> in the first column shall be ignored. Each non-comment line of the character  
 3781 set mapping definition (that is, between the "CHARMAP" and "END CHARMAP" lines of the file)  
 3782 shall be in either of two forms:

3783 "%s %s %s\n", <symbolic-name>, <encoding>, <comments>

3784 or:

3785 "%s...%s %s %s\n", <symbolic-name>, <symbolic-name>,  
 3786 <encoding>, <comments>

3787 In the first format, the line in the character set mapping definition shall define a single symbolic  
 3788 name and a corresponding encoding. A symbolic name is one or more characters from the set  
 3789 shown with visible glyphs in Table 6-1 (on page 111), enclosed between angle brackets. A  
 3790 character following an escape character is interpreted as itself; for example, the sequence  
 3791 "<\\>" represents the symbolic name ">" enclosed between angle brackets.

3792 In the second format, the line in the character set mapping definition shall define a range of one  
 3793 or more symbolic names. In this form, the symbolic names shall consist of zero or more non-  
 3794 numeric characters from the set shown with visible glyphs in Table 6-1 (on page 111), followed  
 3795 by an integer formed by one or more decimal digits. Both integers shall contain the same number  
 3796 of digits. The characters preceding the integer shall be identical in the two symbolic names, and

3797 the integer formed by the digits in the second symbolic name shall be equal to or greater than the  
 3798 integer formed by the digits in the first name. This shall be interpreted as a series of symbolic  
 3799 names formed from the common part and each of the integers between the first and the second  
 3800 integer, inclusive. As an example, <j0101>...<j0104> is interpreted as the symbolic names  
 3801 <j0101>, <j0102>, <j0103>, and <j0104>, in that order.

3802 A character set mapping definition line shall exist for all symbolic names specified in Table 6-1  
 3803 (on page 111), and shall define the coded character value that corresponds to the character  
 3804 indicated in the table, or the coded character value that corresponds to the control character  
 3805 symbolic name. If the control characters commonly associated with the symbolic names in Table  
 3806 6-2 (on page 116) are supported by the implementation, the symbolic name and the  
 3807 corresponding encoding value shall be included in the file. Additional unique symbolic names  
 3808 may be included. A coded character value can be represented by more than one symbolic name.

3809 The encoding part is expressed as one (for single-byte character values) or more concatenated  
 3810 decimal, octal, or hexadecimal constants in the following formats:

```
3811     "%cd%u", <escape_char>, <decimal byte value>
3812     "%cx%x", <escape_char>, <hexadecimal byte value>
3813     "%co", <escape_char>, <octal byte value>
```

3814 Decimal constants shall be represented by two or three decimal digits, preceded by the escape  
 3815 character and the lowercase letter 'd'; for example, "\d05", "\d97", or "\d143".  
 3816 Hexadecimal constants shall be represented by two hexadecimal digits, preceded by the escape  
 3817 character and the lowercase letter 'x'; for example, "\x05", "\x61", or "\x8f". Octal  
 3818 constants shall be represented by two or three octal digits, preceded by the escape character; for  
 3819 example, "\05", "\141", or "\217". In a portable charmap file, each constant represents an 8-  
 3820 bit byte. When constants are concatenated for multi-byte character values, they shall be of the  
 3821 same type, and interpreted in byte order from first to last with the least significant byte of the  
 3822 multi-byte character specified by the last constant. The manner in which these constants are  
 3823 represented in the character stored in the system is implementation-defined. (This notation was  
 3824 chosen for reasons of portability. There is no requirement that the internal representation in the  
 3825 computer memory be in this same order.) Omitting bytes from a multi-byte character definition  
 3826 produces undefined results.

3827 In lines defining ranges of symbolic names, the encoded value shall be the value for the first  
 3828 symbolic name in the range (the symbolic name preceding the ellipsis). Subsequent symbolic  
 3829 names defined by the range shall have encoding values in increasing order. Bytes shall be treated  
 3830 as unsigned octets, and carry shall be propagated between the bytes as necessary to represent  
 3831 the range. For example, the line:

```
3832     <j0101>...<j0104>  \d129\d254
```

3833 is interpreted as:

```
3834     <j0101>             \d129\d254
3835     <j0102>             \d129\d255
3836     <j0103>             \d130\d0
3837     <j0104>             \d130\d1
```

3838 Note that this line is interpreted as the example even on systems with bytes larger than 8 bits.

3839 The comment is optional.

3840 The following declarations can follow the character set mapping definitions (after the "END  
 3841 CHARMAP" statement). Each shall consist of the keyword shown in the following list, starting in  
 3842 column 1, followed by the value(s) to be associated to the keyword, as defined below.

3843           **WIDTH**     An unsigned positive integer value defining the column width (see Section 3.103  
 3844                    (on page 47)) for the printable characters in the coded character set specified in  
 3845                    Table 6-1 (on page 111) and Table 6-2 (on page 116). Coded character set character  
 3846                    values shall be defined using symbolic character names followed by column width  
 3847                    values. Defining a character with more than one **WIDTH** produces undefined  
 3848                    results. The **END WIDTH** keyword shall be used to terminate the **WIDTH**  
 3849                    definitions. Specifying the width of a non-printable character in a **WIDTH**  
 3850                    declaration produces undefined results.

3851           **WIDTH\_DEFAULT**  
 3852                    An unsigned positive integer value defining the default column width for any  
 3853                    printable character not listed by one of the **WIDTH** keywords. If no  
 3854                    **WIDTH\_DEFAULT** keyword is included in the charmap, the default character  
 3855                    width shall be 1.

### 3856           **Example**

3857            After the "END CHARMAP" statement, a syntax for a width definition would be:

```
3858            WIDTH
3859            <A> 1
3860            <B> 1
3861            <C>...<Z> 1
3862            <fool>...<foon> 2
3863            END WIDTH
```

3864            In this example, the numerical code point values represented by the symbols <A> and <B> are  
 3865            assigned a width of 1. The code point values <C> to <Z> inclusive (<C>, <D>, <E>, and so on)  
 3866            are also assigned a width of 1. Using <A>...<Z> would have required fewer lines, but the  
 3867            alternative was shown to demonstrate flexibility. The keyword **WIDTH\_DEFAULT** could have  
 3868            been added as appropriate.

## 3869   **6.4.1 State-Dependent Character Encodings**

3870            This section addresses the use of state-dependent character encodings (that is, those in which the  
 3871            encoding of a character is dependent on one or more shift codes that may precede it).

3872            A single-shift encoding (where each character not in the initial shift state is preceded by a shift  
 3873            code) can be defined in the charmap format if each shift-code/character sequence is considered a  
 3874            multi-byte character, defined using the concatenated-constant format described in Section 6.4  
 3875            (on page 115). If the implementation supports a character encoding of this type, all of the  
 3876            standard utilities shall support it. A locking-shift encoding (where the state of the character is  
 3877            determined by a shift code that may affect more than the single character following it) could be  
 3878            defined with an extension to the charmap format described in Section 6.4 (on page 115). If the  
 3879            implementation supports a character encoding of this type, any of the standard utilities that  
 3880            describe character (*versus* byte) or text-file manipulation shall have the following characteristics:

- 3881            1. The utility shall process the statefully encoded data as a concatenation of state-  
 3882            independent characters. The presence of redundant locking shifts shall not affect the  
 3883            comparison of two statefully encoded strings.
- 3884            2. A utility that divides, truncates, or extracts substrings from statefully encoded data shall  
 3885            produce output that contains locking shifts at the beginning or end of the resulting data, if  
 3886            appropriate, to retain correct state information.

3888 **7.1 General**

3889 A *locale* is the definition of the subset of a user's environment that depends on language and  
 3890 cultural conventions. It is made up from one or more categories. Each category is identified by  
 3891 its name and controls specific aspects of the behavior of components of the system. Category  
 3892 names correspond to the following environment variable names:

3893 *LC\_CTYPE* Character classification and case conversion.

3894 *LC\_COLLATE* Collation order.

3895 *LC\_MONETARY* Monetary formatting.

3896 *LC\_NUMERIC* Numeric, non-monetary formatting.

3897 *LC\_TIME* Date and time formats.

3898 *LC\_MESSAGES* Formats of informative and diagnostic messages and interactive responses.

3899 The standard utilities in the Shell and Utilities volume of IEEE Std 1003.1-200x shall base their  
 3900 behavior on the current locale, as defined in the ENVIRONMENT VARIABLES section for each  
 3901 utility. The behavior of some of the C-language functions defined in the System Interfaces  
 3902 volume of IEEE Std 1003.1-200x shall also be modified based on the current locale, as defined by  
 3903 the last call to *setlocale()*.

3904 Locales other than those supplied by the implementation can be created via the *localedef* utility,  
 3905 provided that the *\_POSIX2\_LOCALEDEF* symbol is defined on the system. Even if *localedef* is not  
 3906 provided, all implementations conforming to the System Interfaces volume of  
 3907 IEEE Std 1003.1-200x shall provide one or more locales that behave as described in this chapter.  
 3908 The input to the utility is described in Section 7.3 (on page 120). The value that is used to specify  
 3909 a locale when using environment variables shall be the string specified as the *name* operand to  
 3910 the *localedef* utility when the locale was created. The strings "C" and "POSIX" are reserved as  
 3911 identifiers for the POSIX locale (see Section 7.2 (on page 120)). When the value of a locale  
 3912 environment variable begins with a slash ('/'), it shall be interpreted as the pathname of the  
 3913 locale definition; the type of file (regular, directory, and so on) used to store the locale definition  
 3914 is implementation-defined. If the value does not begin with a slash, the mechanism used to  
 3915 locate the locale is implementation-defined.

3916 If different character sets are used by the locale categories, the results achieved by an application  
 3917 utilizing these categories are undefined. Likewise, if different codesets are used for the data  
 3918 being processed by interfaces whose behavior is dependent on the current locale, or the codeset  
 3919 is different from the codeset assumed when the locale was created, the result is also undefined.

3920 Applications can select the desired locale by invoking the *setlocale()* function (or equivalent)  
 3921 with the appropriate value. If the function is invoked with an empty string, such as:

```
3922     setlocale(LC_ALL, "");
```

3923 the value of the corresponding environment variable is used. If the environment variable is  
 3924 unset or is set to the empty string, the implementation shall set the appropriate environment as  
 3925 defined in Chapter 8 (on page 157).

## 3926 7.2 POSIX Locale

3927 Conforming systems shall provide a *POSIX locale*, also known as the C locale. The behavior of  
3928 standard utilities and functions in the POSIX locale shall be as if the locale was defined via the  
3929 *localedef* utility with input data from the POSIX locale tables in Section 7.3.

3930 The tables in Section 7.3 describe the characteristics and behavior of the POSIX locale for data  
3931 consisting entirely of characters from the portable character set and the control character set. For  
3932 other characters, the behavior is unspecified. For C-language programs, the POSIX locale shall be  
3933 the default locale when the *setlocale()* function is not called.

3934 The POSIX locale can be specified by assigning to the appropriate environment variables the  
3935 values "C" or "POSIX".

3936 All implementations shall define a locale as the default locale, to be invoked when no  
3937 environment variables are set, or set to the empty string. This default locale can be the POSIX  
3938 locale or any other implementation-defined locale. Some implementations may provide facilities  
3939 for local installation administrators to set the default locale, customizing it for each location.  
3940 IEEE Std 1003.1-200x does not require such a facility.

## 3941 7.3 Locale Definition

3942 The capability to specify additional locales to those provided by an implementation is optional,  
3943 denoted by the `_POSIX2_LOCALEDEF` symbol. If the option is not supported, only  
3944 implementation-supplied locales are available. Such locales shall be documented using the  
3945 format specified in this section.

3946 Locales can be described with the file format presented in this section. The file format is that  
3947 accepted by the *localedef* utility. For the purposes of this section, the file is referred to as the *locale  
3948 definition file*, but no locales shall be affected by this file unless it is processed by *localedef* or some  
3949 similar mechanism. Any requirements in this section imposed upon the utility shall apply to  
3950 *localedef* or to any other similar utility used to install locale information using the locale  
3951 definition file format described here.

3952 The locale definition file shall contain one or more locale category source definitions, and shall  
3953 not contain more than one definition for the same locale category. If the file contains source  
3954 definitions for more than one category, implementation-defined categories, if present, shall  
3955 appear after the categories defined by Section 7.1 (on page 119). A category source definition  
3956 contains either the definition of a category or a **copy** directive. For a description of the **copy**  
3957 directive, see *localedef*. In the event that some of the information for a locale category, as  
3958 specified in this volume of IEEE Std 1003.1-200x, is missing from the locale source definition, the  
3959 behavior of that category, if it is referenced, is unspecified.

3960 A category source definition shall consist of a category header, a category body, and a category  
3961 trailer. A category header shall consist of the character string naming of the category, beginning  
3962 with the characters `LC_`. The category trailer shall consist of the string "END", followed by one  
3963 or more `<blank>`s and the string used in the corresponding category header.

3964 The category body shall consist of one or more lines of text. Each line shall contain an identifier,  
3965 optionally followed by one or more operands. Identifiers shall be either keywords, identifying a  
3966 particular locale element, or collating elements. In addition to the keywords defined in this  
3967 volume of IEEE Std 1003.1-200x, the source can contain implementation-defined keywords. Each  
3968 keyword within a locale shall have a unique name (that is, two categories cannot have a  
3969 commonly-named keyword); no keyword shall start with the characters `LC_`. Identifiers shall be  
3970 separated from the operands by one or more `<blank>`s.

3971 Operands shall be characters, collating elements, or strings of characters. Strings shall be  
 3972 enclosed in double-quotes. Literal double-quotes within strings shall be preceded by the *<escape*  
 3973 *character>*, described below. When a keyword is followed by more than one operand, the  
 3974 operands shall be separated by semicolons; *<blank>*s shall be allowed both before and after a  
 3975 semicolon.

3976 The first category header in the file can be preceded by a line modifying the comment character.  
 3977 It shall have the following format, starting in column 1:

```
3978     "comment_char %c\n", <comment character>
```

3979 The comment character shall default to the number sign ('#'). Blank lines and lines containing  
 3980 the *<comment character>* in the first position shall be ignored.

3981 The first category header in the file can be preceded by a line modifying the escape character to  
 3982 be used in the file. It shall have the following format, starting in column 1:

```
3983     "escape_char %c\n", <escape character>
```

3984 The escape character shall default to backslash, which is the character used in all examples  
 3985 shown in this volume of IEEE Std 1003.1-200x.

3986 A line can be continued by placing an escape character as the last character on the line; this  
 3987 continuation character shall be discarded from the input. Although the implementation need not  
 3988 accept any one portion of a continued line with a length exceeding {LINE\_MAX} bytes, it shall  
 3989 place no limits on the accumulated length of the continued line. Comment lines shall not be  
 3990 continued on a subsequent line using an escaped newline character.

3991 Individual characters, characters in strings, and collating elements shall be represented using  
 3992 symbolic names, as defined below. In addition, characters can be represented using the  
 3993 characters themselves or as octal, hexadecimal, or decimal constants. When non-symbolic  
 3994 notation is used, the resultant locale definitions are in many cases not portable between systems.  
 3995 The left angle bracket ('<') is a reserved symbol, denoting the start of a symbolic name; when  
 3996 used to represent itself it shall be preceded by the escape character. The following rules apply to  
 3997 character representation:

- 3998 1. A character can be represented via a symbolic name, enclosed within angle brackets '<'<br>
 3999 and '>'. The symbolic name, including the angle brackets, shall exactly match a symbolic  
 4000 name defined in the charmap file specified via the *localedef -f* option, and it shall be  
 4001 replaced by a character value determined from the value associated with the symbolic  
 4002 name in the charmap file. The use of a symbolic name not found in the charmap file shall  
 4003 constitute an error, unless the category is *LC\_CTYPE* or *LC\_COLLATE*, in which case it  
 4004 shall constitute a warning condition (see *localedef* for a description of actions resulting from  
 4005 errors and warnings). The specification of a symbolic name in a **collating-element** or  
 4006 **collating-symbol** section that duplicates a symbolic name in the charmap file (if present)  
 4007 shall be an error. Use of the escape character or a right angle bracket within a symbolic  
 4008 name is invalid unless the character is preceded by the escape character.

4009 For example:

```
4010     <c>;<c-cedilla>    "<M><a><y>"
```

- 4011 2. A character in the portable character set can be represented by the character itself, in which  
 4012 case the value of the character is implementation-defined. (Implementations may allow  
 4013 other characters to be represented as themselves, but such locale definitions are not  
 4014 portable.) Within a string, the double-quote character, the escape character, and the right  
 4015 angle bracket character shall be escaped (preceded by the escape character) to be  
 4016 interpreted as the character itself. Outside strings, the characters:

4017 , ; < > *escape\_char*

4018 shall be escaped to be interpreted as the character itself.

4019 For example:

4020 c "May"

4021 3. A character can be represented as an octal constant. An octal constant shall be specified as  
4022 the escape character followed by two or three octal digits. Each constant shall represent a  
4023 byte value. Multi-byte values can be represented by concatenated constants specified in  
4024 byte order with the last constant specifying the least significant byte of the character.

4025 For example:

4026 \143;\347;\143\150 "\115\141\171"

4027 4. A character can be represented as a hexadecimal constant. A hexadecimal constant shall be  
4028 specified as the escape character followed by an 'x' followed by two hexadecimal digits.  
4029 Each constant shall represent a byte value. Multi-byte values can be represented by  
4030 concatenated constants specified in byte order with the last constant specifying the least  
4031 significant byte of the character.

4032 For example:

4033 \x63;\xe7;\x63\x68 "\x4d\x61\x79"

4034 5. A character can be represented as a decimal constant. A decimal constant shall be specified  
4035 as the escape character followed by a 'd' followed by two or three decimal digits. Each  
4036 constant represents a byte value. Multi-byte values can be represented by concatenated  
4037 constants specified in byte order with the last constant specifying the least significant byte  
4038 of the character.

4039 For example:

4040 \d99;\d231;\d99\d104 "\d77\d97\d121"

4041 Implementations may accept single-digit octal, decimal, or hexadecimal constants following the  
4042 escape character. Only characters existing in the character set for which the locale definition is  
4043 created shall be specified, whether using symbolic names, the characters themselves, or octal,  
4044 decimal, or hexadecimal constants. If a charmap file is present, only characters defined in the  
4045 charmap can be specified using octal, decimal, or hexadecimal constants. Symbolic names not  
4046 present in the charmap file can be specified and shall be ignored, as specified under item 1  
4047 above.

### 4048 7.3.1 LC\_CTYPE

4049 The *LC\_CTYPE* category shall define character classification, case conversion, and other  
4050 character attributes. In addition, a series of characters can be represented by three adjacent  
4051 periods representing an ellipsis symbol ("..."). The ellipsis specification shall be interpreted as  
4052 meaning that all values between the values preceding and following it represent valid  
4053 characters. The ellipsis specification shall be valid only within a single encoded character set;  
4054 that is, within a group of characters of the same size. An ellipsis shall be interpreted as including  
4055 in the list all characters with an encoded value higher than the encoded value of the character  
4056 preceding the ellipsis and lower than the encoded value of the character following the ellipsis.

4057 For example:

4058 \x30;...;\x39;

4059 includes in the character class all characters with encoded values between the endpoints.

4060 The following keywords shall be recognized. In the descriptions, the term “automatically  
4061 included” means that it shall not be an error either to include or omit any of the referenced  
4062 characters; the implementation provides them if missing (even if the entire keyword is missing)  
4063 and accepts them silently if present. When the implementation automatically includes a missing  
4064 character, it shall have an encoded value dependent on the charmap file in effect (see the  
4065 description of the *localedef* **-f** option); otherwise, it shall have a value derived from an  
4066 implementation-defined character mapping.

4067 The character classes **digit**, **xdigit**, **lower**, **upper**, and **space** have a set of automatically included  
4068 characters. These only need to be specified if the character values (that is, encoding) differ from  
4069 the implementation default values. It is not possible to define a locale without these  
4070 automatically included characters unless some implementation extension is used to prevent  
4071 their inclusion. Such a definition would not be a proper superset of the C or POSIX locale and  
4072 thus, it might not be possible for conforming applications to work properly.

4073 **copy** Specify the name of an existing locale which shall be used as the definition of |  
4074 this category. If this keyword is specified, no other keyword shall be specified. |

4075 **upper** Define characters to be classified as uppercase letters.  
4076 In the POSIX locale, the 26 uppercase letters shall be included: |  
4077 A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

4078 In a locale definition file, no character specified for the keywords **cntrl**, **digit**,  
4079 **punct**, or **space** shall be specified. The uppercase letters <A> to <Z>, as  
4080 defined in Section 6.4 (on page 115) (the portable character set), are  
4081 automatically included in this class.

4082 **lower** Define characters to be classified as lowercase letters.  
4083 In the POSIX locale, the 26 lowercase letters shall be included: |  
4084 a b c d e f g h i j k l m n o p q r s t u v w x y z

4085 In a locale definition file, no character specified for the keywords **cntrl**, **digit**,  
4086 **punct**, or **space** shall be specified. The lowercase letters <a> to <z> of the  
4087 portable character set are automatically included in this class.

4088 **alpha** Define characters to be classified as letters.  
4089 In the POSIX locale, all characters in the classes **upper** and **lower** shall be |  
4090 included. |

4091 In a locale definition file, no character specified for the keywords **cntrl**, **digit**,  
4092 **punct**, or **space** shall be specified. Characters classified as either **upper** or  
4093 **lower** are automatically included in this class.

4094 **digit** Define the characters to be classified as numeric digits.  
4095 In the POSIX locale, only:  
4096 0 1 2 3 4 5 6 7 8 9  
4097 shall be included. |

4098 In a locale definition file, only the digits <zero>, <one>, <two>, <three>,  
4099 <four>, <five>, <six>, <seven>, <eight>, and <nine> shall be specified, and in  
4100 contiguous ascending sequence by numerical value. The digits <zero> to  
4101 <nine> of the portable character set are automatically included in this class.

4102 4103 4104 4105	<b>alnum</b>	Define characters to be classified as letters and numeric digits. Only the characters specified for the <b>alpha</b> and <b>digit</b> keywords shall be specified. Characters specified for the keywords <b>alpha</b> and <b>digit</b> are automatically included in this class.
4106 4107 4108	<b>space</b>	Define characters to be classified as white-space characters. In the POSIX locale, at a minimum, the <space>, <form-feed>, <newline>, <carriage-return>, <tab>, and <vertical-tab> shall be included.
4109 4110 4111 4112 4113		In a locale definition file, no character specified for the keywords <b>upper</b> , <b>lower</b> , <b>alpha</b> , <b>digit</b> , <b>graph</b> , or <b>xdigit</b> shall be specified. The <space>, <form-feed>, <newline>, <carriage-return>, <tab>, and <vertical-tab> of the portable character set, and any characters included in the class <b>blank</b> are automatically included in this class.
4114 4115	<b>cntrl</b>	Define characters to be classified as control characters. In the POSIX locale, no characters in classes <b>alpha</b> or <b>print</b> shall be included.
4116 4117		In a locale definition file, no character specified for the keywords <b>upper</b> , <b>lower</b> , <b>alpha</b> , <b>digit</b> , <b>punct</b> , <b>graph</b> , <b>print</b> , or <b>xdigit</b> shall be specified.
4118 4119 4120	<b>punct</b>	Define characters to be classified as punctuation characters. In the POSIX locale, neither the <space> nor any characters in classes <b>alpha</b> , <b>digit</b> , or <b>cntrl</b> shall be included.
4121 4122		In a locale definition file, no character specified for the keywords <b>upper</b> , <b>lower</b> , <b>alpha</b> , <b>digit</b> , <b>cntrl</b> , <b>xdigit</b> , or as the <space> shall be specified.
4123 4124	<b>graph</b>	Define characters to be classified as printable characters, not including the <space>.
4125 4126		In the POSIX locale, all characters in classes <b>alpha</b> , <b>digit</b> , and <b>punct</b> shall be included; no characters in class <b>cntrl</b> shall be included.
4127 4128 4129		In a locale definition file, characters specified for the keywords <b>upper</b> , <b>lower</b> , <b>alpha</b> , <b>digit</b> , <b>xdigit</b> , and <b>punct</b> are automatically included in this class. No character specified for the keyword <b>cntrl</b> shall be specified.
4130 4131	<b>print</b>	Define characters to be classified as printable characters, including the <space>.
4132 4133		In the POSIX locale, all characters in class <b>graph</b> shall be included; no characters in class <b>cntrl</b> shall be included.
4134 4135 4136		In a locale definition file, characters specified for the keywords <b>upper</b> , <b>lower</b> , <b>alpha</b> , <b>digit</b> , <b>xdigit</b> , <b>punct</b> , <b>graph</b> , and the <space> are automatically included in this class. No character specified for the keyword <b>cntrl</b> shall be specified.
4137 4138	<b>xdigit</b>	Define the characters to be classified as hexadecimal digits. In the POSIX locale, only:
4139		0 1 2 3 4 5 6 7 8 9 A B C D E F a b c d e f
4140		shall be included.
4141 4142 4143		In a locale definition file, only the characters defined for the class <b>digit</b> shall be specified, in contiguous ascending sequence by numerical value, followed by one or more sets of six characters representing the hexadecimal digits 10 to 15

4144		inclusive, with each set in ascending order (for example, <A>, <B>, <C>, <D>, <E>, <F>, <a>, <b>, <c>, <d>, <e>, <f>). The digits <zero> to <nine>, the uppercase letters <A> to <F>, and the lowercase letters <a> to <f> of the portable character set are automatically included in this class.
4145		
4146		
4147		
4148	<b>blank</b>	Define characters to be classified as <blank>s.
4149		In the POSIX locale, only the <space> and <tab> shall be included.
4150		In a locale definition file, the <space> and <tab> are automatically included in
4151		this class. LI <b>charclass</b> Define one or more locale-specific character class
4152		names as strings separated by semicolons. Each named character class can
4153		then be defined subsequently in the <i>LC_CTYPE</i> definition. A character class
4154		name shall consist of at least one and at most {CHARCLASS_NAME_MAX}
4155		bytes of alphanumeric characters from the portable filename character set. The
4156		first character of a character class name shall not be a digit. The name shall not
4157		match any of the <i>LC_CTYPE</i> keywords defined in this volume of
4158		IEEE Std 1003.1-200x. Future revisions of IEEE Std 1003.1-200x will not specify
4159		any <i>LC_CTYPE</i> keywords containing uppercase letters.
4160	<i>charclass-name</i>	Define characters to be classified as belonging to the named locale-specific
4161		character class. In the POSIX locale, locale-specific named character classes
4162		need not exist.
4163		If a class name is defined by a <b>charclass</b> keyword, but no characters are
4164		subsequently assigned to it, this is not an error; it represents a class without
4165		any characters belonging to it.
4166		The <i>charclass-name</i> can be used as the <i>property</i> argument to the <i>wctype()</i>
4167		function, in regular expression and shell pattern-matching bracket
4168		expressions, and by the <i>tr</i> command.
4169	<b>toupper</b>	Define the mapping of lowercase letters to uppercase letters.
4170		In the POSIX locale, at a minimum, the 26 lowercase characters:
4171		a b c d e f g h i j k l m n o p q r s t u v w x y z
4172		shall be mapped to the corresponding 26 uppercase characters:
4173		A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
4174		In a locale definition file, the operand shall consist of character pairs,
4175		separated by semicolons. The characters in each character pair shall be
4176		separated by a comma and the pair enclosed by parentheses. The first
4177		character in each pair is the lowercase letter, the second the corresponding
4178		uppercase letter. Only characters specified for the keywords <b>lower</b> and <b>upper</b>
4179		shall be specified. The lowercase letters <a> to <z>, and their corresponding
4180		uppercase letters <A> to <Z>, of the portable character set are automatically
4181		included in this mapping, but only when the <b>toupper</b> keyword is omitted
4182		from the locale definition.
4183	<b>tolower</b>	Define the mapping of uppercase letters to lowercase letters.
4184		In the POSIX locale, at a minimum, the 26 uppercase characters:
4185		A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
4186		shall be mapped to the corresponding 26 lowercase characters:

4187 a b c d e f g h i j k l m n o p q r s t u v w x y z

4188 In a locale definition file, the operand shall consist of character pairs, |  
 4189 separated by semicolons. The characters in each character pair shall be |  
 4190 separated by a comma and the pair enclosed by parentheses. The first |  
 4191 character in each pair is the uppercase letter, the second the corresponding |  
 4192 lowercase letter. Only characters specified for the keywords **lower** and **upper** |  
 4193 shall be specified. If the **tolower** keyword is omitted from the locale definition,  
 4194 the mapping is the reverse mapping of the one specified for **toupper**.

4195 The following table shows the character class combinations allowed:

4196 **Table 7-1** Valid Character Class Combinations

In Class	Can Also Belong To										
	upper	lower	alpha	digit	space	cntrl	punct	graph	print	xdigit	blank
4199 <b>upper</b>	—	—	A	x	x	x	x	A	A	—	x
4200 <b>lower</b>	—	—	A	x	x	x	x	A	A	—	x
4201 <b>alpha</b>	—	—	—	x	x	x	x	A	A	—	x
4202 <b>digit</b>	x	x	x	—	x	x	x	A	A	A	x
4203 <b>space</b>	x	x	x	x	—	—	*	*	*	x	—
4204 <b>cntrl</b>	x	x	x	x	—	—	x	x	x	x	—
4205 <b>punct</b>	x	x	x	x	—	x	—	A	A	x	—
4206 <b>graph</b>	—	—	—	—	—	x	—	—	A	—	—
4207 <b>print</b>	—	—	—	—	—	x	—	—	—	—	—
4208 <b>xdigit</b>	—	—	—	—	x	x	x	A	A	—	x
4209 <b>blank</b>	x	x	x	x	A	—	*	*	*	x	—

4210 **Notes:**

- 4211 1. Explanation of codes:  
 4212 A Automatically included; see text.  
 4213 — Permitted.  
 4214 x Mutually-exclusive.  
 4215 \* See note 2.
- 4216 2. The <space>, which is part of the **space** and **blank** classes, cannot belong to **punct** or  
 4217 **graph**, but shall automatically belong to the **print** class. Other **space** or **blank** characters  
 4218 can be classified as any of **punct**, **graph**, or **print**.

4219 **7.3.1.1 LC\_CTYPE Category in the POSIX Locale**

4220 The character classifications for the POSIX locale follow; the code listing depicts the *localedef*  
 4221 input, the table represents the same information, sorted by character.

```

4222 LC_CTYPE
4223 # The following is the POSIX locale LC_CTYPE.
4224 # "alpha" is by default "upper" and "lower"
4225 # "alnum" is by definition "alpha" and "digit"
4226 # "print" is by default "alnum", "punct" and the <space>
4227 # "graph" is by default "alnum" and "punct"
4228 #
4229 upper <A>;<B>;<C>;<D>;<E>;<F>;<G>;<H>;<I>;<J>;<K>;<L>;<M>;\
4230 <N>;<O>;<P>;<Q>;<R>;<S>;<T>;<U>;<V>;<W>;<X>;<Y>;<Z>
4231 #
    
```

```

4232 lower <a>;<b>;<c>;<d>;<e>;<f>;<g>;<h>;<i>;<j>;<k>;<l>;<m>;\
4233 <n>;<o>;<p>;<q>;<r>;<s>;<t>;<u>;<v>;<w>;<x>;<y>;<z>
4234 #
4235 digit <zero>;<one>;<two>;<three>;<four>;<five>;<six>;\
4236 <seven>;<eight>;<nine>
4237 #
4238 space <tab>;<newline>;<vertical-tab>;<form-feed>;\
4239 <carriage-return>;<space>
4240 #
4241 cntrl <alert>;<backspace>;<tab>;<newline>;<vertical-tab>;\
4242 <form-feed>;<carriage-return>;\
4243 <NUL>;<SOH>;<STX>;<ETX>;<EOT>;<ENQ>;<ACK>;<SO>;\
4244 <SI>;<DLE>;<DC1>;<DC2>;<DC3>;<DC4>;<NAK>;<SYN>;\
4245 <ETB>;<CAN>;<EM>;<SUB>;<ESC>;<IS4>;<IS3>;<IS2>;\
4246 <IS1>;<DEL>
4247 #
4248 punct <exclamation-mark>;<quotation-mark>;<number-sign>;\
4249 <dollar-sign>;<percent-sign>;<ampersand>;<apostrophe>;\
4250 <left-parenthesis>;<right-parenthesis>;<asterisk>;\
4251 <plus-sign>;<comma>;<hyphen>;<period>;<slash>;\
4252 <colon>;<semicolon>;<less-than-sign>;<equals-sign>;\
4253 <greater-than-sign>;<question-mark>;<commercial-at>;\
4254 <left-square-bracket>;<backslash>;<right-square-bracket>;\
4255 <circumflex>;<underscore>;<grave-accent>;<left-curly-bracket>;\
4256 <vertical-line>;<right-curly-bracket>;<tilde>
4257 #
4258 xdigit <zero>;<one>;<two>;<three>;<four>;<five>;<six>;<seven>;\
4259 <eight>;<nine>;<A>;<B>;<C>;<D>;<E>;<F>;<a>;<b>;<c>;<d>;<e>;<f>
4260 #
4261 blank <space>;<tab>
4262 #
4263 toupper (<a>,<A>);(<b>,<B>);(<c>,<C>);(<d>,<D>);(<e>,<E>);\
4264 (<f>,<F>);(<g>,<G>);(<h>,<H>);(<i>,<I>);(<j>,<J>);\
4265 (<k>,<K>);(<l>,<L>);(<m>,<M>);(<n>,<N>);(<o>,<O>);\
4266 (<p>,<P>);(<q>,<Q>);(<r>,<R>);(<s>,<S>);(<t>,<T>);\
4267 (<u>,<U>);(<v>,<V>);(<w>,<W>);(<x>,<X>);(<y>,<Y>);(<z>,<Z>)
4268 #
4269 tolower (<A>,<a>);(<B>,<b>);(<C>,<c>);(<D>,<d>);(<E>,<e>);\
4270 (<F>,<f>);(<G>,<g>);(<H>,<h>);(<I>,<i>);(<J>,<j>);\
4271 (<K>,<k>);(<L>,<l>);(<M>,<m>);(<N>,<n>);(<O>,<o>);\
4272 (<P>,<p>);(<Q>,<q>);(<R>,<r>);(<S>,<s>);(<T>,<t>);\
4273 (<U>,<u>);(<V>,<v>);(<W>,<w>);(<X>,<x>);(<Y>,<y>);(<Z>,<z>)
4274 END LC_CTYPE

```

4275  
4276  
4277  
4278  
4279  
4280  
4281  
4282  
4283  
4284  
4285  
4286  
4287  
4288  
4289  
4290  
4291  
4292  
4293  
4294  
4295  
4296  
4297  
4298  
4299  
4300  
4301  
4302  
4303  
4304  
4305  
4306  
4307  
4308  
4309  
4310  
4311  
4312  
4313  
4314  
4315  
4316  
4317  
4318  
4319  
4320  
4321  
4322  
4323

Symbolic Name	Other Case	Character Classes
<NUL>		cntrl
<SOH>		cntrl
<STX>		cntrl
<ETX>		cntrl
<EOT>		cntrl
<ENQ>		cntrl
<ACK>		cntrl
<alert>		cntrl
<backspace>		cntrl
<tab>		cntrl, space, blank
<newline>		cntrl, space
<vertical-tab>		cntrl, space
<form-feed>		cntrl, space
<carriage-return>		cntrl, space
<SO>		cntrl
<SI>		cntrl
<DLE>		cntrl
<DC1>		cntrl
<DC2>		cntrl
<DC3>		cntrl
<DC4>		cntrl
<NAK>		cntrl
<SYN>		cntrl
<ETB>		cntrl
<CAN>		cntrl
<EM>		cntrl
<SUB>		cntrl
<ESC>		cntrl
<IS4>		cntrl
<IS3>		cntrl
<IS2>		cntrl
<IS1>		cntrl
<space>		space, print, blank
<exclamation-mark>		punct, print, graph
<quotation-mark>		punct, print, graph
<number-sign>		punct, print, graph
<dollar-sign>		punct, print, graph
<percent-sign>		punct, print, graph
<ampersand>		punct, print, graph
<apostrophe>		punct, print, graph
<left-parenthesis>		punct, print, graph
<right-parenthesis>		punct, print, graph
<asterisk>		punct, print, graph
<plus-sign>		punct, print, graph
<comma>		punct, print, graph
<hyphen>		punct, print, graph
<period>		punct, print, graph

4324  
4325  
4326  
4327  
4328  
4329  
4330  
4331  
4332  
4333  
4334  
4335  
4336  
4337  
4338  
4339  
4340  
4341  
4342  
4343  
4344  
4345  
4346  
4347  
4348  
4349  
4350  
4351  
4352  
4353  
4354  
4355  
4356  
4357  
4358  
4359  
4360  
4361  
4362  
4363  
4364  
4365  
4366  
4367  
4368  
4369  
4370  
4371  
4372

Symbolic Name	Other Case	Character Classes
<slash>		punct, print, graph
<zero>		digit, xdigit, print, graph
<one>		digit, xdigit, print, graph
<two>		digit, xdigit, print, graph
<three>		digit, xdigit, print, graph
<four>		digit, xdigit, print, graph
<five>		digit, xdigit, print, graph
<six>		digit, xdigit, print, graph
<seven>		digit, xdigit, print, graph
<eight>		digit, xdigit, print, graph
<nine>		digit, xdigit, print, graph
<colon>		punct, print, graph
<semicolon>		punct, print, graph
<less-than-sign>		punct, print, graph
<equals-sign>		punct, print, graph
<greater-than-sign>		punct, print, graph
<question-mark>		punct, print, graph
<commercial-at>		punct, print, graph
<A>	<a>	upper, xdigit, alpha, print, graph
<B>	<b>	upper, xdigit, alpha, print, graph
<C>	<c>	upper, xdigit, alpha, print, graph
<D>	<d>	upper, xdigit, alpha, print, graph
<E>	<e>	upper, xdigit, alpha, print, graph
<F>	<f>	upper, xdigit, alpha, print, graph
<G>	<g>	upper, alpha, print, graph
<H>	<h>	upper, alpha, print, graph
<I>	<i>	upper, alpha, print, graph
<J>	<j>	upper, alpha, print, graph
<K>	<k>	upper, alpha, print, graph
<L>	<l>	upper, alpha, print, graph
<M>	<m>	upper, alpha, print, graph
<N>	<n>	upper, alpha, print, graph
<O>	<o>	upper, alpha, print, graph
<P>	<p>	upper, alpha, print, graph
<Q>	<q>	upper, alpha, print, graph
<R>	<r>	upper, alpha, print, graph
<S>	<s>	upper, alpha, print, graph
<T>	<t>	upper, alpha, print, graph
<U>	<u>	upper, alpha, print, graph
<V>	<v>	upper, alpha, print, graph
<W>	<w>	upper, alpha, print, graph
<X>	<x>	upper, alpha, print, graph
<Y>	<y>	upper, alpha, print, graph
<Z>	<z>	upper, alpha, print, graph
<left-square-bracket>		punct, print, graph
<backslash>		punct, print, graph
<right-square-bracket>		punct, print, graph

4373  
4374  
4375  
4376  
4377  
4378  
4379  
4380  
4381  
4382  
4383  
4384  
4385  
4386  
4387  
4388  
4389  
4390  
4391  
4392  
4393  
4394  
4395  
4396  
4397  
4398  
4399  
4400  
4401  
4402  
4403  
4404  
4405  
4406  
4407  
4408

Symbolic Name	Other Case	Character Classes
<circumflex>		punct, print, graph
<underscore>		punct, print, graph
<grave-accent>		punct, print, graph
<a>	<A>	lower, xdigit, alpha, print, graph
<b>	<B>	lower, xdigit, alpha, print, graph
<c>	<C>	lower, xdigit, alpha, print, graph
<d>	<D>	lower, xdigit, alpha, print, graph
<e>	<E>	lower, xdigit, alpha, print, graph
<f>	<F>	lower, xdigit, alpha, print, graph
<g>	<G>	lower, alpha, print, graph
<h>	<H>	lower, alpha, print, graph
<i>	<I>	lower, alpha, print, graph
<j>	<J>	lower, alpha, print, graph
<k>	<K>	lower, alpha, print, graph
<l>	<L>	lower, alpha, print, graph
<m>	<M>	lower, alpha, print, graph
<n>	<N>	lower, alpha, print, graph
<o>	<O>	lower, alpha, print, graph
<p>	<P>	lower, alpha, print, graph
<q>	<Q>	lower, alpha, print, graph
<r>	<R>	lower, alpha, print, graph
<s>	<S>	lower, alpha, print, graph
<t>	<T>	lower, alpha, print, graph
<u>	<U>	lower, alpha, print, graph
<v>	<V>	lower, alpha, print, graph
<w>	<W>	lower, alpha, print, graph
<x>	<X>	lower, alpha, print, graph
<y>	<Y>	lower, alpha, print, graph
<z>	<Z>	lower, alpha, print, graph
<left-curly-bracket>		punct, print, graph
<vertical-line>		punct, print, graph
<right-curly-bracket>		punct, print, graph
<tilde>		punct, print, graph
<DEL>		cntrl

4409 **7.3.2 LC\_COLLATE**

4410 The *LC\_COLLATE* category provides a collation sequence definition for numerous utilities in the  
4411 Shell and Utilities volume of IEEE Std 1003.1-200x (*sort*, *uniq*, and so on), regular expression  
4412 matching (see Chapter 9 (on page 165)) and the *strcoll()*, *strxfrm()*, *wscoll()*, and *wcsxfrm()*  
4413 functions in the System Interfaces volume of IEEE Std 1003.1-200x.

4414 A collation sequence definition shall define the relative order between collating elements  
4415 (characters and multi-character collating elements) in the locale. This order is expressed in terms  
4416 of collation values; that is, by assigning each element one or more collation values (also known  
4417 as collation weights). This does not imply that implementations shall assign such values, but  
4418 that ordering of strings using the resultant collation definition in the locale behaves as if such  
4419 assignment is done and used in the collation process. At least the following capabilities are  
4420 provided:

- 4421 1. **Multi-character collating elements.** Specification of multi-character collating elements  
4422 (that is, sequences of two or more characters to be collated as an entity).

- 4423 2. **User-defined ordering of collating elements.** Each collating element shall be assigned a  
 4424 collation value defining its order in the character (or basic) collation sequence. This  
 4425 ordering is used by regular expressions and pattern matching and, unless collation weights  
 4426 are explicitly specified, also as the collation weight to be used in sorting.
- 4427 3. **Multiple weights and equivalence classes.** Collating elements can be assigned one or  
 4428 more (up to the limit {COLL\_WEIGHTS\_MAX}, as defined in <limits.h>) collating weights  
 4429 for use in sorting. The first weight is hereafter referred to as the primary weight.
- 4430 4. **One-to-many mapping.** A single character is mapped into a string of collating elements.
- 4431 5. **Equivalence class definition.** Two or more collating elements have the same collation  
 4432 value (primary weight).
- 4433 6. **Ordering by weights.** When two strings are compared to determine their relative order,  
 4434 the two strings are first broken up into a series of collating elements; the elements in each  
 4435 successive pair of elements are then compared according to the relative primary weights  
 4436 for the elements. If equal, and more than one weight has been assigned, then the pairs of  
 4437 collating elements are recompared according to the relative subsequent weights, until  
 4438 either a pair of collating elements compare unequal or the weights are exhausted.

4439 The following keywords shall be recognized in a collation sequence definition. They are  
 4440 described in detail in the following sections.

4441	<b>copy</b>	Specify the name of an existing locale which shall be used as the
4442		definition of this category. If this keyword is specified, no other keyword
4443		shall be specified.
4444	<b>collating-element</b>	Define a collating-element symbol representing a multi-character
4445		collating element. This keyword is optional.
4446	<b>collating-symbol</b>	Define a collating symbol for use in collation order statements. This
4447		keyword is optional.
4448	<b>order_start</b>	Define collation rules. This statement shall be followed by one or more
4449		collation order statements, assigning character collation values and
4450		collation weights to collating elements.
4451	<b>order_end</b>	Specify the end of the collation-order statements.

#### 4452 7.3.2.1 *The collating-element Keyword*

4453 In addition to the collating elements in the character set, the **collating-element** keyword can be  
 4454 used to define multi-character collating elements. The syntax is as follows:

```
4455     "collating-element %s from \"%s\"\\n", <collating-symbol>, <string>
```

4456 The <collating-symbol> operand shall be a symbolic name, enclosed between angle brackets ('<' and '>'), and shall not duplicate any symbolic name in the current charmap file (if any), or any other symbolic name defined in this collation definition. The string operand is a string of two or more characters that collates as an entity. A <collating-element> defined via this keyword is only recognized with the *LC\_COLLATE* category.

4461 For example:

```
4462     collating-element <ch> from "<c><h>"
4463     collating-element <e-acute> from "<acute><e>"
4464     collating-element <ll> from "ll"
```

4465 7.3.2.2 *The collating-symbol Keyword*

4466 This keyword shall be used to define symbols for use in collation sequence statements; that is,  
4467 between the **order\_start** and the **order\_end** keywords. The syntax is as follows:

4468 `"collating-symbol %s\n", <collating-symbol>`

4469 The *<collating-symbol>* shall be a symbolic name, enclosed between angle brackets ('<' and  
4470 '>'), and shall not duplicate any symbolic name in the current charmap file (if any), or any  
4471 other symbolic name defined in this collation definition. A *<collating-symbol>* defined via this  
4472 keyword is only recognized within the *LC\_COLLATE* category.

4473 For example:

4474 `collating-symbol <UPPER_CASE>`  
4475 `collating-symbol <HIGH>`

4476 The **collating-symbol** keyword defines a symbolic name that can be associated with a relative  
4477 position in the character order sequence. While such a symbolic name does not represent any  
4478 collating element, it can be used as a weight.

4479 7.3.2.3 *The order\_start Keyword*

4480 The **order\_start** keyword shall precede collation order entries and also define the number of  
4481 weights for this collation sequence definition and other collation rules. The syntax is as follows:

4482 `"order_start %s;%s;...;%s\n", <sort-rules>, <sort-rules> ...`

4483 The operands to the **order\_start** keyword are optional. If present, the operands define rules to be  
4484 applied when strings are compared. The number of operands define how many weights each  
4485 element is assigned; if no operands are present, one **forward** operand is assumed. If present, the  
4486 first operand defines rules to be applied when comparing strings using the first (primary)  
4487 weight; the second when comparing strings using the second weight, and so on. Operands shall  
4488 be separated by semicolons (';'). Each operand shall consist of one or more collation  
4489 directives, separated by commas (','). If the number of operands exceeds the  
4490 {COLL\_WEIGHTS\_MAX} limit, the utility shall issue a warning message. The following  
4491 directives shall be supported:

4492 **forward** Specifies that comparison operations for the weight level shall proceed from start  
4493 of string towards the end of string.

4494 **backward** Specifies that comparison operations for the weight level shall proceed from end of  
4495 string towards the beginning of string.

4496 **position** Specifies that comparison operations for the weight level shall consider the relative  
4497 position of elements in the strings not subject to **IGNORE**. The string containing  
4498 an element not subject to **IGNORE** after the fewest collating elements subject to  
4499 **IGNORE** from the start of the compare shall collate first. If both strings contain a  
4500 character not subject to **IGNORE** in the same relative position, the collating values  
4501 assigned to the elements shall determine the ordering. In case of equality,  
4502 subsequent characters not subject to **IGNORE** shall be considered in the same  
4503 manner.

4504 The directives **forward** and **backward** are mutually-exclusive.

4505 If no operands are specified, a single **forward** operand shall be assumed.

4506 For example:

4507           order\_start       forward;backward

#### 4508 7.3.2.4 Collation Order

4509           The **order\_start** keyword shall be followed by collating identifier entries. The syntax for the  
4510 collating element entries is as follows:

4511           "%s %s;%s;...;%s\n", <collating-identifier>, <weight>, <weight>, ...

4512           Each *collating-identifier* shall consist of either a character (in any of the forms defined in Section  
4513 7.3 (on page 120)), a <collating-element>, a <collating-symbol>, an ellipsis, or the special symbol  
4514 **UNDEFINED**. The order in which collating elements are specified determines the character  
4515 order sequence, such that each collating element shall compare less than the elements following  
4516 it.

4517           A <collating-element> shall be used to specify multi-character collating elements, and indicates  
4518 that the character sequence specified via the <collating-element> is to be collated as a unit and in  
4519 the relative order specified by its place.

4520           A <collating-symbol> can be used to define a position in the relative order for use in weights. No  
4521 weights shall be specified with a <collating-symbol>.

4522           The ellipsis symbol specifies that a sequence of characters shall collate according to their  
4523 encoded character values. It shall be interpreted as indicating that all characters with a coded  
4524 character set value higher than the value of the character in the preceding line, and lower than  
4525 the coded character set value for the character in the following line, in the current coded  
4526 character set, shall be placed in the character collation order between the previous and the  
4527 following character in ascending order according to their coded character set values. An initial  
4528 ellipsis shall be interpreted as if the preceding line specified the NUL character, and a trailing  
4529 ellipsis as if the following line specified the highest coded character set value in the current  
4530 coded character set. An ellipsis shall be treated as invalid if the preceding or following lines do  
4531 not specify characters in the current coded character set. The use of the ellipsis symbol ties the  
4532 definition to a specific coded character set and may preclude the definition from being portable  
4533 between implementations.

4534           The symbol **UNDEFINED** shall be interpreted as including all coded character set values not  
4535 specified explicitly or via the ellipsis symbol. Such characters shall be inserted in the character  
4536 collation order at the point indicated by the symbol, and in ascending order according to their  
4537 coded character set values. If no **UNDEFINED** symbol is specified, and the current coded  
4538 character set contains characters not specified in this section, the utility shall issue a warning  
4539 message and place such characters at the end of the character collation order.

4540           The optional operands for each collation-element shall be used to define the primary, secondary,  
4541 or subsequent weights for the collating element. The first operand specifies the relative primary  
4542 weight, the second the relative secondary weight, and so on. Two or more collation-elements can  
4543 be assigned the same weight; they belong to the same *equivalence class* if they have the same  
4544 primary weight. Collation shall behave as if, for each weight level, elements subject to **IGNORE**  
4545 are removed, unless the **position** collation directive is specified for the corresponding level with  
4546 the **order\_start** keyword. Then each successive pair of elements shall be compared according to  
4547 the relative weights for the elements. If the two strings compare equal, the process shall be  
4548 repeated for the next weight level, up to the limit {COLL\_WEIGHTS\_MAX}.

4549           Weights shall be expressed as characters (in any of the forms specified in Section 7.3 (on page  
4550 120)), <collating-symbol>s, <collating-element>s, an ellipsis, or the special symbol **IGNORE**. A  
4551 single character, a <collating-symbol>, or a <collating-element> shall represent the relative position  
4552 in the character collating sequence of the character or symbol, rather than the character or  
4553 characters themselves. Thus, rather than assigning absolute values to weights, a particular

4554 weight is expressed using the relative order value assigned to a collating element based on its  
4555 order in the character collation sequence.

4556 One-to-many mapping is indicated by specifying two or more concatenated characters or  
4557 symbolic names. For example, if the <eszet> is given the string "<s><s>" as a weight,  
4558 comparisons are performed as if all occurrences of the <eszet> are replaced by "<s><s>"  
4559 (assuming that "<s>" has the collating weight "<s>"). If it is necessary to define <eszet> and  
4560 "<s><s>" as an equivalence class, then a collating element must be defined for the string "ss".

4561 All characters specified via an ellipsis shall by default be assigned unique weights, equal to the  
4562 relative order of characters. Characters specified via an explicit or implicit **UNDEFINED** special  
4563 symbol shall by default be assigned the same primary weight (that is, they belong to the same  
4564 equivalence class). An ellipsis symbol as a weight shall be interpreted to mean that each  
4565 character in the sequence shall have unique weights, equal to the relative order of their character  
4566 in the character collation sequence. The use of the ellipsis as a weight shall be treated as an error  
4567 if the collating element is neither an ellipsis nor the special symbol **UNDEFINED**.

4568 The special keyword **IGNORE** as a weight shall indicate that when strings are compared using  
4569 the weights at the level where **IGNORE** is specified, the collating element shall be ignored; that  
4570 is, as if the string did not contain the collating element. In regular expressions and pattern  
4571 matching, all characters that are subject to **IGNORE** in their primary weight form an  
4572 equivalence class.

4573 An empty operand shall be interpreted as the collating element itself.

4574 For example, the order statement:

```
4575     <a>     <a> ; <a>
```

4576 is equal to:

```
4577     <a>
```

4578 An ellipsis can be used as an operand if the collating element was an ellipsis, and shall be  
4579 interpreted as the value of each character defined by the ellipsis.

4580 The collation order as defined in this section affects the interpretation of bracket expressions in  
4581 regular expressions (see Section 9.3.5 (on page 168)).

4582 For example:

```
4583     order_start  forward ; backward
4584     UNDEFINED   IGNORE ; IGNORE
4585     <LOW>
4586     <space>     <LOW> ; <space>
4587     ...         <LOW> ; ...
4588     <a>         <a> ; <a>
4589     <a-acute>   <a> ; <a-acute>
4590     <a-grave>   <a> ; <a-grave>
4591     <A>        <a> ; <A>
4592     <A-acute>  <a> ; <A-acute>
4593     <A-grave>  <a> ; <A-grave>
4594     <ch>       <ch> ; <ch>
4595     <Ch>       <ch> ; <Ch>
4596     <s>        <s> ; <s>
4597     <eszet>    "<s><s>" ; "<eszet><eszet>"
4598     order_end
```

4599 This example is interpreted as follows:

- 4600 1. The **UNDEFINED** means that all characters not specified in this definition (explicitly or  
4601 via the ellipsis) shall be ignored for collation purposes.
- 4602 2. All characters between <space> and 'a' shall have the same primary equivalence class  
4603 and individual secondary weights based on their ordinal encoded values.
- 4604 3. All characters based on the uppercase or lowercase character 'a' belong to the same  
4605 primary equivalence class.
- 4606 4. The multi-character collating element <ch> is represented by the collating symbol <ch>  
4607 and belongs to the same primary equivalence class as the multi-character collating element  
4608 <Ch>.

#### 4609 7.3.2.5 *The order\_end Keyword*

4610 The collating order entries shall be terminated with an **order\_end** keyword.

#### 4611 7.3.2.6 *LC\_COLLATE Category in the POSIX Locale*

4612 The collation sequence definition of the POSIX locale follows; the code listing depicts the  
4613 *localedef* input.

```
4614 LC_COLLATE
4615 # This is the POSIX locale definition for the LC_COLLATE category.
4616 # The order is the same as in the ASCII codeset.
4617 order_start forward
4618 <NUL>
4619 <SOH>
4620 <STX>
4621 <ETX>
4622 <EOT>
4623 <ENQ>
4624 <ACK>
4625 <alert>
4626 <backspace>
4627 <tab>
4628 <newline>
4629 <vertical-tab>
4630 <form-feed>
4631 <carriage-return>
4632 <SO>
4633 <SI>
4634 <DLE>
4635 <DC1>
4636 <DC2>
4637 <DC3>
4638 <DC4>
4639 <NAK>
4640 <SYN>
4641 <ETB>
4642 <CAN>
4643 <EM>
4644 <SUB>
4645 <ESC>
```

4646	<IS4>
4647	<IS3>
4648	<IS2>
4649	<IS1>
4650	<space>
4651	<exclamation-mark>
4652	<quotation-mark>
4653	<number-sign>
4654	<dollar-sign>
4655	<percent-sign>
4656	<ampersand>
4657	<apostrophe>
4658	<left-parenthesis>
4659	<right-parenthesis>
4660	<asterisk>
4661	<plus-sign>
4662	<comma>
4663	<hyphen>
4664	<period>
4665	<slash>
4666	<zero>
4667	<one>
4668	<two>
4669	<three>
4670	<four>
4671	<five>
4672	<six>
4673	<seven>
4674	<eight>
4675	<nine>
4676	<colon>
4677	<semicolon>
4678	<less-than-sign>
4679	<equals-sign>
4680	<greater-than-sign>
4681	<question-mark>
4682	<commercial-at>
4683	<A>
4684	<B>
4685	<C>
4686	<D>
4687	<E>
4688	<F>
4689	<G>
4690	<H>
4691	<I>
4692	<J>
4693	<K>
4694	<L>
4695	<M>
4696	<N>
4697	<O>

```
4698      <P>
4699      <Q>
4700      <R>
4701      <S>
4702      <T>
4703      <U>
4704      <V>
4705      <W>
4706      <X>
4707      <Y>
4708      <Z>
4709      <left-square-bracket>
4710      <backslash>
4711      <right-square-bracket>
4712      <circumflex>
4713      <underscore>
4714      <grave-accent>
4715      <a>
4716      <b>
4717      <c>
4718      <d>
4719      <e>
4720      <f>
4721      <g>
4722      <h>
4723      <i>
4724      <j>
4725      <k>
4726      <l>
4727      <m>
4728      <n>
4729      <o>
4730      <p>
4731      <q>
4732      <r>
4733      <s>
4734      <t>
4735      <u>
4736      <v>
4737      <w>
4738      <x>
4739      <y>
4740      <z>
4741      <left-curly-bracket>
4742      <vertical-line>
4743      <right-curly-bracket>
4744      <tilde>
4745      <DEL>
4746      order_end
4747      #
4748      END LC_COLLATE
```

4749 **7.3.3 LC\_MONETARY**

4750 The *LC\_MONETARY* category shall define the rules and symbols that are used to format  
 4751 XSI monetary numeric information. This information is available through the *localeconv()* function  
 4752 and is used by the *strfmon()* function.

4753 XSI Some of the information is also available in an alternative form via the *nl\_langinfo()* function  
 4754 (see CRNCYSTR in <**langinfo.h**>).

4755 The following items are defined in this category of the locale. The item names are the keywords  
 4756 recognized by the *localedef* utility when defining a locale. They are also similar to the member  
 4757 names of the **lconv** structure defined in <**locale.h**>; see <**locale.h**> for the exact symbols in the  
 4758 header. The *localeconv()* function returns {CHAR\_MAX} for unspecified integer items and the  
 4759 empty string (" ") for unspecified or size zero string items.

4760 In a locale definition file, the operands are strings, formatted as indicated by the grammar in  
 4761 Section 7.4 (on page 149). For some keywords, the strings can contain only integers. Keywords  
 4762 that are not provided, string values set to the empty string (" "), or integer keywords set to -1,  
 4763 are used to indicate that the value is not available in the locale. The following keywords shall be  
 4764 recognized:

4765 **copy** Specify the name of an existing locale which shall be used as the |  
 4766 definition of this category. If this keyword is specified, no other keyword |  
 4767 shall be specified. |

4768 **Note:** This is a *localedef* utility keyword, unavailable through *localeconv()*.

4769 **int\_curr\_symbol** The international currency symbol. The operand shall be a four-character  
 4770 string, with the first three characters containing the alphabetic  
 4771 international currency symbol in accordance with those specified in the  
 4772 ISO 4217: 1995 standard. The fourth character shall be the character used  
 4773 to separate the international currency symbol from the monetary  
 4774 quantity.

4775 **currency\_symbol** The string that shall be used as the local currency symbol.

4776 **mon\_decimal\_point** The operand is a string containing the symbol that shall be used as the  
 4777 decimal delimiter (radix character) in monetary formatted quantities. |

4778 **mon\_thousands\_sep** The operand is a string containing the symbol that shall be used as a  
 4779 separator for groups of digits to the left of the decimal delimiter in |  
 4780 formatted monetary quantities. |

4781 **mon\_grouping** Define the size of each group of digits in formatted monetary quantities.  
 4782 The operand is a sequence of integers separated by semicolons. Each  
 4783 integer specifies the number of digits in each group, with the initial  
 4784 integer defining the size of the group immediately preceding the decimal  
 4785 delimiter, and the following integers defining the preceding groups. If the  
 4786 last integer is not -1, then the size of the previous group (if any) shall be  
 4787 repeatedly used for the remainder of the digits. If the last integer is -1,  
 4788 then no further grouping shall be performed.

4789 **positive\_sign** A string that shall be used to indicate a non-negative-valued formatted  
 4790 monetary quantity.

4791 **negative\_sign** A string that shall be used to indicate a negative-valued formatted  
 4792 monetary quantity.

4793 **int\_frac\_digits** An integer representing the number of fractional digits (those to the right  
 4794 of the decimal delimiter) to be written in a formatted monetary quantity

4795		using <b>int_curr_symbol</b> .
4796	<b>frac_digits</b>	An integer representing the number of fractional digits (those to the right of the decimal delimiter) to be written in a formatted monetary quantity using <b>currency_symbol</b> .
4797		
4798		
4799	<b>p_cs_precedes</b>	An integer set to 1 if the <b>currency_symbol</b> precedes the value for a monetary quantity with a non-negative value, and set to 0 if the symbol succeeds the value.
4800		
4801		
4802	<b>p_sep_by_space</b>	An integer set to 0 if no space separates the <b>currency_symbol</b> from the value for a monetary quantity with a non-negative value, set to 1 if a space separates the symbol from the value, and set to 2 if a space separates the symbol and the sign string, if adjacent.
4803		
4804		
4805		
4806	<b>n_cs_precedes</b>	An integer set to 1 if the <b>currency_symbol</b> precedes the value for a monetary quantity with a negative value, and set to 0 if the symbol succeeds the value.
4807		
4808		
4809	<b>n_sep_by_space</b>	An integer set to 0 if no space separates the <b>currency_symbol</b> from the value for a monetary quantity with a negative value, set to 1 if a space separates the symbol from the value, and set to 2 if a space separates the symbol and the sign string, if adjacent.
4810		
4811		
4812		
4813	<b>p_sign_posn</b>	An integer set to a value indicating the positioning of the <b>positive_sign</b> for a monetary quantity with a non-negative value. The following integer values shall be recognized for <b>int_n_sign_posn</b> , <b>int_p_sign_posn</b> , <b>n_sign_posn</b> , and <b>p_sign_posn</b> :
4814		
4815		
4816		
4817		0    Parentheses enclose the quantity and the <b>currency_symbol</b> .
4818		1    The sign string precedes the quantity and the <b>currency_symbol</b> .
4819		2    The sign string succeeds the quantity and the <b>currency_symbol</b> .
4820		3    The sign string precedes the <b>currency_symbol</b> .
4821		4    The sign string succeeds the <b>currency_symbol</b> .
4822	<b>n_sign_posn</b>	An integer set to a value indicating the positioning of the <b>negative_sign</b> for a negative formatted monetary quantity.
4823		
4824	<b>int_p_cs_precedes</b>	An integer set to 1 if the <b>int_curr_symbol</b> precedes the value for a monetary quantity with a non-negative value, and set to 0 if the symbol succeeds the value.
4825		
4826		
4827	<b>int_n_cs_precedes</b>	An integer set to 1 if the <b>int_curr_symbol</b> precedes the value for a monetary quantity with a negative value, and set to 0 if the symbol succeeds the value.
4828		
4829		
4830	<b>int_p_sep_by_space</b>	An integer to set 0 if no space separates the <b>int_curr_symbol</b> from the value for a monetary quantity with a non-negative value, set to 1 if a space separates the symbol from the value, and set to 2 if a space separates the symbol and the sign string, if adjacent.
4831		
4832		
4833		
4834	<b>int_n_sep_by_space</b>	An integer set to 0 if no space separates the <b>int_curr_symbol</b> from the value for a monetary quantity with a negative value, set to 1 if a space separates the symbol from the value, and set to 2 if a space separates the symbol and the sign string, if adjacent.
4835		
4836		
4837		



4886 **7.3.4 LC\_NUMERIC**

4887 The *LC\_NUMERIC* category shall define the rules and symbols that are used to format non-  
 4888 XSI monetary numeric information. This information is available through the *localeconv()* function.  
 4889 Some of the information is also available in an alternative form via the *nl\_langinfo()* function.

4890 The following items are defined in this category of the locale. The item names are the keywords  
 4891 recognized by the *localedef* utility when defining a locale. They are also similar to the member  
 4892 names of the **lconv** structure defined in `<locale.h>`; see `<locale.h>` for the exact symbols in the  
 4893 header. The *localeconv()* function returns `{CHAR_MAX}` for unspecified integer items and the  
 4894 empty string (" ") for unspecified or size zero string items.

4895 In a locale definition file, the operands are strings, formatted as indicated by the grammar in  
 4896 Section 7.4 (on page 149). For some keywords, the strings can only contain integers. Keywords  
 4897 that are not provided, string values set to the empty string (" "), or integer keywords set to `-1`,  
 4898 shall be used to indicate that the value is not available in the locale. The following keywords  
 4899 shall be recognized:

4900 **copy** Specify the name of an existing locale which shall be used as the definition of |  
 4901 this category. If this keyword is specified, no other keyword shall be specified. |

4902 **Note:** This is a *localedef* utility keyword, unavailable through *localeconv()*.

4903 **decimal\_point** The operand is a string containing the symbol that shall be used as the  
 4904 decimal delimiter (radix character) in numeric, non-monetary formatted  
 4905 quantities. This keyword cannot be omitted and cannot be set to the empty  
 4906 string. In contexts where standards limit the **decimal\_point** to a single byte,  
 4907 the result of specifying a multi-byte operand shall be unspecified.

4908 **thousands\_sep** The operand is a string containing the symbol that shall be used as a separator  
 4909 for groups of digits to the left of the decimal delimiter in numeric, non-  
 4910 monetary formatted monetary quantities. In contexts where standards limit  
 4911 the **thousands\_sep** to a single byte, the result of specifying a multi-byte  
 4912 operand shall be unspecified.

4913 **grouping** Define the size of each group of digits in formatted non-monetary quantities.  
 4914 The operand is a sequence of integers separated by semicolons. Each integer  
 4915 specifies the number of digits in each group, with the initial integer defining  
 4916 the size of the group immediately preceding the decimal delimiter, and the  
 4917 following integers defining the preceding groups. If the last integer is not `-1`,  
 4918 then the size of the previous group (if any) shall be repeatedly used for the  
 4919 remainder of the digits. If the last integer is `-1`, then no further grouping shall  
 4920 be performed.

4921 **7.3.4.1 LC\_NUMERIC Category in the POSIX Locale**

4922 The non-monetary numeric formatting definitions for the POSIX locale follow; the code listing  
 4923 depicting the *localedef* input, the table representing the same information with the addition of  
 4924 XSI *localeconv()* values, and *nl\_langinfo()* constants.

```

4925 LC_NUMERIC
4926 # This is the POSIX locale definition for
4927 # the LC_NUMERIC category.
4928 #
4929 decimal_point    "<period>"
4930 thousands_sep    " "
4931 grouping         -1
4932 #
  
```

4933 END LC\_NUMERIC

Item	langinfo Constant	POSIX Locale Value	localeconv() Value	localedef Value
decimal_point	RADIXCHAR	."	."	.
thousands_sep	THOUSEP	N/A	."	."
grouping	—	N/A	."	-1

4939 XSI In the preceding table, the **langinfo Constant** column represents an XSI-conforming extension.  
 4940 The entry N/A indicates that the value is not available in the POSIX locale.

4941 **7.3.5 LC\_TIME**

4942 The *LC\_TIME* category shall define the interpretation of the conversion specifications supported  
 4943 XSI by the *date* utility and shall affect the behavior of the *strftime()*, *wcsftime()*, *strptime()*, and  
 4944 *nl\_langinfo()* functions. Since the interfaces for C-language access and locale definition differ  
 4945 significantly, they are described separately.

4946 **7.3.5.1 LC\_TIME Locale Definition**

4947 For locale definition, the following mandatory keywords shall be recognized:

4948 **copy** Specify the name of an existing locale which shall be used as the definition of  
 4949 this category. If this keyword is specified, no other keyword shall be specified.

4950 **abday** Define the abbreviated weekday names, corresponding to the %a conversion  
 4951 specification (conversion specification in the *strptime()*, *wcsftime()*, and  
 4952 *strptime()* functions). The operand shall consist of seven semicolon-separated  
 4953 strings, each surrounded by double-quotes. The first string shall be the  
 4954 abbreviated name of the day corresponding to Sunday, the second the  
 4955 abbreviated name of the day corresponding to Monday, and so on.

4956 **day** Define the full weekday names, corresponding to the %A conversion  
 4957 specification. The operand shall consist of seven semicolon-separated strings,  
 4958 each surrounded by double-quotes. The first string is the full name of the day  
 4959 corresponding to Sunday, the second the full name of the day corresponding  
 4960 to Monday, and so on.

4961 **abmon** Define the abbreviated month names, corresponding to the %b conversion  
 4962 specification. The operand shall consist of twelve semicolon-separated strings,  
 4963 each surrounded by double-quotes. The first string shall be the abbreviated  
 4964 name of the first month of the year (January), the second the abbreviated  
 4965 name of the second month, and so on.

4966 **mon** Define the full month names, corresponding to the %B conversion  
 4967 specification. The operand shall consist of twelve semicolon-separated strings,  
 4968 each surrounded by double-quotes. The first string shall be the full name of  
 4969 the first month of the year (January), the second the full name of the second  
 4970 month, and so on.

4971 **d\_t\_fmt** Define the appropriate date and time representation, corresponding to the %c  
 4972 conversion specification. The operand shall consist of a string containing any  
 4973 combination of characters and conversion specifications. In addition, the  
 4974 string can contain escape sequences defined in the table in Table 5-1 (on page  
 4975 108) ('\\', '\a', '\b', '\f', '\n', '\r', '\t', '\v').

4976	<b>d_fmt</b>	Define the appropriate date representation, corresponding to the %X conversion specification. The operand shall consist of a string containing any combination of characters and conversion specifications. In addition, the string can contain escape sequences defined in the table in Table 5-1 (on page 108).
4977		
4978		
4979		
4980		
4981	<b>t_fmt</b>	Define the appropriate time representation, corresponding to the %X conversion specification. The operand shall consist of a string containing any combination of characters and conversion specifications. In addition, the string can contain escape sequences defined in the table in Table 5-1 (on page 108).
4982		
4983		
4984		
4985		
4986	<b>am_pm</b>	Define the appropriate representation of the <i>ante meridiem</i> and <i>post meridiem</i> strings, corresponding to the %p conversion specification. The operand shall consist of two strings, separated by a semicolon, each surrounded by double-quotes. The first string shall represent the <i>ante meridiem</i> designation, the last string the <i>post meridiem</i> designation.
4987		
4988		
4989		
4990		
4991	<b>t_fmt_ampm</b>	Define the appropriate time representation in the 12-hour clock format with <b>am_pm</b> , corresponding to the %r conversion specification. The operand shall consist of a string and can contain any combination of characters and conversion specifications. If the string is empty, the 12-hour format is not supported in the locale.
4992		
4993		
4994		
4995		
4996	<b>era</b>	Define how years are counted and displayed for each era in a locale. The operand shall consist of semicolon-separated strings. Each string shall be an era description segment with the format:  <i>direction:offset:start_date:end_date:era_name:era_format</i>  according to the definitions below. There can be as many era description segments as are necessary to describe the different eras.
4997		
4998		
4999		
5000		
5001		
5002	<b>Note:</b>	The start of an era might not be the earliest point in the era—it may be the latest. For example, the Christian era BC starts on the day before January 1, AD 1, and increases with earlier time.
5003		
5004		
5005	<i>direction</i>	Either a '+' or a '-' character. The '+' character shall indicate that years closer to the <i>start_date</i> have lower numbers than those closer to the <i>end_date</i> . The '-' character shall indicate that years closer to the <i>start_date</i> have higher numbers than those closer to the <i>end_date</i> .
5006		
5007		
5008		
5009		
5010	<i>offset</i>	The number of the year closest to the <i>start_date</i> in the era, corresponding to the %EY conversion specification.
5011		
5012	<i>start_date</i>	A date in the form yyyy/mm/dd, where yyyy, mm, and dd are the year, month, and day numbers respectively of the start of the era. Years prior to AD 1 shall be represented as negative numbers.
5013		
5014		
5015		
5016	<i>end_date</i>	The ending date of the era, in the same format as the <i>start_date</i> , or one of the two special values "-*" or "+*". The value "-*" shall indicate that the ending date is the beginning of time. The value "+*" shall indicate that the ending date is the end of time.
5017		
5018		
5019		
5020	<i>era_name</i>	A string representing the name of the era, corresponding to the %EC conversion specification.
5021		

5022		<i>era_format</i>	A string for formatting the year in the era, corresponding to the
5023			%EY conversion specification.
5024	<b>era_d_fmt</b>		Define the format of the date in alternative era notation, corresponding to the
5025			%Ex conversion specification.
5026	<b>era_t_fmt</b>		Define the locale's appropriate alternative time format, corresponding to the
5027			%EX conversion specification.
5028	<b>era_d_t_fmt</b>		Define the locale's appropriate alternative date and time format,
5029			corresponding to the %Ec conversion specification.
5030	<b>alt_digits</b>		Define alternative symbols for digits, corresponding to the %O modified
5031			conversion specification. The operand shall consist of semicolon-separated
5032			strings, each surrounded by double-quotes. The first string shall be the
5033			alternative symbol corresponding with zero, the second string the symbol
5034			corresponding with one, and so on. Up to 100 alternative symbol strings can
5035			be specified. The %O modifier shall indicate that the string corresponding to
5036			the value specified via the conversion specification shall be used instead of the
5037			value.
5038	<b>7.3.5.2</b>	<i>LC_TIME C-Language Access</i>	
5039	XSI		This section describes extensions to access information in the <i>LC_TIME</i> category using the
5040			<i>nl_langinfo()</i> function. This functionality is dependent on support of the XSI extension (and the
5041			rest of this section is not further shaded for this option).
5042			The following constants used to identify items of <i>langinfo</i> data can be used as arguments to the
5043			<i>nl_langinfo()</i> function to access information in the <i>LC_TIME</i> category. These constants are
5044			defined in the <b>&lt;langinfo.h&gt;</b> header.
5045	<b>ABDAY_x</b>		The abbreviated weekday names (for example Sun), where <i>x</i> is a number from
5046			1 to 7.
5047	<b>DAY_x</b>		The full weekday names (for example Sunday), where <i>x</i> is a number from 1 to
5048			7.
5049	<b>ABMON_x</b>		The abbreviated month names (for example Jan), where <i>x</i> is a number from 1
5050			to 12.
5051	<b>MON_x</b>		The full month names (for example January), where <i>x</i> is a number from 1 to
5052			12.
5053	<b>D_T_FMT</b>		The appropriate date and time representation.
5054	<b>D_FMT</b>		The appropriate date representation.
5055	<b>T_FMT</b>		The appropriate time representation.
5056	<b>AM_STR</b>		The appropriate ante-meridiem affix.
5057	<b>PM_STR</b>		The appropriate post-meridiem affix.
5058	<b>T_FMT_AMPM</b>		The appropriate time representation in the 12-hour clock format with
5059			AM_STR and PM_STR.
5060	<b>ERA</b>		The era description segments, which describe how years are counted and
5061			displayed for each era in a locale. Each era description segment shall have the
5062			format:

```

5063         direction:offset:start_date:end_date:era_name:era_format
5064         according to the definitions below. There can be as many era description
5065         segments as are necessary to describe the different eras. Era description
5066         segments are separated by semicolons.
5067         direction      Either a '+' or a '-' character. The '+' character shall indicate
5068         that years closer to the start_date have lower numbers than those
5069         closer to the end_date. The '-' character shall indicate that
5070         years closer to the start_date have higher numbers than those
5071         closer to the end_date.
5072         offset         The number of the year closest to the start_date in the era.
5073         start_date      A date in the form yyyy/mm/dd, where yyyy, mm, and dd are the
5074         year, month, and day numbers respectively of the start of the
5075         era. Years prior to AD 1 shall be represented as negative
5076         numbers.
5077         end_date        The ending date of the era, in the same format as the start_date,
5078         or one of the two special values "-*" or "+*". The value "-*"
5079         shall indicate that the ending date is the beginning of time. The
5080         value "+*" shall indicate that the ending date is the end of time.
5081         era_name         The era, corresponding to the %EC conversion specification.
5082         era_format      The format of the year in the era, corresponding to the %EY
5083         conversion specification.
5084     ERA_D_FMT      The era date format.
5085     ERA_T_FMT      The locale's appropriate alternative time format, corresponding to the %EX
5086     conversion specification.
5087     ERA_D_T_FMT    The locale's appropriate alternative date and time format, corresponding to
5088     the %Ec conversion specification.
5089     ALT_DIGITS     The alternative symbols for digits, corresponding to the %O conversion
5090     specification modifier. The value consists of semicolon-separated symbols.
5091     The first is the alternative symbol corresponding to zero, the second is the
5092     symbol corresponding to one, and so on. Up to 100 alternative symbols may
5093     be specified.
5094 7.3.5.3 LC_TIME Category in the POSIX Locale
5095     The LC_TIME category definition of the POSIX locale follows; the code listing depicts the
5096     localedef input; the table represents the same information with the addition of localedef keywords,
5097     conversion specifiers used by the date utility and the strptime(), wcsftime(), and strptime()
5098     XSI functions, and nl_langinfo() constants.
5099     LC_TIME
5100     # This is the POSIX locale definition for
5101     # the LC_TIME category.
5102     #
5103     # Abbreviated weekday names (%a)
5104     abday      "<S><u><n>" ; "<M><o><n>" ; "<T><u><e>" ; "<W><e><d>" ; \
5105               "<T><h><u>" ; "<F><r><i>" ; "<S><a><t>"
5106     #
5107     # Full weekday names (%A)

```

```

5108     day          "<S><u><n><d><a><y>" ; "<M><o><n><d><a><y>" ; \
5109             "<T><u><e><s><d><a><y>" ; "<W><e><d><n><e><s><d><a><y>" ; \
5110             "<T><h><u><r><s><d><a><y>" ; "<F><r><i><d><a><y>" ; \
5111             "<S><a><t><u><r><d><a><y>"
5112     #
5113     # Abbreviated month names (%b)
5114     abmon         "<J><a><n>" ; "<F><e><b>" ; "<M><a><r>" ; \
5115             "<A><p><r>" ; "<M><a><y>" ; "<J><u><n>" ; \
5116             "<J><u><l>" ; "<A><u><g>" ; "<S><e><p>" ; \
5117             "<O><c><t>" ; "<N><o><v>" ; "<D><e><c>"
5118     #
5119     # Full month names (%B)
5120     mon           "<J><a><n><u><a><r><y>" ; "<F><e><b><r><u><a><r><y>" ; \
5121             "<M><a><r><c><h>" ; "<A><p><r><i><l>" ; \
5122             "<M><a><y>" ; "<J><u><n><e>" ; \
5123             "<J><u><l><y>" ; "<A><u><g><u><s><t>" ; \
5124             "<S><e><p><t><e><m><b><e><r>" ; "<O><c><t><o><b><e><r>" ; \
5125             "<N><o><v><e><m><b><e><r>" ; "<D><e><c><e><m><b><e><r>"
5126     #
5127     # Equivalent of AM/PM (%p)          "AM" ; "PM"
5128     am_pm         "<A><M>" ; "<P><M>"
5129     #
5130     # Appropriate date and time representation (%c)
5131     # "%a %b %e %H:%M:%S %Y"
5132     d_t_fmt       "<percent-sign><a><space><percent-sign><b>\
5133             <space><percent-sign><e><space><percent-sign><H>\
5134             <colon><percent-sign><M><colon><percent-sign><S>\
5135             <space><percent-sign><Y>"
5136     #
5137     # Appropriate date representation (%x) "%m/%d/%y"
5138     d_fmt         "<percent-sign><m><slash><percent-sign><d>\
5139             <slash><percent-sign><y>"
5140     #
5141     # Appropriate time representation (%X) "%H:%M:%S"
5142     t_fmt         "<percent-sign><H><colon><percent-sign><M>\
5143             <colon><percent-sign><S>"
5144     #
5145     # Appropriate 12-hour time representation (%r) "%I:%M:%S %p"
5146     t_fmt_ampm   "<percent-sign><I><colon><percent-sign><M><colon>\
5147             <percent-sign><S><space><percent_sign><p>"
5148     #
5149     END LC_TIME

```

5150  
5151  
5152  
5153  
5154  
5155

localedef Keyword	langinfo Constant	Conversion Specification	POSIX Locale Value
d_t_fmt	D_T_FMT	%c	"%a %b %e %H:%M:%S %Y"
d_fmt	D_FMT	%x	"%m/%d/%y"
t_fmt	T_FMT	%X	"%H:%M:%S"

5156  
5157  
5158  
5159  
5160  
5161  
5162  
5163  
5164  
5165  
5166  
5167  
5168  
5169  
5170  
5171  
5172  
5173  
5174  
5175  
5176  
5177  
5178  
5179  
5180  
5181  
5182  
5183  
5184  
5185  
5186  
5187  
5188  
5189  
5190  
5191  
5192  
5193  
5194  
5195  
5196  
5197  
5198  
5199  
5200  
5201  
5202  
5203  
5204

localedef Keyword	langinfo Constant	Conversion Specification	POSIX Locale Value
am_pm	AM_STR	%p	"AM"
am_pm	PM_STR	%p	"PM"
t_fmt_ampm	T_FMT_AMPMPM	%r	"%I:%M:%S %p"
day	DAY_1	%A	"Sunday"
day	DAY_2	%A	"Monday"
day	DAY_3	%A	"Tuesday"
day	DAY_4	%A	"Wednesday"
day	DAY_5	%A	"Thursday"
day	DAY_6	%A	"Friday"
day	DAY_7	%A	"Saturday"
abday	ABDAY_1	%a	"Sun"
abday	ABDAY_2	%a	"Mon"
abday	ABDAY_3	%a	"Tue"
abday	ABDAY_4	%a	"Wed"
abday	ABDAY_5	%a	"Thu"
abday	ABDAY_6	%a	"Fri"
abday	ABDAY_7	%a	"Sat"
mon	MON_1	%B	"January"
mon	MON_2	%B	"February"
mon	MON_3	%B	"March"
mon	MON_4	%B	"April"
mon	MON_5	%B	"May"
mon	MON_6	%B	"June"
mon	MON_7	%B	"July"
mon	MON_8	%B	"August"
mon	MON_9	%B	"September"
mon	MON_10	%B	"October"
mon	MON_11	%B	"November"
mon	MON_12	%B	"December"
abmon	ABMON_1	%b	"Jan"
abmon	ABMON_2	%b	"Feb"
abmon	ABMON_3	%b	"Mar"
abmon	ABMON_4	%b	"Apr"
abmon	ABMON_5	%b	"May"
abmon	ABMON_6	%b	"Jun"
abmon	ABMON_7	%b	"Jul"
abmon	ABMON_8	%b	"Aug"
abmon	ABMON_9	%b	"Sep"
abmon	ABMON_10	%b	"Oct"
abmon	ABMON_11	%b	"Nov"
abmon	ABMON_12	%b	"Dec"
era	ERA	%EC, %Ey, %EY	N/A
era_d_fmt	ERA_D_FMT	%Ex	N/A
era_t_fmt	ERA_T_FMT	%EX	N/A
era_d_t_fmt	ERA_D_T_FMT	%Ec	N/A
alt_digits	ALT_DIGITS	%O	N/A

5205 XSI In the preceding table, the **langinfo Constant** column represents an XSI-conformant extension.

5206 The entry “N/A” indicates the value is not available in the POSIX locale.

5207 **7.3.6 LC\_MESSAGES**

5208 The *LC\_MESSAGES* category shall define the format and values used by various utilities for  
 5209 XSI affirmative and negative responses. This information is available through the *nl\_langinfo()*  
 5210 function.

5211 XSI The message catalog used by the standard utilities and selected by the *catopen()* function shall be  
 5212 determined by the setting of *NLSPATH*; see Chapter 8 (on page 157). The *LC\_MESSAGES*  
 5213 category can be specified as part of an *NLSPATH* substitution field.

5214 The following keywords shall be recognized as part of the locale definition file.

5215 **copy** Specify the name of an existing locale which shall be used as the definition of this |  
 5216 category. If this keyword is specified, no other keyword shall be specified. |

5217 **Note:** This is a *localedef* keyword, unavailable through *nl\_langinfo()*.

5218 **yesexpr** The operand consists of an extended regular expression (see Section 9.4 (on page  
 5219 171)) that describes the acceptable affirmative response to a question expecting an  
 5220 affirmative or negative response.

5221 **noexpr** The operand consists of an extended regular expression that describes the  
 5222 acceptable negative response to a question expecting an affirmative or negative  
 5223 response.

5224 **7.3.6.1 LC\_MESSAGES Category for the POSIX Locale**

5225 The format and values for affirmative and negative responses of the POSIX locale follow; the  
 5226 XSI code listing depicting the *localedef* input, the table representing the same information with the  
 5227 addition of *nl\_langinfo()* constants.

```
5228 LC_MESSAGES
5229 # This is the POSIX locale definition for
5230 # the LC_MESSAGES category.
5231 #
5232 yesexpr "<circumflex><left-square-bracket><y><Y><right-square-bracket>"
5233 #
5234 noexpr "<circumflex><left-square-bracket><n><N><right-square-bracket>"
5235 #
5236 END LC_MESSAGES
```

localedef Keyword	langinfo Constant	POSIX Locale Value
<b>yesexpr</b>	YEEXPR	"^[yY]"
<b>noexpr</b>	NOEXPR	"^[nN]"

5240 XSI In the preceding table, the **langinfo Constant** column represents an XSI-conformant extension.

## 5241 7.4 Locale Definition Grammar

5242 The grammar and lexical conventions in this section shall together describe the syntax for the  
 5243 locale definition source. The general conventions for this style of grammar are described in the  
 5244 Shell and Utilities volume of IEEE Std 1003.1-200x, Section 1.10, Grammar Conventions. The  
 5245 grammar shall take precedence over the text in this chapter.

### 5246 7.4.1 Locale Lexical Conventions

5247 The lexical conventions for the locale definition grammar are described in this section.

5248 The following tokens shall be processed (in addition to those string constants shown in the  
 5249 grammar):

5250	LOC_NAME	A string of characters representing the name of a locale.
5251	CHAR	Any single character.
5252	NUMBER	A decimal number, represented by one or more decimal digits.
5253	COLLSYMBOL	A symbolic name, enclosed between angle brackets. The string cannot duplicate any charmap symbol defined in the current charmap (if any), or a COLLELEMENT symbol.
5254		
5255		
5256	COLLELEMENT	A symbolic name, enclosed between angle brackets, which cannot duplicate either any charmap symbol or a COLLSYMBOL symbol.
5257		
5258	CHARCLASS	A string of alphanumeric characters from the portable character set, the first of which is not a digit, consisting of at least one and at most {CHARCLASS_NAME_MAX} bytes, and optionally surrounded by double-quotes.
5259		
5260		
5261		
5262	CHARSYMBOL	A symbolic name, enclosed between angle brackets, from the current charmap (if any).
5263		
5264	OCTAL_CHAR	One or more octal representations of the encoding of each byte in a single character. The octal representation consists of an escape character (normally a backslash) followed by two or more octal digits.
5265		
5266		
5267		
5268	HEX_CHAR	One or more hexadecimal representations of the encoding of each byte in a single character. The hexadecimal representation consists of an escape character followed by the constant x and two or more hexadecimal digits.
5269		
5270		
5271		
5272	DECIMAL_CHAR	One or more decimal representations of the encoding of each byte in a single character. The decimal representation consists of an escape character followed by a character 'd' and two or more decimal digits.
5273		
5274		
5275		
5276	ELLIPSIS	The string " . . . ".
5277	EXTENDED_REG_EXP	An extended regular expression as defined in the grammar in Section 9.5 (on page 175).
5278		
5279	EOL	The line termination character newline.

5280 **7.4.2 Locale Grammar**

```

5281      This section presents the grammar for the locale definition.
5282      %token          LOC_NAME
5283      %token          CHAR
5284      %token          NUMBER
5285      %token          COLLSYMBOL COLLELEMENT
5286      %token          CHARSYMBOL OCTAL_CHAR HEX_CHAR DECIMAL_CHAR
5287      %token          ELLIPSIS
5288      %token          EXTENDED_REG_EXP
5289      %token          EOL
5290      %start         locale_definition
5291      %%
5292      locale_definition : global_statements locale_categories
5293                       | locale_categories
5294                       ;
5295      global_statements : global_statements symbol_redefine
5296                       | symbol_redefine
5297                       ;
5298      symbol_redefine   : 'escape_char' CHAR EOL
5299                       | 'comment_char' CHAR EOL
5300                       ;
5301      locale_categories : locale_categories locale_category
5302                       | locale_category
5303                       ;
5304      locale_category   : lc_ctype | lc_collate | lc_messages
5305                       | lc_monetary | lc_numeric | lc_time
5306                       ;
5307      /* The following grammar rules are common to all categories */
5308      char_list         : char_list char_symbol
5309                       | char_symbol
5310                       ;
5311      char_symbol       : CHAR | CHARSYMBOL
5312                       | OCTAL_CHAR | HEX_CHAR | DECIMAL_CHAR
5313                       ;
5314      elem_list         : elem_list char_symbol
5315                       | elem_list COLLSYMBOL
5316                       | elem_list COLLELEMENT
5317                       | char_symbol
5318                       | COLLSYMBOL
5319                       | COLLELEMENT
5320                       ;
5321      symb_list         : symb_list COLLSYMBOL
5322                       | COLLSYMBOL
5323                       ;

```

```

5324     locale_name      : LOC_NAME
5325                       | ''' LOC_NAME '''
5326                       ;

5327     /* The following is the LC_CTYPE category grammar */

5328     lc_ctype          : ctype_hdr ctype_keywords      ctype_tlr
5329                       | ctype_hdr 'copy' locale_name EOL ctype_tlr
5330                       ;

5331     ctype_hdr         : 'LC_CTYPE' EOL
5332                       ;

5333     ctype_keywords    : ctype_keywords ctype_keyword
5334                       | ctype_keyword
5335                       ;

5336     ctype_keyword     : charclass_keyword charclass_list EOL
5337                       | charconv_keyword charconv_list EOL
5338                       | 'charclass' charclass_namelist EOL
5339                       ;

5340     charclass_namelist : charclass_namelist ';' CHARCLASS
5341                       | CHARCLASS
5342                       ;

5343     charclass_keyword : 'upper' | 'lower' | 'alpha' | 'digit'
5344                       | 'punct' | 'xdigit' | 'space' | 'print'
5345                       | 'graph' | 'blank' | 'cntrl' | 'alnum'
5346                       | CHARCLASS
5347                       ;

5348     charclass_list    : charclass_list ';' char_symbol
5349                       | charclass_list ';' ELLIPSIS ';' char_symbol
5350                       | char_symbol
5351                       ;

5352     charconv_keyword  : 'toupper'
5353                       | 'tolower'
5354                       ;

5355     charconv_list     : charconv_list ';' charconv_entry
5356                       | charconv_entry
5357                       ;

5358     charconv_entry    : '(' char_symbol ',' char_symbol ')'
5359                       ;

5360     ctype_tlr         : 'END' 'LC_CTYPE' EOL
5361                       ;

5362     /* The following is the LC_COLLATE category grammar */

5363     lc_collate        : collate_hdr collate_keywords    collate_tlr
5364                       | collate_hdr 'copy' locale_name EOL collate_tlr
5365                       ;

5366     collate_hdr       : 'LC_COLLATE' EOL
5367                       ;

```

```

5368     collate_keywords      :           order_statements
5369                               | opt_statements order_statements
5370                               ;

5371     opt_statements         : opt_statements collating_symbols
5372                               | opt_statements collating_elements
5373                               | collating_symbols
5374                               | collating_elements
5375                               ;

5376     collating_symbols      : 'collating-symbol' COLLSYMBOL EOL
5377                               ;

5378     collating_elements     : 'collating-element' COLLELEMENT
5379                               | 'from' ''' elem_list ''' EOL
5380                               ;

5381     order_statements       : order_start collation_order order_end
5382                               ;

5383     order_start            : 'order_start' EOL
5384                               | 'order_start' order_opts EOL
5385                               ;

5386     order_opts             : order_opts ';' order_opt
5387                               | order_opt
5388                               ;

5389     order_opt              : order_opt ',' opt_word
5390                               | opt_word
5391                               ;

5392     opt_word               : 'forward' | 'backward' | 'position'
5393                               ;

5394     collation_order        : collation_order collation_entry
5395                               | collation_entry
5396                               ;

5397     collation_entry        : COLLSYMBOL EOL
5398                               | collation_element weight_list EOL
5399                               | collation_element                EOL
5400                               ;

5401     collation_element      : char_symbol
5402                               | COLLELEMENT
5403                               | ELLIPSIS
5404                               | 'UNDEFINED'
5405                               ;

5406     weight_list           : weight_list ';' weight_symbol
5407                               | weight_list ';'
5408                               | weight_symbol
5409                               ;

5410     weight_symbol         : /* empty */
5411                               | char_symbol
5412                               | COLLSYMBOL
5413                               | ''' elem_list '''

```

```

5414         | ''' symb_list '''
5415         | ELLIPSIS
5416         | 'IGNORE'
5417         ;
5418     order_end      : 'order_end' EOL
5419         ;
5420     collate_tlr    : 'END' 'LC_COLLATE' EOL
5421         ;
5422     /* The following is the LC_MESSAGES category grammar */
5423     lc_messages     : messages_hdr messages_keywords      messages_tlr
5424         | messages_hdr 'copy' locale_name EOL messages_tlr
5425         ;
5426     messages_hdr    : 'LC_MESSAGES' EOL
5427         ;
5428     messages_keywords : messages_keywords messages_keyword
5429         | messages_keyword
5430         ;
5431     messages_keyword : 'yesexpr' ''' EXTENDED_REG_EXP ''' EOL
5432         | 'noexpr'   ''' EXTENDED_REG_EXP ''' EOL
5433         ;
5434     messages_tlr    : 'END' 'LC_MESSAGES' EOL
5435         ;
5436     /* The following is the LC_MONETARY category grammar */
5437     lc_monetary     : monetary_hdr monetary_keywords      monetary_tlr
5438         | monetary_hdr 'copy' locale_name EOL monetary_tlr
5439         ;
5440     monetary_hdr    : 'LC_MONETARY' EOL
5441         ;
5442     monetary_keywords : monetary_keywords monetary_keyword
5443         | monetary_keyword
5444         ;
5445     monetary_keyword : mon_keyword_string mon_string EOL
5446         | mon_keyword_char NUMBER EOL
5447         | mon_keyword_char '-1' EOL
5448         | mon_keyword_grouping mon_group_list EOL
5449         ;
5450     mon_keyword_string : 'int_curr_symbol' | 'currency_symbol'
5451         | 'mon_decimal_point' | 'mon_thousands_sep'
5452         | 'positive_sign' | 'negative_sign'
5453         ;
5454     mon_string       : ''' char_list '''
5455         | '''
5456         ;

```

```

5457     mon_keyword_char      : 'int_frac_digits' | 'frac_digits'
5458                             | 'p_cs_precedes' | 'p_sep_by_space'
5459                             | 'n_cs_precedes' | 'n_sep_by_space'
5460                             | 'p_sign_posn' | 'n_sign_posn'
5461                             ;

5462     mon_keyword_grouping   : 'mon_grouping'
5463                             ;

5464     mon_group_list         : NUMBER
5465                             | mon_group_list ';' NUMBER
5466                             ;

5467     monetary_tlr          : 'END' 'LC_MONETARY' EOL
5468                             ;

5469     /* The following is the LC_NUMERIC category grammar */
5470     lc_numeric              : numeric_hdr numeric_keywords      numeric_tlr
5471                             | numeric_hdr 'copy' locale_name EOL numeric_tlr
5472                             ;

5473     numeric_hdr            : 'LC_NUMERIC' EOL
5474                             ;

5475     numeric_keywords       : numeric_keywords numeric_keyword
5476                             | numeric_keyword
5477                             ;

5478     numeric_keyword        : num_keyword_string num_string EOL
5479                             | num_keyword_grouping num_group_list EOL
5480                             ;

5481     num_keyword_string     : 'decimal_point'
5482                             | 'thousands_sep'
5483                             ;

5484     num_string             : '"' char_list '"'
5485                             | '""'
5486                             ;

5487     num_keyword_grouping   : 'grouping'
5488                             ;

5489     num_group_list         : NUMBER
5490                             | num_group_list ';' NUMBER
5491                             ;

5492     numeric_tlr            : 'END' 'LC_NUMERIC' EOL
5493                             ;

5494     /* The following is the LC_TIME category grammar */
5495     lc_time                 : time_hdr time_keywords           time_tlr
5496                             | time_hdr 'copy' locale_name EOL time_tlr
5497                             ;

5498     time_hdr               : 'LC_TIME' EOL
5499                             ;

```

```

5500     time_keywords      : time_keywords time_keyword
5501                           | time_keyword
5502                           ;
5503     time_keyword        : time_keyword_name time_list EOL
5504                           | time_keyword_fmt time_string EOL
5505                           | time_keyword_opt time_list EOL
5506                           ;
5507     time_keyword_name    : 'abday' | 'day' | 'abmon' | 'mon'
5508                           ;
5509     time_keyword_fmt     : 'd_t_fmt' | 'd_fmt' | 't_fmt'
5510                           | 'am_pm' | 't_fmt_ampm'
5511                           ;
5512     time_keyword_opt     : 'era' | 'era_d_fmt' | 'era_t_fmt'
5513                           | 'era_d_t_fmt' | 'alt_digits'
5514                           ;
5515     time_list            : time_list ';' time_string
5516                           | time_string
5517                           ;
5518     time_string         : '"' char_list '"'
5519                           ;
5520     time_tlr            : 'END' 'LC_TIME' EOL
5521                           ;

```



# Environment Variables

5523

## 5524 8.1 Environment Variable Definition

5525 Environment variables defined in this chapter affect the operation of multiple utilities, functions,  
 5526 and applications. There are other environment variables that are of interest only to specific  
 5527 utilities. Environment variables that apply to a single utility only are defined as part of the  
 5528 utility description. See the ENVIRONMENT VARIABLES section of the utility descriptions in  
 5529 the Shell and Utilities volume of IEEE Std 1003.1-200x for information on environment variable  
 5530 usage.

5531 The value of an environment variable is a string of characters. For a C-language program, an  
 5532 array of strings called the environment shall be made available when a process begins. The array  
 5533 is pointed to by the external variable *environ*, which is defined as:

```
5534     extern char **environ;
```

5535 These strings have the form *name=value*; *names* shall not contain the character '='. For values to  
 5536 be portable across systems conforming to IEEE Std 1003.1-200x, the value shall be composed of  
 5537 characters from the portable character set (except NUL and as indicated below). There is no  
 5538 meaning associated with the order of strings in the environment. If more than one string in a  
 5539 process' environment has the same *name*, the consequences are undefined.

5540 Environment variable names used by the utilities in the Shell and Utilities volume of  
 5541 IEEE Std 1003.1-200x consist solely of uppercase letters, digits, and the '\_' (underscore) from  
 5542 the characters defined in Table 6-1 (on page 111) and do not begin with a digit. Other characters  
 5543 may be permitted by an implementation; applications shall tolerate the presence of such names.  
 5544 Uppercase and lowercase letters shall retain their unique identities and shall not be folded  
 5545 together. The name space of environment variable names containing lowercase letters is  
 5546 reserved for applications. Applications can define any environment variables with names from  
 5547 this name space without modifying the behavior of the standard utilities.

5548 **Note:** Other applications may have difficulty dealing with environment variable names that start  
 5549 with a digit. For this reason, use of such names is not recommended anywhere.

5550 The *values* that the environment variables may be assigned are not restricted except that they are  
 5551 considered to end with a null byte and the total space used to store the environment and the  
 5552 arguments to the process is limited to {ARG\_MAX} bytes.

5553 Other *name=value* pairs may be placed in the environment by, for example, calling any of the  
 5554 XSI *setenv()*, *unsetenv()*, or *putenv()* functions, manipulating the *environ* variable, or by using *envp*  
 5555 arguments when creating a process; see *exec* in the System Interfaces volume of  
 5556 IEEE Std 1003.1-200x.

5557 It is unwise to conflict with certain variables that are frequently exported by widely used  
 5558 command interpreters and applications:

5559					
5560		ARFLAGS	IFS	MAILPATH	PS1
5561		CC	LANG	MAILRC	PS2
5562		CDPATH	LC_ALL	MAKEFLAGS	PS3
5563		CFLAGS	LC_COLLATE	MAKESHELL	PS4
5564		CHARSET	LC_CTYPE	MANPATH	PWD
5565		COLUMNS	LC_MESSAGES	MBOX	RANDOM
5566		DATMSK	LC_MONETARY	MORE	SECONDS
5567		DEAD	LC_NUMERIC	MSGVERB	SHELL
5568		EDITOR	LC_TIME	NLSPATH	TERM
5569		ENV	LDFLAGS	NPROC	TERMCAP
5570		EXINIT	LEX	OLDPWD	TERMINFO
5571		FC	LFLAGS	OPTARG	TMPDIR
5572		FCEDIT	LINENO	OPTERR	TZ
5573		FFLAGS	LINES	OPTIND	USER
5574		GET	LISTER	PAGER	VISUAL
5575		GFLAGS	LOGNAME	PATH	YACC
5576		HISTFILE	LPDEST	PPID	YFLAGS
5577		HISTORY	MAIL	PRINTER	
5578		HISTSZ	MAILCHECK	PROCLANG	
5579		HOME	MAILER	PROJECTDIR	

5580 If the variables in the following two sections are present in the environment during the  
5581 execution of an application or utility, they shall be given the meaning described below. Some are  
5582 placed into the environment by the implementation at the time the user logs in; all can be added  
5583 or changed by the user or any ancestor of the current process. The implementation adds or  
5584 changes environment variables named in IEEE Std 1003.1-200x only as specified in  
5585 IEEE Std 1003.1-200x. If they are defined in the application's environment, the utilities in the  
5586 Shell and Utilities volume of IEEE Std 1003.1-200x and the functions in the System Interfaces  
5587 volume of IEEE Std 1003.1-200x assume they have the specified meaning. Conforming  
5588 applications shall not set these environment variables to have meanings other than as described.  
5589 See *getenv()* and the Shell and Utilities volume of IEEE Std 1003.1-200x, Section 2.12, Shell  
5590 Execution Environment for methods of accessing these variables.

## 5591 8.2 Internationalization Variables

5592 This section describes environment variables that are relevant to the operation of  
5593 internationalized interfaces described in IEEE Std 1003.1-200x.

5594 Users may use the following environment variables to announce specific localization  
5595 requirements to applications. Applications can retrieve this information using the *setlocale()*  
5596 function to initialize the correct behavior of the internationalized interfaces. The descriptions of  
5597 the internationalization environment variables describe the resulting behavior only when the  
5598 application locale is initialized in this way. The use of the internationalization variables by  
5599 utilities described in the Shell and Utilities volume of IEEE Std 1003.1-200x are described in the  
5600 ENVIRONMENT VARIABLES section for those utilities in addition to the global effects  
5601 described in this section.

5602 *LANG* This variable shall determine the locale category for native language, local  
5603 customs, and coded character set in the absence of the *LC\_ALL* and other *LC\_\**  
5604 (*LC\_COLLATE*, *LC\_CTYPE*, *LC\_MESSAGES*, *LC\_MONETARY*, *LC\_NUMERIC*,  
5605 *LC\_TIME*) environment variables. This can be used by applications to  
5606 determine the language to use for error messages and instructions, collating  
5607 sequences, date formats, and so on.

5608	<i>LC_ALL</i>	This variable shall determine the values for all locale categories. The value of the <i>LC_ALL</i> environment variable has precedence over any of the other environment variables starting with <i>LC_(LC_COLLATE, LC_CTYPE, LC_MESSAGES, LC_MONETARY, LC_NUMERIC, LC_TIME)</i> and the <i>LANG</i> environment variable.
5609		
5610		
5611		
5612		
5613	<i>LC_COLLATE</i>	This variable shall determine the locale category for character collation. It determines collation information for regular expressions and sorting, including equivalence classes and multi-character collating elements, in various utilities and the <i>strcoll()</i> and <i>strxfrm()</i> functions. Additional semantics of this variable, if any, are implementation-defined.
5614		
5615		
5616		
5617		
5618	<i>LC_CTYPE</i>	This variable shall determine the locale category for character handling functions, such as <i>tolower()</i> , <i>toupper()</i> , and <i>isalpha()</i> . This environment variable determines the interpretation of sequences of bytes of text data as characters (for example, single as opposed to multi-byte characters), the classification of characters (for example, alpha, digit, graph), and the behavior of character classes. Additional semantics of this variable, if any, are implementation-defined.
5619		
5620		
5621		
5622		
5623		
5624		
5625	<i>LC_MESSAGES</i>	This variable shall determine the locale category for processing affirmative and negative responses and the language and cultural conventions in which messages should be written. It also affects the behavior of the <i>catopen()</i>
5626		
5627	XSI	function in determining the message catalog. Additional semantics of this
5628		variable, if any, are implementation-defined. The language and cultural
5629		conventions of diagnostic and informative messages whose format is
5630		unspecified by IEEE Std 1003.1-200x should be affected by the setting of
5631		<i>LC_MESSAGES</i> .
5632		
5633	<i>LC_MONETARY</i>	This variable shall determine the locale category for monetary-related numeric
5634		formatting information. Additional semantics of this variable, if any, are
5635		implementation-defined.
5636	<i>LC_NUMERIC</i>	This variable shall determine the locale category for numeric formatting (for
5637		example, thousands separator and radix character) information in various
5638		utilities as well as the formatted I/O operations in <i>printf()</i> and <i>scanf()</i> and the
5639		string conversion functions in <i>strtod()</i> . Additional semantics of this variable,
5640		if any, are implementation-defined.
5641	<i>LC_TIME</i>	This variable shall determine the locale category for date and time formatting
5642		information. It affects the behavior of the time functions in <i>strftime()</i> .
5643		Additional semantics of this variable, if any, are implementation-defined.
5644	XSI	<i>NLSPATH</i> This variable shall contain a sequence of templates that the <i>catopen()</i>
5645		function uses when attempting to locate message catalogs. Each template consists of an
5646		optional prefix, one or more conversion specifications, a filename, and an
5647		optional suffix.
5648		For example:
5649		<code>NLSPATH="/system/nlslib/%N.cat"</code>
5650		defines that <i>catopen()</i> should look for all message catalogs in the directory
5651		<code>/system/nlslib</code> , where the catalog name should be constructed from the <i>name</i>
5652		parameter passed to <i>catopen()</i> ( <code>%N</code> ), with the suffix <code>.cat</code> .
5653		Conversion specifications consist of a <code>'%'</code> symbol, followed by a single-letter
5654		keyword. The following keywords are currently defined:

5655	%N	The value of the <i>name</i> parameter passed to <i>catopen()</i> .
5656	%L	The value of the <i>LC_MESSAGES</i> category.
5657	%l	The <i>language</i> element from the <i>LC_MESSAGES</i> category.
5658	%t	The <i>territory</i> element from the <i>LC_MESSAGES</i> category.
5659	%c	The <i>codeset</i> element from the <i>LC_MESSAGES</i> category.
5660	%%	A single '%' character.
5661		An empty string is substituted if the specified value is not currently defined.
5662		The separators underscore ('_') and period ('.') are not included in the %t
5663		and %c conversion specifications.
5664		Templates defined in <i>NLSPATH</i> are separated by colons (':'). A leading or
5665		two adjacent colons "::" is equivalent to specifying %N. For example:
5666		<code>NLSPATH=":%N.cat:/nlslib/%L/%N.cat"</code>
5667		indicates to <i>catopen()</i> that it should look for the requested message catalog in
5668		<i>name</i> , <i>name.cat</i> , and <i>/nlslib/category/name.cat</i> , where <i>category</i> is the value of the
5669		<i>LC_MESSAGES</i> category of the current locale.
5670		Users should not set the <i>NLSPATH</i> variable unless they have a specific reason
5671		to override the default system path. Setting <i>NLSPATH</i> to override the default
5672		system path produces undefined results in the standard utilities and in
5673		applications with appropriate privileges.
5674		The environment variables <i>LANG</i> , <i>LC_ALL</i> , <i>LC_COLLATE</i> , <i>LC_CTYPE</i> , <i>LC_MESSAGES</i> ,
5675	XSI	<i>LC_MONETARY</i> , <i>LC_NUMERIC</i> , <i>LC_TIME</i> , and <i>NLSPATH</i> provide for the support of
5676		internationalized applications. The standard utilities shall make use of these environment
5677		variables as described in this section and the individual ENVIRONMENT VARIABLES sections
5678		for the utilities. If these variables specify locale categories that are not based upon the same
5679		underlying codeset, the results are unspecified.
5680		The values of locale categories shall be determined by a precedence order; the first condition met
5681		below determines the value:
5682		1. If the <i>LC_ALL</i> environment variable is defined and is not null, the value of <i>LC_ALL</i> shall be
5683		used.
5684		2. If the <i>LC_*</i> environment variable ( <i>LC_COLLATE</i> , <i>LC_CTYPE</i> , <i>LC_MESSAGES</i> ,
5685		<i>LC_MONETARY</i> , <i>LC_NUMERIC</i> , <i>LC_TIME</i> ) is defined and is not null, the value of the
5686		environment variable shall be used to initialize the category that corresponds to the
5687		environment variable.
5688		3. If the <i>LANG</i> environment variable is defined and is not null, the value of the <i>LANG</i>
5689		environment variable shall be used.
5690		4. If the <i>LANG</i> environment variable is not set or is set to the empty string, the
5691		implementation-defined default locale shall be used.
5692		If the locale value is "C" or "POSIX", the POSIX locale shall be used and the standard utilities
5693		behave in accordance with the rules in Section 7.2 (on page 120) for the associated category.
5694		If the locale value begins with a slash, it shall be interpreted as the pathname of a file that was
5695		created in the output format used by the <i>localedf</i> utility; see OUTPUT FILES under <i>localedf</i> .
5696		Referencing such a pathname shall result in that locale being used for the indicated category.

5697 XSI If the locale value has the form:

5698 `language[_territory][.codeset]`

5699 it refers to an implementation-provided locale, where settings of language, territory, and codeset  
5700 are implementation-defined.

5701 `LC_COLLATE`, `LC_CTYPE`, `LC_MESSAGES`, `LC_MONETARY`, `LC_NUMERIC`, and `LC_TIME` are  
5702 defined to accept an additional field `@modifier`, which allows the user to select a specific instance  
5703 of localization data within a single category (for example, for selecting the dictionary as opposed  
5704 to the character ordering of data). The syntax for these environment variables is thus defined as:

5705 `[language[_territory][.codeset][@modifier]]`

5706 For example, if a user wanted to interact with the system in French, but required to sort German  
5707 text files, `LANG` and `LC_COLLATE` could be defined as:

5708 `LANG=Fr_FR`  
5709 `LC_COLLATE=De_DE`

5710 This could be extended to select dictionary collation (say) by use of the `@modifier` field; for  
5711 example:

5712 `LC_COLLATE=De_DE@dict`

5713

5714 An implementation may support other formats.

5715 If the locale value is not recognized by the implementation, the behavior is unspecified.

5716 At runtime, these values are bound to a program's locale by calling the `setlocale()` function.

5717 Additional criteria for determining a valid locale name are implementation-defined.

### 5718 8.3 Other Environment Variables

5719 **`COLUMNS`** This variable shall represent a decimal integer >0 used to indicate the user's  
5720 preferred width in column positions for the terminal screen or window; see  
5721 Section 3.103 (on page 47). If this variable is unset or null, the implementation  
5722 determines the number of columns, appropriate for the terminal or window,  
5723 in an unspecified manner. When `COLUMNS` is set, any terminal-width  
5724 information implied by `TERM` is overridden. Users and conforming |  
5725 applications should not set `COLUMNS` unless they wish to override the |  
5726 system selection and produce output unrelated to the terminal characteristics.

5727 Users should not need to set this variable in the environment unless there is a  
5728 specific reason to override the implementation's default behavior, such as to  
5729 display data in an area arbitrarily smaller than the terminal or window.

5730 XSI **`DATMSK`** Indicates the pathname of the template file used by `getdate()`.

5731 **`HOME`** The system shall initialize this variable at the time of login to be a pathname of  
5732 the user's home directory. See <`pwd.h`>.

5733 **`LINES`** This variable shall represent a decimal integer >0 used to indicate the user's  
5734 preferred number of lines on a page or the vertical screen or window size in  
5735 lines. A line in this case is a vertical measure large enough to hold the tallest  
5736 character in the character set being displayed. If this variable is unset or null,  
5737 the implementation determines the number of lines, appropriate for the

5738		terminal or window (size, terminal baud rate, and so on), in an unspecified manner. When <i>LINES</i> is set, any terminal-height information implied by
5739		<i>TERM</i> is overridden. Users and conforming applications should not set <i>LINES</i>
5740		unless they wish to override the system selection and produce output
5741		unrelated to the terminal characteristics.
5742		
5743		Users should not need to set this variable in the environment unless there is a
5744		specific reason to override the implementation's default behavior, such as to
5745		display data in an area arbitrarily smaller than the terminal or window.
5746	<i>LOGNAME</i>	The system shall initialize this variable at the time of login to be the user's
5747		login name. See < <i>pwd.h</i> >. For a value of <i>LOGNAME</i> to be portable across
5748		implementations of IEEE Std 1003.1-200x, the value should be composed of
5749		characters from the portable filename character set.
5750	XSI <i>MSGVERB</i>	Describes which message components shall be used in writing messages by
5751		<i>fmtmsg()</i> .
5752	<i>PATH</i>	This variable shall represent the sequence of path prefixes that certain
5753		functions and utilities apply in searching for an executable file known only by
5754		a filename. The prefixes shall be separated by a colon (':'). When a non-
5755		zero-length prefix is applied to this filename, a slash shall be inserted between
5756		the prefix and the filename. A zero-length prefix is a legacy feature that
5757		indicates the current working directory. It appears as two adjacent colons
5758		(" : : "), as an initial colon preceding the rest of the list, or as a trailing colon
5759		following the rest of the list. A strictly conforming application shall use an
5760		actual pathname (such as <i>.</i> ) to represent the current working directory in
5761		<i>PATH</i> . The list shall be searched from beginning to end, applying the filename
5762		to each prefix, until an executable file with the specified name and appropriate
5763		execution permissions is found. If the pathname being sought contains a slash,
5764		the search through the path prefixes shall not be performed. If the pathname
5765		begins with a slash, the specified path is resolved (see Section 4.11 (on page
5766		98)). If <i>PATH</i> is unset or is set to null, the path search is implementation-
5767		defined.
5768	<i>PWD</i>	This variable shall represent an absolute pathname of the current working
5769		directory. It shall not contain any filename components of dot or dot-dot. The
5770		value is set by the <i>cd</i> utility.
5771	<i>SHELL</i>	This variable shall represent a pathname of the user's preferred command
5772		language interpreter. If this interpreter does not conform to the Shell
5773		Command Language in the Shell and Utilities volume of IEEE Std 1003.1-200x,
5774		Chapter 2, Shell Command Language, utilities may behave differently from
5775		those described in IEEE Std 1003.1-200x.
5776	<i>TMPDIR</i>	This variable shall represent a pathname of a directory made available for
5777		programs that need a place to create temporary files.
5778	<i>TERM</i>	This variable shall represent the terminal type for which output is to be
5779		prepared. This information is used by utilities and application programs
5780		wishing to exploit special capabilities specific to a terminal. The format and
5781		allowable values of this environment variable are unspecified.
5782	<i>TZ</i>	This variable shall represent timezone information. The contents of the
5783		environment variable named <i>TZ</i> shall be used by the <i>ctime()</i> , <i>localtime()</i> ,
5784		<i>strftime()</i> , and <i>mktime()</i> functions, and by various utilities, to override the
5785		default timezone. The value of <i>TZ</i> has one of the two forms (spaces inserted

5786 for clarity):

5787        : *characters*

5788 or:

5789        *std offset dst offset, rule*

5790 If *TZ* is of the first format (that is, if the first character is a colon), the  
5791 characters following the colon are handled in an implementation-defined  
5792 manner.

5793 The expanded format (for all *TZs* whose value does not have a colon as the  
5794 first character) is as follows:

5795        *stdoffset[dst[offset][,start[/time],end[/time]]]*

5796 Where:

5797 *std* and *dst*   Indicate no less than three, nor more than {TZNAME\_MAX},  
5798 bytes that are the designation for the standard (*std*) or the  
5799 alternative (*dst*—such as Daylight Savings Time) timezone. Only  
5800 *std* is required; if *dst* is missing, then the alternative time does  
5801 not apply in this locale.

5802 Each of these fields may occur in either of two formats quoted or  
5803 unquoted:

5804 — In the quoted form, the first character shall be the less-than  
5805 ('<') character and the last character shall be the greater-than  
5806 ('>') character. All characters between these quoting  
5807 characters shall be alphanumeric characters in the current  
5808 locale, the plus-sign ('+') character, or the minus-sign ('-')  
5809 character. The *std* and *dst* fields in this case shall not include  
5810 the quoting characters. |

5811 — In the unquoted form, all characters in these fields shall be  
5812 alphabetic characters in the current locale.

5813 The interpretation of these fields is unspecified if either field is  
5814 less than three bytes (except for the case when *dst* is missing),  
5815 more than {TZNAME\_MAX} bytes, or if they contain characters  
5816 other than those specified.

5817 *offset*       Indicates the value added to the local time to arrive at  
5818 Coordinated Universal Time. The *offset* has the form:

5819        *hh[:mm[:ss]]*

5820 The minutes (*mm*) and seconds (*ss*) are optional. The hour (*hh*)  
5821 shall be required and may be a single digit. The *offset* following  
5822 *std* shall be required. If no *offset* follows *dst*, the alternative time  
5823 is assumed to be one hour ahead of standard time. One or more  
5824 digits may be used; the value is always interpreted as a decimal  
5825 number. The hour shall be between zero and 24, and the minutes  
5826 (and seconds)—if present—between zero and 59. The result of  
5827 using values outside of this range is unspecified. If preceded by  
5828 a '-', the timezone shall be east of the Prime Meridian;  
5829 otherwise, it shall be west (which may be indicated by an  
5830 optional preceding '+').

5831 *rule* Indicates when to change to and back from the alternative time.  
 5832 The *rule* has the form:  
 5833  $date[/time], date[/time]$   
 5834 where the first *date* describes when the change from standard to  
 5835 alternative time occurs and the second *date* describes when the  
 5836 change back happens. Each *time* field describes when, in current  
 5837 local time, the change to the other time is made.  
 5838 The format of *date* is one of the following:  
 5839 *Jn* The Julian day  $n$  ( $1 \leq n \leq 365$ ). Leap days shall not be  
 5840 counted. That is, in all years—including leap years—  
 5841 February 28 is day 59 and March 1 is day 60. It is  
 5842 impossible to refer explicitly to the occasional February  
 5843 29.  
 5844 *n* The zero-based Julian day ( $0 \leq n \leq 365$ ). Leap days shall  
 5845 be counted, and it is possible to refer to February 29.  
 5846 *Mm.n.d* The  $d$ 'th day ( $0 \leq d \leq 6$ ) of week  $n$  of month  $m$  of the  
 5847 year ( $1 \leq n \leq 5$ ,  $1 \leq m \leq 12$ , where week 5 means “the  
 5848 last  $d$  day in month  $m$ ” which may occur in either the  
 5849 fourth or the fifth week). Week 1 is the first week in  
 5850 which the  $d$ 'th day occurs. Day zero is Sunday.  
 5851 The *time* has the same format as *offset* except that no leading sign  
 5852 ('-' or '+') is allowed. The default, if *time* is not given, shall be  
 5853 02:00:00.

# Regular Expressions

5854

5855 *Regular Expressions* (REs) provide a mechanism to select specific strings from a set of character  
5856 strings.

5857 Regular expressions are a context-independent syntax that can represent a wide variety of  
5858 character sets and character set orderings, where these character sets are interpreted according  
5859 to the current locale. While many regular expressions can be interpreted differently depending  
5860 on the current locale, many features, such as character class expressions, provide for contextual  
5861 invariance across locales.

5862 The Basic Regular Expression (BRE) notation and construction rules in Section 9.3 (on page 167)  
5863 shall apply to most utilities supporting regular expressions. Some utilities, instead, support the  
5864 Extended Regular Expressions (ERE) described in Section 9.4 (on page 171); any exceptions for  
5865 both cases are noted in the descriptions of the specific utilities using regular expressions. Both  
5866 BREs and ERAs are supported by the Regular Expression Matching interface in the System  
5867 Interfaces volume of IEEE Std 1003.1-200x under *regcomp()*, *regex()*, and related functions.

## 5868 9.1 Regular Expression Definitions

5869 For the purposes of this section, the following definitions shall apply:

### 5870 **entire regular expression**

5871 The concatenated set of one or more BREs or ERAs that make up the pattern specified for  
5872 string selection.

### 5873 **matched**

5874 A sequence of zero or more characters shall be said to be matched by a BRE or ERA when  
5875 the characters in the sequence correspond to a sequence of characters defined by the  
5876 pattern.

5877 Matching shall be based on the bit pattern used for encoding the character, not on the  
5878 graphic representation of the character. This means that if a character set contains two or  
5879 more encodings for a graphic symbol, or if the strings searched contain text encoded in  
5880 more than one codeset, no attempt is made to search for any other representation of the  
5881 encoded symbol. If that is required, the user can specify equivalence classes containing all  
5882 variations of the desired graphic symbol.

5883 The search for a matching sequence starts at the beginning of a string and stops when the  
5884 first sequence matching the expression is found, where *first* is defined to mean “begins  
5885 earliest in the string”. If the pattern permits a variable number of matching characters and  
5886 thus there is more than one such sequence starting at that point, the longest such sequence  
5887 is matched. For example: the BRE "bb\*" matches the second to fourth characters of *abbbc*,  
5888 and the ERA (*wee* | *week*)(*knights* | *night*) matches all ten characters of *weeknights*.

5889 Consistent with the whole match being the longest of the leftmost matches, each subpattern,  
5890 from left to right, shall match the longest possible string. For this purpose, a null string shall  
5891 be considered to be longer than no match at all. For example, matching the BRE  
5892 "\(.\*\).\*" against "abcdef", the subexpression "(\1)" is "abcdef", and matching  
5893 the BRE "\(a\*\).\*" against "bc", the subexpression "(\1)" is the null string.

5894 When a multi-character collating element in a bracket expression (see Section 9.3.5 (on page  
5895 168)) is involved, the longest sequence shall be measured in characters consumed from the

5896 string to be matched; that is, the collating element counts not as one element, but as the  
5897 number of characters it matches.

5898 **BRE (ERE) matching a single character**

5899 A BRE or ERE that shall match either a single character or a single collating element.

5900 Only a BRE or ERE of this type that includes a bracket expression (see Section 9.3.5 (on page  
5901 168)) can match a collating element.

5902 **BRE (ERE) matching multiple characters**

5903 A BRE or ERE that shall match a concatenation of single characters or collating elements.

5904 Such a BRE or ERE is made up from a BRE (ERE) matching a single character and BRE (ERE)  
5905 special characters.

5906 **invalid**

5907 This section uses the term *invalid* for certain constructs or conditions. Invalid REs shall  
5908 cause the utility or function using the RE to generate an error condition. When *invalid* is not  
5909 used, violations of the specified syntax or semantics for REs produce undefined results: this  
5910 may entail an error, enabling an extended syntax for that RE, or using the construct in error  
5911 as literal characters to be matched. For example, the BRE construct "`\{1,2,3\}`" does not  
5912 comply with the grammar. A conforming application cannot rely on it producing an error  
5913 nor matching the literal characters "`\{1,2,3\}`".

5914 **9.2 Regular Expression General Requirements**

5915 The requirements in this section shall apply to both basic and extended regular expressions.

5916 The use of regular expressions is generally associated with text processing. REs (BREs and EREs)  
5917 operate on text strings; that is, zero or more characters followed by an end-of-string delimiter  
5918 (typically NUL). Some utilities employing regular expressions limit the processing to lines; that  
5919 is, zero or more characters followed by a <newline>. In the regular expression processing  
5920 described in IEEE Std 1003.1-200x, the <newline> is regarded as an ordinary character and both a  
5921 period and a non-matching list can match one. The Shell and Utilities volume of  
5922 IEEE Std 1003.1-200x specifies within the individual descriptions of those standard utilities  
5923 employing regular expressions whether they permit matching of <newline>s; if not stated  
5924 otherwise, the use of literal <newline>s or any escape sequence equivalent produces undefined  
5925 results. Those utilities (like *grep*) that do not allow <newline>s to match are responsible for  
5926 eliminating any <newline> from strings before matching against the RE. The *regcomp()* function  
5927 in the System Interfaces volume of IEEE Std 1003.1-200x, however, can provide support for such  
5928 processing without violating the rules of this section.

5929 The interfaces specified in IEEE Std 1003.1-200x do not permit the inclusion of a NUL character  
5930 in an RE or in the string to be matched. If during the operation of a standard utility a NUL is  
5931 included in the text designated to be matched, that NUL may designate the end of the text string  
5932 for the purposes of matching.

5933 When a standard utility or function that uses regular expressions specifies that pattern matching  
5934 shall be performed without regard to the case (uppercase or lowercase) of either data or  
5935 patterns, then when each character in the string is matched against the pattern, not only the  
5936 character, but also its case counterpart (if any), shall be matched. This definition of case-  
5937 insensitive processing is intended to allow matching of multi-character collating elements as  
5938 well as characters, as each character in the string is matched using both its cases. For example,  
5939 in a locale where "Ch" is a multi-character collating element and where a matching list expression  
5940 matches such elements, the RE "[[.Ch.]]" when matched against the string "char", is in

5941 reality matched against "ch", "Ch", "cH", and "CH".  
 5942 The implementation shall support any regular expression that does not exceed 256 bytes in  
 5943 length.

## 5944 **9.3 Basic Regular Expressions**

### 5945 **9.3.1 BREs Matching a Single Character or Collating Element**

5946 A BRE ordinary character, a special character preceded by a backslash or a period, shall match a  
 5947 single character. A bracket expression shall match a single character or a single collating  
 5948 element.

### 5949 **9.3.2 BRE Ordinary Characters**

5950 An ordinary character is a BRE that matches itself: any character in the supported character set,  
 5951 except for the BRE special characters listed in Section 9.3.3.

5952 The interpretation of an ordinary character preceded by a backslash ('\*\*') is undefined, except  
 5953 for:

- 5954 • The characters ' ) ', ' ( ', ' { ', and ' } '
- 5955 • The digits 1 to 9 inclusive (see Section 9.3.6 (on page 170))
- 5956 • A character inside a bracket expression

### 5957 **9.3.3 BRE Special Characters**

5958 A *BRE special character* has special properties in certain contexts. Outside those contexts, or when  
 5959 preceded by a backslash, such a character is a BRE that matches the special character itself. The  
 5960 BRE special characters and the contexts in which they have their special meaning are as follows:

5961 . [ \*\* The period, left-bracket, and backslash shall be special except when used in a bracket  
 5962 expression (see Section 9.3.5 (on page 168)). An expression containing a '[' that is not  
 5963 preceded by a backslash and is not part of a bracket expression produces undefined  
 5964 results.

5965 \* The asterisk shall be special except when used:

- 5966 • In a bracket expression
- 5967 • As the first character of an entire BRE (after an initial '^', if any)
- 5968 • As the first character of a subexpression (after an initial '^', if any); see Section  
 5969 9.3.6 (on page 170)

5970 ^ The circumflex shall be special when used as:

- 5971 • An anchor (see Section 9.3.8 (on page 171))
- 5972 • The first character of a bracket expression (see Section 9.3.5 (on page 168))

5973 \$ The dollar sign shall be special when used as an anchor.

5974 **9.3.4 Periods in BREs**

5975 A period ( ' . ' ), when used outside a bracket expression, is a BRE that shall match any character  
5976 in the supported character set except NUL.

5977 **9.3.5 RE Bracket Expression**

5978 A bracket expression (an expression enclosed in square brackets, " [ ] ") is an RE that shall match |  
5979 a single collating element contained in the non-empty set of collating elements represented by |  
5980 the bracket expression.

5981 The following rules and definitions apply to bracket expressions:

5982 1. A *bracket expression* is either a matching list expression or a non-matching list expression. It  
5983 consists of one or more expressions: collating elements, collating symbols, equivalence  
5984 classes, character classes, or range expressions. The right-bracket ( ' ] ' ) shall lose its special  
5985 meaning and represents itself in a bracket expression if it occurs first in the list (after an  
5986 initial circumflex ( ' ^ ' ), if any). Otherwise, it shall terminate the bracket expression, unless  
5987 it appears in a collating symbol (such as " [ . ] . ] ") or is the ending right-bracket for a  
5988 collating symbol, equivalence class, or character class. The special characters ' . ' , ' \* ' ,  
5989 ' [ ' , and ' \ ' (period, asterisk, left-bracket, and backslash, respectively) shall lose their  
5990 special meaning within a bracket expression.

5991 The character sequences " [ . " , " [ = " , and " [ : " (left-bracket followed by a period, equals-  
5992 sign, or colon) shall be special inside a bracket expression and are used to delimit collating  
5993 symbols, equivalence class expressions, and character class expressions. These symbols  
5994 shall be followed by a valid expression and the matching terminating sequence " . ] " ,  
5995 "= ] " , or " : ] " , as described in the following items.

5996 2. A *matching list expression* specifies a list that shall match any single-character collating |  
5997 element in any of the expressions represented in the list. The first character in the list shall |  
5998 not be the circumflex; for example, "[ abc ] " is an RE that matches any of the characters |  
5999 ' a ' , ' b ' , or ' c ' . It is unspecified whether a matching list expression matches a multi-  
6000 character collating element that is matched by one of the expressions.

6001 3. A *non-matching list expression* begins with a circumflex ( ' ^ ' ), and specifies a list that shall |  
6002 match any single-character collating element except for the expressions represented in the |  
6003 list after the leading circumflex. For example, "[ ^ abc ] " is an RE that matches any |  
6004 character except the characters ' a ' , ' b ' , or ' c ' . It is unspecified whether a non-matching  
6005 list expression matches a multi-character collating element that is not matched by any of  
6006 the expressions. The circumflex shall have this special meaning only when it occurs first in  
6007 the list, immediately following the left-bracket.

6008 4. A *collating symbol* is a collating element enclosed within bracket-period ( " [ . " and " . ] ") |  
6009 delimiters. Collating elements are defined as described in Section 7.3.2.4 (on page 133). |  
6010 Conforming applications shall represent multi-character collating elements as collating |  
6011 symbols when it is necessary to distinguish them from a list of the individual characters |  
6012 that make up the multi-character collating element. For example, if the string " ch " is a  
6013 collating element defined using the line:

```
6014 collating-element <ch-digraph> from "<c><h>"
```

6015 in the locale definition, the expression "[ [ . ch. ] ] " shall be treated as an RE containing  
6016 the collating symbol ' ch ' , while "[ ch ] " shall be treated as an RE matching ' c ' or ' h ' .  
6017 Collating symbols are recognized only inside bracket expressions. If the string is not a  
6018 collating element in the current locale, the expression is invalid.

6019 5. An *equivalence class expression* shall represent the set of collating elements belonging to an  
 6020 equivalence class, as described in Section 7.3.2.4 (on page 133). Only primary equivalence  
 6021 classes shall be recognized. The class shall be expressed by enclosing any one of the  
 6022 collating elements in the equivalence class within bracket-equal ("[" and "=")  
 6023 delimiters. For example, if 'a', 'à', and 'â' belong to the same equivalence class, then  
 6024 "[[=a=]b]", "[[=à=]b]", and "[[=â=]b]" are each equivalent to "[aââb]". If the  
 6025 collating element does not belong to an equivalence class, the equivalence class expression  
 6026 shall be treated as a *collating symbol*.

6027 6. A *character class expression* shall represent the union of two sets: |  
 6028 a. The set of single-character collating elements whose characters belong to the |  
 6029 character class, as defined in the *LC\_CTYPE* category in the current locale. |  
 6030 b. An unspecified set of multi-character collating elements. |

6031 All character classes specified in the current locale shall be recognized. A character class |  
 6032 expression is expressed as a character class name enclosed within bracket-colon ("[" and |  
 6033 ":]") delimiters.

6034 The following character class expressions shall be supported in all locales:

```
6035      [:alnum:]      [:cntrl:]      [:lower:]      [:space:]
6036      [:alpha:]     [:digit:]     [:print:]     [:upper:]
6037      [:blank:]     [:graph:]     [:punct:]     [:xdigit:]
```

6038 XSI In addition, character class expressions of the form:

```
6039      [:name:]
```

6040 are recognized in those locales where the *name* keyword has been given a **charclass**  
 6041 definition in the *LC\_CTYPE* category.

6042 7. In the POSIX locale, a range expression represents the set of collating elements that fall  
 6043 between two elements in the collation sequence, inclusive. In other locales, a range  
 6044 expression has unspecified behavior: strictly conforming applications shall not rely on  
 6045 whether the range expression is valid, or on the set of collating elements matched. A range  
 6046 expression shall be expressed as the starting point and the ending point separated by a  
 6047 hyphen ('-').

6048 In the following, all examples assume the POSIX locale.

6049 The starting range point and the ending range point shall be a collating element or  
 6050 collating symbol. An equivalence class expression used as a starting or ending point of a  
 6051 range expression produces unspecified results. An equivalence class can be used portably  
 6052 within a bracket expression, but only outside the range. If the represented set of collating  
 6053 elements is empty, it is unspecified whether the expression matches nothing, or is treated  
 6054 as invalid.

6055 The interpretation of range expressions where the ending range point is also the starting  
 6056 range point of a subsequent range expression (for example, "[a-m-o]") is undefined.

6057 The hyphen character shall be treated as itself if it occurs first (after an initial '^', if any)  
 6058 or last in the list, or as an ending range point in a range expression. As examples, the  
 6059 expressions "[-ac]" and "[ac-]" are equivalent and match any of the characters 'a',  
 6060 'c', or '-'; "[^ac-]" and "[^ac-]" are equivalent and match any characters except  
 6061 'a', 'c', or '-'; the expression "[%--]" matches any of the characters between '%' and  
 6062 '-' inclusive; the expression "[--@]" matches any of the characters between '-' and  
 6063 '@' inclusive; and the expression "[a--@]" is either invalid or equivalent to '@',

6064 because the letter 'a' follows the symbol '-' in the POSIX locale. To use a hyphen as the  
 6065 starting range point, it shall either come first in the bracket expression or be specified as a  
 6066 collating symbol; for example, "[ ] [.-.] -0]", which matches either a right bracket or  
 6067 any character or collating element that collates between hyphen and 0, inclusive.

6068 If a bracket expression specifies both '-' and ']', the ']' shall be placed first (after the  
 6069 '^', if any) and the '-' last within the bracket expression.

### 6070 9.3.6 BREs Matching Multiple Characters

6071 The following rules can be used to construct BREs matching multiple characters from BREs  
 6072 matching a single character:

6073 1. The concatenation of BREs shall match the concatenation of the strings matched by each  
 6074 component of the BRE.

6075 2. A *subexpression* can be defined within a BRE by enclosing it between the character pairs  
 6076 "\(" and "\)". Such a subexpression shall match whatever it would have matched  
 6077 without the "\(" and "\)", except that anchoring within subexpressions is optional  
 6078 behavior; see Section 9.3.8 (on page 171). Subexpressions can be arbitrarily nested.

6079 3. The *back-reference expression* '\n' shall match the same (possibly empty) string of |  
 6080 characters as was matched by a subexpression enclosed between "\(" and "\)"  
 6081 preceding the '\n'. The character 'n' shall be a digit from 1 through 9, specifying the  
 6082 *n*th subexpression (the one that begins with the *n*th "\(" from the beginning of the  
 6083 pattern and ends with the corresponding paired "\)"). The expression is invalid if less  
 6084 than *n* subexpressions precede the '\n'. For example, the expression "\(.\*)\1\$" |  
 6085 matches a line consisting of two adjacent appearances of the same string, and the  
 6086 expression "\(a\)\*\1" fails to match 'a'. When the referenced subexpression matched  
 6087 more than one string, the back-referenced expression shall refer to the last matched string.  
 6088 If the subexpression referenced by the back-reference matches more than one string  
 6089 because of an asterisk ('\*') or an interval expression (see item (5)), the back-reference  
 6090 shall match the last (rightmost) of these strings.

6091 4. When a BRE matching a single character, a subexpression, or a back-reference is followed  
 6092 by the special character asterisk ('\*'), together with that asterisk it shall match what zero  
 6093 or more consecutive occurrences of the BRE would match. For example, "[ab]\*" and  
 6094 "[ab][ab]" are equivalent when matching the string "ab".

6095 5. When a BRE matching a single character, a subexpression, or a back-reference is followed  
 6096 by an *interval expression* of the format "\{m\}", "\{m,\}", or "\{m,n\}", together with  
 6097 that interval expression it shall match what repeated consecutive occurrences of the BRE  
 6098 would match. The values of *m* and *n* are decimal integers in the range 0  
 6099  $\leq m \leq n \leq \{RE\_DUP\_MAX\}$ , where *m* specifies the exact or minimum number of occurrences  
 6100 and *n* specifies the maximum number of occurrences. The expression "\{m\}" shall match  
 6101 exactly *m* occurrences of the preceding BRE, "\{m,\}" shall match at least *m* occurrences,  
 6102 and "\{m,n\}" shall match any number of occurrences between *m* and *n*, inclusive.

6103 For example, in the string "abababcccccd" the BRE "c\{3\}" is matched by  
 6104 characters '7' to '9', the BRE "\(ab\)\{4,\}" is not matched at all, and the BRE  
 6105 "c\{1,3\}d" is matched by characters ten to thirteen.

6106 The behavior of multiple adjacent duplication symbols ('\*' and intervals) produces undefined  
 6107 results.

6108 A subexpression repeated by an asterisk ('\*') or an interval expression shall not match a null  
 6109 expression unless this is the only match for the repetition or it is necessary to satisfy the exact or

6110 minimum number of occurrences for the interval expression.

### 6111 9.3.7 BRE Precedence

6112 The order of precedence shall be as shown in the following table:

BRE Precedence (from high to low)	
6113 Collation-related bracket symbols	[==] [::] [..]
6114 Escaped characters	\<special character>
6115 Bracket expression	[ ]
6116 Subexpressions/back-references	\(\) \n
6117 Single-character-BRE duplication	* \{m,n\}
6118 Concatenation	
6119 Anchoring	^ \$
6120	

### 6121 9.3.8 BRE Expression Anchoring

6122 A BRE can be limited to matching strings that begin or end a line; this is called *anchoring*. The  
 6123 circumflex and dollar sign special characters shall be considered BRE anchors in the following  
 6124 contexts:

- 6125 1. A circumflex ('^') shall be an anchor when used as the first character of an entire BRE.  
 6126 The implementation may treat the circumflex as an anchor when used as the first character  
 6127 of a subexpression. The circumflex shall anchor the expression (or optionally  
 6128 subexpression) to the beginning of a string; only sequences starting at the first character of  
 6129 a string shall be matched by the BRE. For example, the BRE "^ab" matches "ab" in the  
 6130 string "abcdef", but fails to match in the string "cdefab". The BRE "\(^ab\)" may  
 6131 match the former string. A portable BRE shall escape a leading circumflex in a  
 6132 subexpression to match a literal circumflex.
- 6133 2. A dollar sign ('\$') shall be an anchor when used as the last character of an entire BRE.  
 6134 The implementation may treat a dollar sign as an anchor when used as the last character of  
 6135 a subexpression. The dollar sign shall anchor the expression (or optionally subexpression)  
 6136 to the end of the string being matched; the dollar sign can be said to match the end-of-  
 6137 string following the last character.
- 6138 3. A BRE anchored by both '^' and '\$' shall match only an entire string. For example, the  
 6139 BRE "^abcdef\$" matches strings consisting only of "abcdef".

## 6140 9.4 Extended Regular Expressions

6141 The *extended regular expression* (ERE) notation and construction rules shall apply to utilities  
 6142 defined as using extended regular expressions; any exceptions to the following rules are noted in  
 6143 the descriptions of the specific utilities using EREs.

6144 **9.4.1 EREs Matching a Single Character or Collating Element**

6145 An ERE ordinary character, a special character preceded by a backslash, or a period shall match  
 6146 a single character. A bracket expression shall match a single character or a single collating  
 6147 element. An *ERE matching a single character* enclosed in parentheses shall match the same as the  
 6148 ERE without parentheses would have matched.

6149 **9.4.2 ERE Ordinary Characters**

6150 An *ordinary character* is an ERE that matches itself. An ordinary character is any character in the  
 6151 supported character set, except for the ERE special characters listed in Section 9.4.3. The  
 6152 interpretation of an ordinary character preceded by a backslash ( '\ ' ) is undefined.

6153 **9.4.3 ERE Special Characters**

6154 An *ERE special character* has special properties in certain contexts. Outside those contexts, or  
 6155 when preceded by a backslash, such a character shall be an ERE that matches the special  
 6156 character itself. The extended regular expression special characters and the contexts in which  
 6157 they shall have their special meaning are as follows:

6158 . [ \ ( The period, left-bracket, backslash, and left-parenthesis shall be special except when  
 6159 used in a bracket expression (see Section 9.3.5 (on page 168)). Outside a bracket  
 6160 expression, a left-parenthesis immediately followed by a right-parenthesis produces  
 6161 undefined results.

6162 ) The right-parenthesis shall be special when matched with a preceding left-parenthesis,  
 6163 both outside a bracket expression.

6164 \* + ? { The asterisk, plus-sign, question-mark, and left-brace shall be special except when used  
 6165 in a bracket expression (see Section 9.3.5 (on page 168)). Any of the following uses  
 6166 produce undefined results:

- 6167 • If these characters appear first in an ERE, or immediately following a vertical-line,  
 6168 circumflex, or left-parenthesis

- 6169 • If a left-brace is not part of a valid interval expression (see Section 9.4.6 (on page  
 6170 173))

6171 | The vertical-line is special except when used in a bracket expression (see Section 9.3.5  
 6172 (on page 168)). A vertical-line appearing first or last in an ERE, or immediately  
 6173 following a vertical-line or a left-parenthesis, or immediately preceding a right-  
 6174 parenthesis, produces undefined results.

6175 ^ The circumflex shall be special when used as:

- 6176 • An anchor (see Section 9.4.9 (on page 174))

- 6177 • The first character of a bracket expression (see Section 9.3.5 (on page 168))

6178 \$ The dollar sign shall be special when used as an anchor.

6179 **9.4.4 Periods in EREs**

6180 A period ('.'), when used outside a bracket expression, is an ERE that shall match any  
6181 character in the supported character set except NUL.

6182 **9.4.5 ERE Bracket Expression**

6183 The rules for ERE Bracket Expressions are the same as for Basic Regular Expressions; see Section  
6184 9.3.5 (on page 168).

6185 **9.4.6 EREs Matching Multiple Characters**

6186 The following rules shall be used to construct EREs matching multiple characters from EREs  
6187 matching a single character:

- 6188 1. A *concatenation of EREs* shall match the concatenation of the character sequences matched  
6189 by each component of the ERE. A concatenation of EREs enclosed in parentheses shall  
6190 match whatever the concatenation without the parentheses matches. For example, both the  
6191 ERE "cd" and the ERE "(cd)" are matched by the third and fourth character of the string  
6192 "abcdefabcdef".
- 6193 2. When an ERE matching a single character or an ERE enclosed in parentheses is followed by  
6194 the special character plus-sign ('+'), together with that plus-sign it shall match what one  
6195 or more consecutive occurrences of the ERE would match. For example, the ERE  
6196 "b+(bc)" matches the fourth to seventh characters in the string "acabbbbcde". And,  
6197 "[ab]+" and "[ab][ab]\*" are equivalent.
- 6198 3. When an ERE matching a single character or an ERE enclosed in parentheses is followed by  
6199 the special character asterisk ('\*'), together with that asterisk it shall match what zero or  
6200 more consecutive occurrences of the ERE would match. For example, the ERE "b\*c"  
6201 matches the first character in the string "cabbbbcde", and the ERE "b\*cd" matches the  
6202 third to seventh characters in the string "cabbbbcdebbbbbbbcdbc". And, "[ab]\*" and  
6203 "[ab][ab]" are equivalent when matching the string "ab".
- 6204 4. When an ERE matching a single character or an ERE enclosed in parentheses is followed by  
6205 the special character question-mark ('?'), together with that question-mark it shall match  
6206 what zero or one consecutive occurrences of the ERE would match. For example, the ERE  
6207 "b?c" matches the second character in the string "acabbbbcde".
- 6208 5. When an ERE matching a single character or an ERE enclosed in parentheses is followed by  
6209 an *interval expression* of the format "{m}", "{m,}", or "{m,n}", together with that  
6210 interval expression it shall match what repeated consecutive occurrences of the ERE would  
6211 match. The values of *m* and *n* are decimal integers in the range  $0 \leq m \leq n \leq \{RE\_DUP\_MAX\}$ ,  
6212 where *m* specifies the exact or minimum number of occurrences and *n* specifies the  
6213 maximum number of occurrences. The expression "{m}" matches exactly *m* occurrences  
6214 of the preceding ERE, "{m,}" matches at least *m* occurrences, and "{m,n}" matches any  
6215 number of occurrences between *m* and *n*, inclusive.

6216 For example, in the string "abababcccccd" the ERE "c{3}" is matched by characters  
6217 '7' to '9' and the ERE "(ab){2,}" is matched by characters one to six.

6218 The behavior of multiple adjacent duplication symbols ('+', '\*', '?', and intervals) produces  
6219 undefined results.

6220 An ERE matching a single character repeated by an '\*', '?', or an interval expression shall not  
6221 match a null expression unless this is the only match for the repetition or it is necessary to satisfy  
6222 the exact or minimum number of occurrences for the interval expression.

6223 **9.4.7 ERE Alternation**

6224 Two EREs separated by the special character vertical-line ('|') shall match a string that is  
 6225 matched by either. For example, the ERE "a((bc)|d)" matches the string "abc" and the string  
 6226 "ad". Single characters, or expressions matching single characters, separated by the vertical bar  
 6227 and enclosed in parentheses, shall be treated as an ERE matching a single character.

6228 **9.4.8 ERE Precedence**

6229 The order of precedence shall be as shown in the following table:

ERE Precedence (from high to low)	
6230 Collation-related bracket symbols	[==] [::] [..]
6231 Escaped characters	\<special character>
6232 Bracket expression	[ ]
6233 Grouping	( )
6234 Single-character-ERE duplication	* + ? {m,n}
6235 Concatenation	
6236 Anchoring	^ \$
6237 Alternation	

6239 For example, the ERE "abba|cde" matches either the string "abba" or the string "cde"  
 6240 (rather than the string "abbade" or "abbcde", because concatenation has a higher order of  
 6241 precedence than alternation).

6242 **9.4.9 ERE Expression Anchoring**

6243 An ERE can be limited to matching strings that begin or end a line; this is called *anchoring*. The  
 6244 circumflex and dollar sign special characters shall be considered ERE anchors when used  
 6245 anywhere outside a bracket expression. This shall have the following effects:

- 6246 1. A circumflex ('^') outside a bracket expression shall anchor the expression or  
 6247 subexpression it begins to the beginning of a string; such an expression or subexpression  
 6248 can match only a sequence starting at the first character of a string. For example, the EREs  
 6249 "^ab" and "(^ab)" match "ab" in the string "abcdef", but fail to match in the string  
 6250 "cdefab", and the ERE "a^b" is valid, but can never match because the 'a' prevents the  
 6251 expression "^b" from matching starting at the first character.
- 6252 2. A dollar sign ('\$') outside a bracket expression shall anchor the expression or  
 6253 subexpression it ends to the end of a string; such an expression or subexpression can  
 6254 match only a sequence ending at the last character of a string. For example, the EREs  
 6255 "ef\$" and "(ef\$)" match "ef" in the string "abcdef", but fail to match in the string  
 6256 "cdefab", and the ERE "e\$f" is valid, but can never match because the 'f' prevents the  
 6257 expression "e\$" from matching ending at the last character.

6258 **9.5 Regular Expression Grammar**

6259 Grammars describing the syntax of both basic and extended regular expressions are presented in  
 6260 this section. The grammar takes precedence over the text. See the Shell and Utilities volume of  
 6261 IEEE Std 1003.1-200x, Section 1.10, Grammar Conventions.

6262 **9.5.1 BRE/ERE Grammar Lexical Conventions**

6263 The lexical conventions for regular expressions are as described in this section.

6264 Except as noted, the longest possible token or delimiter beginning at a given point is recognized.

6265 The following tokens are processed (in addition to those string constants shown in the  
 6266 grammar):

6267 COLL\_ELEM\_SINGLE

6268 Any single-character collating element, unless it is a META\_CHAR.

6269 COLL\_ELEM\_MULTI Any multi-character collating element.

6270 BACKREF

6271 Applicable only to basic regular expressions. The character string  
 consisting of '\ ' followed by a single-digit numeral, '1' to '9'.

6272 DUP\_COUNT

6273 Represents a numeric constant. It shall be an integer in the range 0  
 6274 ≤DUP\_COUNT ≤{RE\_DUP\_MAX}. This token is only recognized when  
 6275 the context of the grammar requires it. At all other times, digits not  
 preceded by '\ ' are treated as ORD\_CHAR.

6276 META\_CHAR

One of the characters:

6277 ^ When found first in a bracket expression

6278 – When found anywhere but first (after an initial '^', if any) or  
 6279 last in a bracket expression, or as the ending range point in a  
 6280 range expression

6281 ] When found anywhere but first (after an initial '^', if any) in a  
 6282 bracket expression

6283 L\_ANCHOR

6284 Applicable only to basic regular expressions. The character '^' when it  
 6285 appears as the first character of a basic regular expression and when not  
 6286 QUOTED\_CHAR. The '^' may be recognized as an anchor elsewhere;  
 see Section 9.3.8 (on page 171).

6287 ORD\_CHAR

A character, other than one of the special characters in SPEC\_CHAR.

6288 QUOTED\_CHAR

In a BRE, one of the character sequences:

6289 \^ \. \\* \[ \ \$ \ \

6290 In an ERE, one of the character sequences:

6291 \^ \. \[ \ \$ \ ( \ ) \ |  
 6292 \\* \+ \? \{ \ \

6293 R\_ANCHOR

6294 (Applicable only to basic regular expressions.) The character '\$' when it  
 6295 appears as the last character of a basic regular expression and when not  
 6296 QUOTED\_CHAR. The '\$' may be recognized as an anchor elsewhere;  
 see Section 9.3.8 (on page 171).

6297 SPEC\_CHAR

For basic regular expressions, one of the following special characters:

6298	.	Anywhere outside bracket expressions
6299	\	Anywhere outside bracket expressions
6300	[	Anywhere outside bracket expressions
6301	^	When used as an anchor (see Section 9.3.8 (on page 171)) or
6302		when first in a bracket expression
6303	\$	When used as an anchor
6304	*	Anywhere except first in an entire RE, anywhere in a bracket
6305		expression, directly following "\(", directly following an
6306		anchoring '^'
6307		For extended regular expressions, shall be one of the following special
6308		characters found anywhere outside bracket expressions:
6309	^ . [ \$ ( )	
6310	* + ? { \	
6311		(The close-parenthesis shall be considered special in this context only if
6312		matched with a preceding open-parenthesis.)

## 6313 9.5.2 RE and Bracket Expression Grammar

6314 This section presents the grammar for basic regular expressions, including the bracket  
6315 expression grammar that is common to both BREs and EREs.

```

6316 %token   ORD_CHAR QUOTED_CHAR DUP_COUNT
6317 %token   BACKREF L_ANCHOR R_ANCHOR
6318 %token   Back_open_paren  Back_close_paren
6319 /*      '\('           '\)'           */
6320 %token   Back_open_brace  Back_close_brace
6321 /*      '\{'           '\}'           */
6322 /* The following tokens are for the Bracket Expression
6323    grammar common to both REs and EREs. */
6324 %token   COLL_ELEM_SINGLE COLL_ELEM_MULTI META_CHAR
6325 %token   Open_equal Equal_close Open_dot Dot_close Open_colon Colon_close
6326 /*      '['           '='           '[' '.'           '.']'           '[' ':'           ':']' */
6327 %token   class_name
6328 /* class_name is a keyword to the LC_CTYPE locale category */
6329 /* (representing a character class) in the current locale */
6330 /* and is only recognized between [: and :] */
6331 %start   basic_reg_exp
6332 %%
6333 /* -----
6334    Basic Regular Expression
6335    -----
6336 */
6337 basic_reg_exp : RE_expression
6338               | L_ANCHOR
6339               | R_ANCHOR

```

```

6340         | L_ANCHOR                R_ANCHOR
6341         | L_ANCHOR RE_expression
6342         | RE_expression R_ANCHOR
6343         | L_ANCHOR RE_expression R_ANCHOR
6344         ;
6345 RE_expression : simple_RE
6346         | RE_expression simple_RE
6347         ;
6348 simple_RE : nondupl_RE
6349         | nondupl_RE RE_dupl_symbol
6350         ;
6351 nondupl_RE : one_char_or_coll_elem_RE
6352         | Back_open_paren RE_expression Back_close_paren
6353         | BACKREF
6354         ;
6355 one_char_or_coll_elem_RE : ORD_CHAR
6356         | QUOTED_CHAR
6357         | '.'
6358         | bracket_expression
6359         ;
6360 RE_dupl_symbol : '*'
6361         | Back_open_brace DUP_COUNT                Back_close_brace
6362         | Back_open_brace DUP_COUNT ','            Back_close_brace
6363         | Back_open_brace DUP_COUNT ',' DUP_COUNT Back_close_brace
6364         ;

6365 /* -----
6366    Bracket Expression
6367    -----
6368 */
6369 bracket_expression : '[' matching_list ']'
6370         | '[' nonmatching_list ']'
6371         ;
6372 matching_list : bracket_list
6373         ;
6374 nonmatching_list : '^' bracket_list
6375         ;
6376 bracket_list : follow_list
6377         | follow_list '-'
6378         ;
6379 follow_list : expression_term
6380         | follow_list expression_term
6381         ;
6382 expression_term : single_expression
6383         | range_expression
6384         ;
6385 single_expression : end_range
6386         | character_class
6387         | equivalence_class
6388         ;
6389 range_expression : start_range end_range
6390         | start_range '-'
6391         ;

```

```

6392     start_range      : end_range '-'
6393                       ;
6394     end_range         : COLL_ELEM_SINGLE
6395                       | collating_symbol
6396                       ;
6397     collating_symbol  : Open_dot COLL_ELEM_SINGLE Dot_close
6398                       | Open_dot COLL_ELEM_MULTI Dot_close
6399                       | Open_dot META_CHAR Dot_close
6400                       ;
6401     equivalence_class : Open_equal COLL_ELEM_SINGLE Equal_close
6402                       | Open_equal COLL_ELEM_MULTI Equal_close
6403                       ;
6404     character_class   : Open_colon class_name Colon_close
6405                       ;

```

6406 The BRE grammar does not permit L\_ANCHOR or R\_ANCHOR inside "\(" and "\)" (which  
6407 implies that '^' and '\$' are ordinary characters). This reflects the semantic limits on the  
6408 application, as noted in Section 9.3.8 (on page 171). Implementations are permitted to extend the  
6409 language to interpret '^' and '\$' as anchors in these locations, and as such, conforming |  
6410 applications cannot use unescaped '^' and '\$' in positions inside "\(" and "\)" that might |  
6411 be interpreted as anchors.

### 6412 9.5.3 ERE Grammar

6413 This section presents the grammar for extended regular expressions, excluding the bracket  
6414 expression grammar.

6415 **Note:** The bracket expression grammar and the associated %token lines are identical between BREs  
6416 and EREs. It has been omitted from the ERE section to avoid unnecessary editorial duplication.

```

6417 %token  ORD_CHAR QUOTED_CHAR DUP_COUNT
6418 %start  extended_reg_exp
6419 %%
6420 /* -----
6421    Extended Regular Expression
6422    -----
6423 */
6424 extended_reg_exp  :                               ERE_branch
6425                   | extended_reg_exp '|' ERE_branch
6426                   ;
6427 ERE_branch       :                               ERE_expression
6428                   | ERE_branch ERE_expression
6429                   ;
6430 ERE_expression   : one_char_or_coll_elem_ERE
6431                   | '^'
6432                   | '$'
6433                   | '(' extended_reg_exp ')'
6434                   | ERE_expression ERE_dupl_symbol
6435                   ;
6436 one_char_or_coll_elem_ERE : ORD_CHAR
6437                           | QUOTED_CHAR
6438                           | '.'
6439                           | bracket_expression
6440                           ;

```

```

6441     ERE_dupl_symbol      : '*'
6442                          | '+'
6443                          | '?'
6444                          | '{' DUP_COUNT '}'
6445                          | '{' DUP_COUNT ', ' '}'
6446                          | '{' DUP_COUNT ', ' DUP_COUNT '}'
6447                          ;

```

6448 The ERE grammar does not permit several constructs that previous sections specify as having  
 6449 undefined results:

- 6450 • ORD\_CHAR preceded by '\'
- 6451 • One or more *ERE\_dupl\_symbols* appearing first in an ERE, or immediately following '|',  
 6452 '^', or '('
- 6453 • '{' not part of a valid *ERE\_dupl\_symbol*
- 6454 • '|' appearing first or last in an ERE, or immediately following '|' or '(', or immediately  
 6455 preceding ')'

6456 Implementations are permitted to extend the language to allow these. Conforming applications |  
 6457 cannot use such constructs. |



# Directory Structure and Devices

6459

## 6460 10.1 Directory Structure and Files

6461 The following directories shall exist on conforming systems and conforming applications shall |  
 6462 make use of them only as described. Strictly conforming applications shall not assume the |  
 6463 ability to create files in any of these directories, unless specified below.

6464 / The root directory.

6465 /dev Contains /dev/console, /dev/null, and /dev/tty, described below.

6466 The following directory shall exist on conforming systems and shall be used as described.

6467 /tmp A directory made available for programs that need a place to create temporary |  
 6468 files. Applications shall be allowed to create files in this directory, but shall not |  
 6469 assume that such files are preserved between invocations of the application.

6470 The following files shall exist on conforming systems and shall be both readable and writable.

6471 /dev/null An infinite data source and data sink. Data written to /dev/null shall be discarded.  
 6472 Reads from /dev/null shall always return end-of-file (EOF).

6473 /dev/tty In each process, a synonym for the controlling terminal associated with the process  
 6474 group of that process, if any. It is useful for programs or shell procedures that wish  
 6475 to be sure of writing messages to or reading data from the terminal no matter how  
 6476 output has been redirected. It can also be used for programs that demand the name  
 6477 of a file for output, when typed output is desired and it is tiresome to find out  
 6478 what terminal is currently in use.

6479 The following file shall exist on conforming systems and need not be readable or writable:

6480 /dev/console The /dev/console file is a generic name given to the system console (see Section |  
 6481 3.382 (on page 85)). It is usually linked to an implementation-defined special file. It |  
 6482 shall provide an interface to the system console conforming to the requirements of |  
 6483 the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal |  
 6484 Interface. |

## 6485 10.2 Output Devices and Terminal Types

6486 The utilities in the Shell and Utilities volume of IEEE Std 1003.1-200x historically have been  
 6487 implemented on a wide range of terminal types, but a conforming implementation need not  
 6488 support all features of all utilities on every conceivable terminal. IEEE Std 1003.1-200x states  
 6489 which features are optional for certain classes of terminals in the individual utility description  
 6490 sections. The implementation shall document which terminal types it supports and which of  
 6491 these features and utilities are not supported by each terminal.

6492 When a feature or utility is not supported on a specific terminal type, as allowed by  
 6493 IEEE Std 1003.1-200x, and the implementation considers such a condition to be an error  
 6494 preventing use of the feature or utility, the implementation shall indicate such conditions  
 6495 through diagnostic messages or exit status values or both (as appropriate to the specific utility  
 6496 description) that inform the user that the terminal type lacks the appropriate capability.

6497 IEEE Std 1003.1-200x uses a notational convention based on historical practice that identifies  
 6498 some of the control characters defined in Section 7.3.1 (on page 122) in a manner easily  
 6499 remembered by users on many terminals. The correspondence between this “<control>-char”  
 6500 notation and the actual control characters is shown in the following table. When  
 6501 IEEE Std 1003.1-200x refers to a character by its <control>- name, it is referring to the actual  
 6502 control character shown in the Value column of the table, which is not necessarily the exact  
 6503 control key sequence on all terminals. Some terminals have keyboards that do not allow the  
 6504 direct transmission of all the non-alphanumeric characters shown. In such cases, the system  
 6505 documentation shall describe which data sequences transmitted by the terminal are interpreted  
 6506 by the system as representing the special characters.

6507 **Table 10-1** Control Character Names

Name	Value	Symbolic Name	Name	Value	Symbolic Name
<control>-A	<SOH>	<SOH>	<control>-Q	<DC1>	<DC1>
<control>-B	<STX>	<STX>	<control>-R	<DC2>	<DC2>
<control>-C	<ETX>	<ETX>	<control>-S	<DC3>	<DC3>
<control>-D	<EOT>	<EOT>	<control>-T	<DC4>	<DC4>
<control>-E	<ENQ>	<ENQ>	<control>-U	<NAK>	<NAK>
<control>-F	<ACK>	<ACK>	<control>-V	<SYN>	<SYN>
<control>-G	<BEL>	<alert>	<control>-W	<ETB>	<ETB>
<control>-H	<BS>	<backspace>	<control>-X	<CAN>	<CAN>
<control>-I	<HT>	<tab>	<control>-Y	<EM>	<EM>
<control>-J	<LF>	<linefeed>	<control>-Z	<SUB>	<SUB>
<control>-K	<VT>	<vertical-tab>	<control>-[	<ESC>	<ESC>
<control>-L	<FF>	<form-feed>	<control>-\ <control>-]	<FS>	<FS>
<control>-M	<CR>	<carriage-return>	<control>-^	<GS>	<GS>
<control>-N	<SO>	<SO>	<control>-_	<RS>	<RS>
<control>-O	<SI>	<SI>	<control>-?	<US>	<US>
<control>-P	<DLE>	<DLE>		<DEL>	<DEL>

6525 **Note:** The notation uses uppercase letters for arbitrary editorial reasons. There is no implication that  
 6526 the keystrokes represent control-shift-letter sequences.

# General Terminal Interface

6527

6528 This chapter describes a general terminal interface that shall be provided. It shall be supported  
6529 on any asynchronous communications ports if the implementation provides them. It is  
6530 implementation-defined whether it supports network connections or synchronous ports, or  
6531 both.

## 6532 11.1 Interface Characteristics

### 6533 11.1.1 Opening a Terminal Device File

6534 When a terminal device file is opened, it normally causes the thread to wait until a connection is  
6535 established. In practice, application programs seldom open these files; they are opened by  
6536 special programs and become an application's standard input, output, and error files.

6537 As described in *open()*, opening a terminal device file with the `O_NONBLOCK` flag clear shall  
6538 cause the thread to block until the terminal device is ready and available. If `CLOCAL` mode is  
6539 not set, this means blocking until a connection is established. If `CLOCAL` mode is set in the  
6540 terminal, or the `O_NONBLOCK` flag is specified in the *open()*, the *open()* function shall return a  
6541 file descriptor without waiting for a connection to be established.

### 6542 11.1.2 Process Groups

6543 A terminal may have a foreground process group associated with it. This foreground process  
6544 group plays a special role in handling signal-generating input characters, as discussed in Section  
6545 11.1.9 (on page 187).

6546 A command interpreter process supporting job control can allocate the terminal to different jobs,  
6547 or process groups, by placing related processes in a single process group and associating this  
6548 process group with the terminal. A terminal's foreground process group may be set or examined  
6549 by a process, assuming the permission requirements are met; see *tcgetpgrp()* and *tcsetpgrp()*. The  
6550 terminal interface aids in this allocation by restricting access to the terminal by processes that are  
6551 not in the current process group; see Section 11.1.4 (on page 184).

6552 When there is no longer any process whose process ID or process group ID matches the process  
6553 group ID of the foreground process group, the terminal shall have no foreground process group.  
6554 It is unspecified whether the terminal has a foreground process group when there is a process  
6555 whose process ID matches the foreground process ID, but whose process group ID does not. No  
6556 actions defined in IEEE Std 1003.1-200x, other than allocation of a controlling terminal or a  
6557 successful call to *tcsetpgrp()*, cause a process group to become the foreground process group of  
6558 the terminal.

### 6559 11.1.3 The Controlling Terminal

6560 A terminal may belong to a process as its controlling terminal. Each process of a session that has  
6561 a controlling terminal has the same controlling terminal. A terminal may be the controlling  
6562 terminal for at most one session. The controlling terminal for a session is allocated by the session  
6563 leader in an implementation-defined manner. If a session leader has no controlling terminal, and  
6564 opens a terminal device file that is not already associated with a session without using the  
6565 O\_NOCTTY option (see *open()*), it is implementation-defined whether the terminal becomes the  
6566 controlling terminal of the session leader. If a process which is not a session leader opens a  
6567 terminal file, or the O\_NOCTTY option is used on *open()*, then that terminal shall not become  
6568 the controlling terminal of the calling process. When a controlling terminal becomes associated  
6569 with a session, its foreground process group shall be set to the process group of the session  
6570 leader.

6571 The controlling terminal is inherited by a child process during a *fork()* function call. A process  
6572 relinquishes its controlling terminal when it creates a new session with the *setsid()* function;  
6573 other processes remaining in the old session that had this terminal as their controlling terminal  
6574 continue to have it. Upon the close of the last file descriptor in the system (whether or not it is in  
6575 the current session) associated with the controlling terminal, it is unspecified whether all  
6576 processes that had that terminal as their controlling terminal cease to have any controlling  
6577 terminal. Whether and how a session leader can reacquire a controlling terminal after the  
6578 controlling terminal has been relinquished in this fashion is unspecified. A process does not  
6579 relinquish its controlling terminal simply by closing all of its file descriptors associated with the  
6580 controlling terminal if other processes continue to have it open.

6581 When a controlling process terminates, the controlling terminal is dissociated from the current  
6582 session, allowing it to be acquired by a new session leader. Subsequent access to the terminal by  
6583 other processes in the earlier session may be denied, with attempts to access the terminal treated  
6584 as if a modem disconnect had been sensed.

### 6585 11.1.4 Terminal Access Control

6586 If a process is in the foreground process group of its controlling terminal, read operations shall  
6587 be allowed, as described in Section 11.1.5 (on page 185). Any attempts by a process in a  
6588 background process group to read from its controlling terminal cause its process group to be  
6589 sent a SIGTTIN signal unless one of the following special cases applies: if the reading process is  
6590 ignoring or blocking the SIGTTIN signal, or if the process group of the reading process is  
6591 orphaned, the *read()* shall return  $-1$ , with *errno* set to [EIO] and no signal shall be sent. The  
6592 default action of the SIGTTIN signal shall be to stop the process to which it is sent. See  
6593 **<signal.h>**.

6594 If a process is in the foreground process group of its controlling terminal, write operations shall  
6595 be allowed as described in Section 11.1.8 (on page 187). Attempts by a process in a background  
6596 process group to write to its controlling terminal shall cause the process group to be sent a  
6597 SIGTTOU signal unless one of the following special cases applies: if TOSTOP is not set, or if  
6598 TOSTOP is set and the process is ignoring or blocking the SIGTTOU signal, the process is  
6599 allowed to write to the terminal and the SIGTTOU signal is not sent. If TOSTOP is set, and the  
6600 process group of the writing process is orphaned, and the writing process is not ignoring or  
6601 blocking the SIGTTOU signal, the *write()* shall return  $-1$ , with *errno* set to [EIO] and no signal  
6602 shall be sent.

6603 Certain calls that set terminal parameters are treated in the same fashion as *write()*, except that  
6604 TOSTOP is ignored; that is, the effect is identical to that of terminal writes when TOSTOP is set  
6605 (see Section 11.2.5 (on page 193), *tcdrain()*, *tcfLOW()*, *tcfLush()*, *tcsendbreak()*, *tcsetattr()*, and  
6606 *tcsetpgrp()*).

### 6607 11.1.5 Input Processing and Reading Data

6608 A terminal device associated with a terminal device file may operate in full-duplex mode, so that  
 6609 data may arrive even while output is occurring. Each terminal device file has an *input queue*,  
 6610 associated with it, into which incoming data is stored by the system before being read by a  
 6611 process. The system may impose a limit, {MAX\_INPUT}, on the number of bytes that may be  
 6612 stored in the input queue. The behavior of the system when this limit is exceeded is  
 6613 implementation-defined.

6614 Two general kinds of input processing are available, determined by whether the terminal device  
 6615 file is in canonical mode or non-canonical mode. These modes are described in Section 11.1.6 and  
 6616 Section 11.1.7 (on page 186). Additionally, input characters are processed according to the  
 6617 *c\_iflag* (see Section 11.2.2 (on page 189)) and *c\_lflag* (see Section 11.2.5 (on page 193)) fields.  
 6618 Such processing can include *echoing*, which in general means transmitting input characters  
 6619 immediately back to the terminal when they are received from the terminal. This is useful for  
 6620 terminals that can operate in full-duplex mode.

6621 The manner in which data is provided to a process reading from a terminal device file is  
 6622 dependent on whether the terminal file is in canonical or non-canonical mode, and on whether  
 6623 or not the O\_NONBLOCK flag is set by *open()* or *fcntl()*.

6624 If the O\_NONBLOCK flag is clear, then the read request shall be blocked until data is available  
 6625 or a signal has been received. If the O\_NONBLOCK flag is set, then the read request shall be  
 6626 completed, without blocking, in one of three ways:

- 6627 1. If there is enough data available to satisfy the entire request, the *read()* shall complete  
 6628 successfully and shall return the number of bytes read.
- 6629 2. If there is not enough data available to satisfy the entire request, the *read()* shall complete  
 6630 successfully, having read as much data as possible, and shall return the number of bytes it  
 6631 was able to read.
- 6632 3. If there is no data available, the *read()* shall return  $-1$ , with *errno* set to [EAGAIN].

6633 When data is available depends on whether the input processing mode is canonical or non-  
 6634 canonical. The following sections, Section 11.1.6 and Section 11.1.7 (on page 186), describe each  
 6635 of these input processing modes.

### 6636 11.1.6 Canonical Mode Input Processing

6637 In canonical mode input processing, terminal input is processed in units of lines. A line is  
 6638 delimited by a newline character (NL), an end-of-file character (EOF), or an end-of-line (EOL)  
 6639 character. See Section 11.1.9 (on page 187) for more information on EOF and EOL. This means  
 6640 that a read request shall not return until an entire line has been typed or a signal has been  
 6641 received. Also, no matter how many bytes are requested in the *read()* call, at most one line shall  
 6642 be returned. It is not, however, necessary to read a whole line at once; any number of bytes, even  
 6643 one, may be requested in a *read()* without losing information.

6644 If {MAX\_CANON} is defined for this terminal device, it shall be a limit on the number of bytes |  
 6645 in a line. The behavior of the system when this limit is exceeded is implementation-defined. If |  
 6646 {MAX\_CANON} is not defined, there shall be no such limit; see *pathconf()*. |

6647 Erase and kill processing occur when either of two special characters, the ERASE and KILL |  
 6648 characters (see Section 11.1.9 (on page 187)), is received. This processing shall affect data in the |  
 6649 input queue that has not yet been delimited by a newline (NL), EOF, or EOL character. This un- |  
 6650 delimited data makes up the current line. The ERASE character shall delete the last character in |  
 6651 the current line, if there is one. The KILL character shall delete all data in the current line, if there |  
 6652 are any. The ERASE and KILL characters shall have no effect if there is no data in the current |

6653 line. The ERASE and KILL characters themselves shall not be placed in the input queue. |

### 6654 11.1.7 Non-Canonical Mode Input Processing

6655 In non-canonical mode input processing, input bytes are not assembled into lines, and erase and |  
6656 kill processing shall not occur. The values of the MIN and TIME members of the `c_cc` array are |  
6657 used to determine how to process the bytes received. The IEEE Std 1003.1-200x does not specify |  
6658 whether the setting of `O_NONBLOCK` takes precedence over MIN or TIME settings. Therefore, |  
6659 if `O_NONBLOCK` is set, `read()` may return immediately, regardless of the setting of MIN or |  
6660 TIME. Also, if no data is available, `read()` may either return 0, or return `-1` with `errno` set to |  
6661 `[EAGAIN]`.

6662 MIN represents the minimum number of bytes that should be received when the `read()` function |  
6663 returns successfully. TIME is a timer of 0.1 second granularity that is used to time out bursty and |  
6664 short-term data transmissions. If MIN is greater than `{MAX_INPUT}`, the response to the request |  
6665 is undefined. The four possible values for MIN and TIME and their interactions are described |  
6666 below.

#### 6667 Case A: MIN>0, TIME>0

6668 In case A, TIME serves as an inter-byte timer which shall be activated after the first byte is |  
6669 received. Since it is an inter-byte timer, it shall be reset after a byte is received. The interaction |  
6670 between MIN and TIME is as follows. As soon as one byte is received, the inter-byte timer shall |  
6671 be started. If MIN bytes are received before the inter-byte timer expires (remember that the timer |  
6672 is reset upon receipt of each byte), the read shall be satisfied. If the timer expires before MIN |  
6673 bytes are received, the characters received to that point shall be returned to the user. Note that if |  
6674 TIME expires at least one byte shall be returned because the timer would not have been enabled |  
6675 unless a byte was received. In this case (MIN>0, TIME>0) the read shall block until the MIN and |  
6676 TIME mechanisms are activated by the receipt of the first byte, or a signal is received. If data is in |  
6677 the buffer at the time of the `read()`, the result shall be as if data has been received immediately |  
6678 after the `read()`.

#### 6679 Case B: MIN>0, TIME=0

6680 In case B, since the value of TIME is zero, the timer plays no role and only MIN is significant. A |  
6681 pending read shall not be satisfied until MIN bytes are received (that is, the pending read shall |  
6682 block until MIN bytes are received), or a signal is received. A program that uses case B to read |  
6683 record-based terminal I/O may block indefinitely in the read operation.

#### 6684 Case C: MIN=0, TIME>0

6685 In case C, since MIN=0, TIME no longer represents an inter-byte timer. It now serves as a read |  
6686 timer that shall be activated as soon as the `read()` function is processed. A read shall be satisfied |  
6687 as soon as a single byte is received or the read timer expires. Note that in case C if the timer |  
6688 expires, no bytes shall be returned. If the timer does not expire, the only way the read can be |  
6689 satisfied is if a byte is received. If bytes are not received, the read shall not block indefinitely |  
6690 waiting for a byte; if no byte is received within `TIME*0.1` seconds after the read is initiated, the |  
6691 `read()` shall return a value of zero, having read no data. If data is in the buffer at the time of the |  
6692 `read()`, the timer shall be started as if data has been received immediately after the `read()`.

6693 **Case D: MIN=0, TIME=0**

6694 The minimum of either the number of bytes requested or the number of bytes currently available  
 6695 shall be returned without waiting for more bytes to be input. If no characters are available, *read()*  
 6696 shall return a value of zero, having read no data.

6697 **11.1.8 Writing Data and Output Processing**

6698 When a process writes one or more bytes to a terminal device file, they are processed according  
 6699 to the *c\_oflag* field (see Section 11.2.3 (on page 190)). The implementation may provide a  
 6700 buffering mechanism; as such, when a call to *write()* completes, all of the bytes written have  
 6701 been scheduled for transmission to the device, but the transmission has not necessarily  
 6702 completed. See *write()* for the effects of *O\_NONBLOCK* on *write()*.

6703 **11.1.9 Special Characters**

6704 Certain characters have special functions on input or output or both. These functions are  
 6705 summarized as follows:

6706 **INTR** Special character on input, which is recognized if the *ISIG* flag is set. Generates a  
 6707 *SIGINT* signal which is sent to all processes in the foreground process group for which  
 6708 the terminal is the controlling terminal. If *ISIG* is set, the *INTR* character shall be  
 6709 discarded when processed.

6710 **QUIT** Special character on input, which is recognized if the *ISIG* flag is set. Generates a  
 6711 *SIGQUIT* signal which is sent to all processes in the foreground process group for  
 6712 which the terminal is the controlling terminal. If *ISIG* is set, the *QUIT* character shall be  
 6713 discarded when processed.

6714 **ERASE** Special character on input, which is recognized if the *ICANON* flag is set. Erases the  
 6715 last character in the current line; see Section 11.1.6 (on page 185). It shall not erase  
 6716 beyond the start of a line, as delimited by an *NL*, *EOF*, or *EOL* character. If *ICANON* is  
 6717 set, the *ERASE* character shall be discarded when processed.

6718 **KILL** Special character on input, which is recognized if the *ICANON* flag is set. Deletes the  
 6719 entire line, as delimited by an *NL*, *EOF*, or *EOL* character. If *ICANON* is set, the *KILL*  
 6720 character shall be discarded when processed.

6721 **EOF** Special character on input, which is recognized if the *ICANON* flag is set. When  
 6722 received, all the bytes waiting to be read are immediately passed to the process without  
 6723 waiting for a newline, and the *EOF* is discarded. Thus, if there are no bytes waiting  
 6724 (that is, the *EOF* occurred at the beginning of a line), a byte count of zero shall be  
 6725 returned from the *read()*, representing an end-of-file indication. If *ICANON* is set, the  
 6726 *EOF* character shall be discarded when processed.

6727 **NL** Special character on input, which is recognized if the *ICANON* flag is set. It is the line  
 6728 delimiter newline. It cannot be changed.

6729 **EOL** Special character on input, which is recognized if the *ICANON* flag is set. It is an  
 6730 additional line delimiter, like *NL*.

6731 **SUSP** If the *ISIG* flag is set, receipt of the *SUSP* character shall cause a *SIGTSTP* signal to be  
 6732 sent to all processes in the foreground process group for which the terminal is the  
 6733 controlling terminal, and the *SUSP* character shall be discarded when processed.

6734 **STOP** Special character on both input and output, which is recognized if the *IXON* (output  
 6735 control) or *IXOFF* (input control) flag is set. Can be used to suspend output  
 6736 temporarily. It is useful with CRT terminals to prevent output from disappearing

6737 before it can be read. If IXON is set, the STOP character shall be discarded when  
6738 processed.

6739 **START** Special character on both input and output, which is recognized if the IXON (output  
6740 control) or IXOFF (input control) flag is set. Can be used to resume output that has  
6741 been suspended by a STOP character. If IXON is set, the START character shall be  
6742 discarded when processed.

6743 **CR** Special character on input, which is recognized if the ICANON flag is set; it is the  
6744 carriage-return character. When ICANON and ICRNL are set and IGNCR is not set,  
6745 this character shall be translated into an NL, and shall have the same effect as an NL  
6746 character.

6747 The NL and CR characters cannot be changed. It is implementation-defined whether the START  
6748 and STOP characters can be changed. The values for INTR, QUIT, ERASE, KILL, EOF, EOL, and  
6749 SUSP shall be changeable to suit individual tastes. Special character functions associated with  
6750 changeable special control characters can be disabled individually.

6751 If two or more special characters have the same value, the function performed when that  
6752 character is received is undefined.

6753 A special character is recognized not only by its value, but also by its context; for example, an  
6754 implementation may support multi-byte sequences that have a meaning different from the  
6755 meaning of the bytes when considered individually. Implementations may also support  
6756 additional single-byte functions. These implementation-defined multi-byte or single-byte  
6757 functions shall be recognized only if the IEXTEN flag is set; otherwise, data is received without  
6758 interpretation, except as required to recognize the special characters defined in this section.

6759 **XSI** If IEXTEN is set, the ERASE, KILL, and EOF characters can be escaped by a preceding '\'  
6760 character, in which case no special function shall occur.

#### 6761 **11.1.10 Modem Disconnect**

6762 If a modem disconnect is detected by the terminal interface for a controlling terminal, and if  
6763 CLOCAL is not set in the **c\_cflag** field for the terminal (see Section 11.2.4 (on page 192)), the  
6764 SIGHUP signal shall be sent to the controlling process for which the terminal is the controlling  
6765 terminal. Unless other arrangements have been made, this shall cause the controlling process to  
6766 terminate (see *exit()*). Any subsequent read from the terminal device shall return the value of  
6767 zero, indicating end-of-file; see *read()*. Thus, processes that read a terminal file and test for end-  
6768 of-file can terminate appropriately after a disconnect. If the EIO condition as specified in *read()*  
6769 also exists, it is unspecified whether on EOF condition or the [EIO] is returned. Any subsequent  
6770 *write()* to the terminal device shall return -1, with *errno* set to [EIO], until the device is closed.

#### 6771 **11.1.11 Closing a Terminal Device File**

6772 The last process to close a terminal device file shall cause any output to be sent to the device and  
6773 any input to be discarded. If HUPCL is set in the control modes and the communications port  
6774 supports a disconnect function, the terminal device shall perform a disconnect.

6775 **11.2 Parameters that Can be Set**6776 **11.2.1 The termios Structure**

6777 Routines that need to control certain terminal I/O characteristics shall do so by using the  
6778 **termios** structure as defined in the `<termios.h>` header. The members of this structure include  
6779 (but are not limited to):

Member Type	Array Size	Member Name	Description
6780 <code>tcflag_t</code>		<code>c_iflag</code>	Input modes.
6781 <code>tcflag_t</code>		<code>c_oflag</code>	Output modes.
6782 <code>tcflag_t</code>		<code>c_cflag</code>	Control modes.
6783 <code>tcflag_t</code>		<code>c_lflag</code>	Local modes.
6784 <code>cc_t</code>	NCCS	<code>c_cc[]</code>	Control characters.

6787 The types `tcflag_t` and `cc_t` are defined in the `<termios.h>` header. They shall be unsigned  
6788 integer types.

6789 **11.2.2 Input Modes**

6790 Values of the `c_iflag` field describe the basic terminal input control, and are composed of the  
6791 bitwise-inclusive OR of the masks shown, which shall be bitwise-distinct. The mask name  
6792 symbols in this table are defined in `<termios.h>`:

Mask Name	Description
6793 BRKINT	Signal interrupt on break.
6794 ICRNL	Map CR to NL on input.
6795 IGNBRK	Ignore break condition.
6796 IGNCR	Ignore CR.
6797 IGNPAR	Ignore characters with parity errors.
6800 INLCR	Map NL to CR on input.
6801 INPCK	Enable input parity check.
6802 ISTRIP	Strip character.
6803 XSI <code>IXANY</code>	Enable any character to restart output.
6804 <code>IXOFF</code>	Enable start/stop input control.
6805 <code>IXON</code>	Enable start/stop output control.
6806 <code>PARMRK</code>	Mark parity errors.

6807 In the context of asynchronous serial data transmission, a break condition shall be defined as a  
6808 sequence of zero-valued bits that continues for more than the time to send one byte. The entire  
6809 sequence of zero-valued bits is interpreted as a single break condition, even if it continues for a  
6810 time equivalent to more than one byte. In contexts other than asynchronous serial data  
6811 transmission, the definition of a break condition is implementation-defined.

6812 If `IGNBRK` is set, a break condition detected on input shall be ignored; that is, not put on the  
6813 input queue and therefore not read by any process. If `IGNBRK` is not set and `BRKINT` is set, the  
6814 break condition shall flush the input and output queues, and if the terminal is the controlling  
6815 terminal of a foreground process group, the break condition shall generate a single `SIGINT`  
6816 signal to that foreground process group. If neither `IGNBRK` nor `BRKINT` is set, a break  
6817 condition shall be read as a single `0x00`, or if `PARMRK` is set, as `0xff 0x00 0x00`.

6818 If `IGNPAR` is set, a byte with a framing or parity error (other than break) shall be ignored.

6819 If PARMRK is set, and IGNPAR is not set, a byte with a framing or parity error (other than |  
6820 break) shall be given to the application as the three-byte sequence 0xff 0x00 X, where 0xff 0x00 is |  
6821 a two-byte flag preceding each sequence and X is the data of the byte received in error. To avoid |  
6822 ambiguity in this case, if ISTRIP is not set, a valid byte of 0xff is given to the application as 0xff |  
6823 0xff. If neither PARMRK nor IGNPAR is set, a framing or parity error (other than break) shall be |  
6824 given to the application as a single byte 0x00. |

6825 If INPCK is set, input parity checking shall be enabled. If INPCK is not set, input parity checking  
6826 shall be disabled, allowing output parity generation without input parity errors. Note that  
6827 whether input parity checking is enabled or disabled is independent of whether parity detection  
6828 is enabled or disabled (see Section 11.2.4 (on page 192)). If parity detection is enabled but input  
6829 parity checking is disabled, the hardware to which the terminal is connected shall recognize the  
6830 parity bit, but the terminal special file shall not check whether or not this bit is correctly set.

6831 If ISTRIP is set, valid input bytes shall first be stripped to seven bits; otherwise, all eight bits  
6832 shall be processed.

6833 If INLCR is set, a received NL character shall be translated into a CR character. If IGNCR is set, a  
6834 received CR character shall be ignored (not read). If IGNCR is not set and ICRNL is set, a  
6835 received CR character shall be translated into an NL character.

6836 XSI If IXANY is set, any input character shall restart output that has been suspended.

6837 If IXON is set, start/stop output control shall be enabled. A received STOP character shall  
6838 suspend output and a received START character shall restart output. When IXON is set, START  
6839 and STOP characters are not read, but merely perform flow control functions. When IXON is not  
6840 set, the START and STOP characters shall be read.

6841 If IXOFF is set, start/stop input control shall be enabled. The system shall transmit STOP  
6842 characters, which are intended to cause the terminal device to stop transmitting data, as needed  
6843 to prevent the input queue from overflowing and causing implementation-defined behavior, and  
6844 shall transmit START characters, which are intended to cause the terminal device to resume  
6845 transmitting data, as soon as the device can continue transmitting data without risk of  
6846 overflowing the input queue. The precise conditions under which STOP and START characters  
6847 are transmitted are implementation-defined.

6848 The initial input control value after *open()* is implementation-defined.

### 6849 11.2.3 Output Modes

6850 The **c\_oflag** field specifies the terminal interface's treatment of output, and is composed of the  
6851 bitwise-inclusive OR of the masks shown, which shall be bitwise-distinct. The mask name  
6852 symbols in this table are defined in `<termios.h>`:

6853  
6854  
6855  
6856  
6857  
6858  
6859  
6860  
6861  
6862  
6863  
6864  
6865  
6866  
6867  
6868  
6869  
6870  
6871  
6872  
6873  
6874  
6875  
6876  
6877  
6878  
6879  
6880  
6881  
6882  
6883

Mask Name	Description
OPOST	Perform output processing.
ONLCR	Map NL to CR-NL on output.
OCRNL	Map CR to NL on output.
ONOCR	No CR output at column 0.
ONLRET	NL performs CR function.
OFILL	Use fill characters for delay.
OFDEL	Fill is DEL, else NUL.
NLDLY	Select newline delays:
NL0	Newline character type 0.
NL1	Newline character type 1.
CRDLY	Select carriage-return delays:
CR0	Carriage-return delay type 0.
CR1	Carriage-return delay type 1.
CR2	Carriage-return delay type 2.
CR3	Carriage-return delay type 3.
TABDLY	Select horizontal-tab delays:
TAB0	Horizontal-tab delay type 0.
TAB1	Horizontal-tab delay type 1.
TAB2	Horizontal-tab delay type 2.
TAB3	Expand tabs to spaces.
BSDLY	Select backspace delays:
BS0	Backspace-delay type 0.
BS1	Backspace-delay type 1.
VTDLY	Select vertical-tab delays:
VT0	Vertical-tab delay type 0.
VT1	Vertical-tab delay type 1.
FFDLY	Select form-feed delays:
FF0	Form-feed delay type 0.
FF1	Form-feed delay type 1.

6884  
6885  
6886  
  
6887  
6888  
6889  
6890  
6891  
6892  
6893  
  
6894  
6895  
6896  
6897  
6898  
  
6899  
  
6900  
6901

If OPOST is set, output data shall be post-processed as described below, so that lines of text are modified to appear appropriately on the terminal device; otherwise, characters shall be transmitted without change.

XSI

If ONLCR is set, the NL character shall be transmitted as the CR-NL character pair. If OCRNL is set, the CR character shall be transmitted as the NL character. If ONOCR is set, no CR character shall be transmitted when at column 0 (first position). If ONLRET is set, the NL character is assumed to do the carriage-return function; the column pointer shall be set to 0 and the delays specified for CR shall be used. Otherwise, the NL character is assumed to do just the line-feed function; the column pointer remains unchanged. The column pointer shall also be set to 0 if the CR character is actually transmitted.

The delay bits specify how long transmission stops to allow for mechanical or other movement when certain characters are sent to the terminal. In all cases a value of 0 shall indicate no delay. If OFILL is set, fill characters shall be transmitted for delay instead of a timed delay. This is useful for high baud rate terminals which need only a minimal delay. If OFDEL is set, the fill character shall be DEL; otherwise, NUL.

If a form-feed or vertical-tab delay is specified, it shall last for about 2 seconds.

New-line delay shall last about 0.10 seconds. If ONLRET is set, the carriage-return delays shall be used instead of the newline delays. If OFILL is set, two fill characters shall be transmitted.

6902 Carriage-return delay type 1 shall be dependent on the current column position, type 2 shall be  
 6903 about 0.10 seconds, and type 3 shall be about 0.15 seconds. If OFILL is set, delay type 1 shall  
 6904 transmit two fill characters, and type 2, four fill characters.

6905 Horizontal-tab delay type 1 shall be dependent on the current column position. Type 2 shall be  
 6906 about 0.10 seconds. Type 3 specifies that tabs shall be expanded into spaces. If OFILL is set, two  
 6907 fill characters shall be transmitted for any delay.

6908 Backspace delay shall last about 0.05 seconds. If OFILL is set, one fill character shall be  
 6909 transmitted.

6910 The actual delays depend on line speed and system load.

6911 The initial output control value after *open()* is implementation-defined.

#### 6912 11.2.4 Control Modes

6913 The **c\_cflag** field describes the hardware control of the terminal, and is composed of the  
 6914 bitwise-inclusive OR of the masks shown, which shall be bitwise-distinct. The mask name  
 6915 symbols in this table are defined in **<termios.h>**; not all values specified are required to be  
 6916 supported by the underlying hardware:

Mask Name	Description
CLOCAL	Ignore modem status lines.
CREAD	Enable receiver.
CSIZE	Number of bits transmitted or received per byte:
CS5	5 bits
CS6	6 bits
CS7	7 bits
CS8	8 bits.
CSTOPB	Send two stop bits, else one.
HUPCL	Hang up on last close.
PARENB	Parity enable.
PARODD	Odd parity, else even.

6929 In addition, the input and output baud rates are stored in the **termios** structure. The symbols in  
 6930 the following table are defined in **<termios.h>**. Not all values specified are required to be  
 6931 supported by the underlying hardware.

Name	Description	Name	Description
B0	Hang up	B600	600 baud
B50	50 baud	B1200	1200 baud
B75	75 baud	B1800	1800 baud
B110	110 baud	B2400	2400 baud
B134	134.5 baud	B4800	4800 baud
B150	150 baud	B9600	9600 baud
B200	200 baud	B19200	19200 baud
B300	300 baud	B38400	38400 baud

6941 The following functions are provided for getting and setting the values of the input and output  
 6942 baud rates in the **termios** structure: *cfgetispeed()*, *cfgetospeed()*, *cfsetispeed()*, and *cfsetospeed()*.  
 6943 The effects on the terminal device shall not become effective and not all errors need be detected  
 6944 until the *tcsetattr()* function is successfully called.

6945 The CSIZE bits shall specify the number of transmitted or received bits per byte. If ISTRIP is not  
 6946 set, the value of all the other bits is unspecified. If ISTRIP is set, the value of all but the 7 low-

6947 order bits shall be zero, but the value of any other bits beyond CSIZE is unspecified when read. |  
 6948 CSIZE shall not include the parity bit, if any. If CSTOPB is set, two stop bits shall be used; |  
 6949 otherwise, one stop bit. For example, at 110 baud, two stop bits are normally used. |

6950 If CREAD is set, the receiver shall be enabled; otherwise, no characters shall be received. |

6951 If PARENB is set, parity generation and detection shall be enabled and a parity bit is added to |  
 6952 each byte. If parity is enabled, PARODD shall specify odd parity if set; otherwise, even parity |  
 6953 shall be used. |

6954 If HUPCL is set, the modem control lines for the port shall be lowered when the last process |  
 6955 with the port open closes the port or the process terminates. The modem connection shall be |  
 6956 broken. |

6957 If CLOCAL is set, a connection shall not depend on the state of the modem status lines. If |  
 6958 CLOCAL is clear, the modem status lines shall be monitored. |

6959 Under normal circumstances, a call to the *open()* function shall wait for the modem connection |  
 6960 to complete. However, if the O\_NONBLOCK flag is set (see *open()*) or if CLOCAL has been set, |  
 6961 the *open()* function shall return immediately without waiting for the connection. |

6962 If the object for which the control modes are set is not an asynchronous serial connection, some |  
 6963 of the modes may be ignored; for example, if an attempt is made to set the baud rate on a |  
 6964 network connection to a terminal on another host, the baud rate need not be set on the |  
 6965 connection between that terminal and the machine to which it is directly connected. |

6966 The initial hardware control value after *open()* is implementation-defined. |

### 6967 11.2.5 Local Modes

6968 The *c\_lflag* field of the argument structure is used to control various functions. It is composed |  
 6969 of the bitwise-inclusive OR of the masks shown, which shall be bitwise-distinct. The mask name |  
 6970 symbols in this table are defined in `<termios.h>`; not all values specified are required to be |  
 6971 supported by the underlying hardware: |

6972

6973

Mask Name	Description
ECHO	Enable echo.
ECHOE	Echo ERASE as an error correcting backspace.
ECHOK	Echo KILL.
ECHONL	Echo <newline>.
ICANON	Canonical input (erase and kill processing).
IEXTEN	Enable extended (implementation-defined) functions.
ISIG	Enable signals.
NOFLSH	Disable flush after interrupt, quit or suspend.
TOSTOP	Send SIGTTOU for background output.

6974

6975

6976

6977

6978

6979

6980

6981

6982

6983 If ECHO is set, input characters shall be echoed back to the terminal. If ECHO is clear, input |  
 6984 characters shall not be echoed. |

6985 If ECHOE and ICANON are set, the ERASE character shall cause the terminal to erase, if |  
 6986 possible, the last character in the current line from the display. If there is no character to erase, an |  
 6987 implementation may echo an indication that this was the case, or do nothing. |

6988 If ECHOK and ICANON are set, the KILL character shall either cause the terminal to erase the |  
 6989 line from the display or shall echo the newline character after the KILL character. |

6990 If ECHONL and ICANON are set, the newline character shall be echoed even if ECHO is not set.

6991 If ICANON is set, canonical processing shall be enabled. This enables the erase and kill edit  
6992 functions, and the assembly of input characters into lines delimited by NL, EOF, and EOL, as  
6993 described in Section 11.1.6 (on page 185).

6994 If ICANON is not set, read requests shall be satisfied directly from the input queue. A read shall  
6995 not be satisfied until at least MIN bytes have been received or the timeout value TIME expired  
6996 between bytes. The time value represents tenths of a second. See Section 11.1.7 (on page 186) for  
6997 more details.

6998 If IEXTEN is set, implementation-defined functions shall be recognized from the input data. It is  
6999 implementation-defined how IEXTEN being set interacts with ICANON, ISIG, IXON, or IXOFF. |  
7000 If IEXTEN is not set, implementation-defined functions shall not be recognized and the  
7001 corresponding input characters are processed as described for ICANON, ISIG, IXON, and  
7002 IXOFF.

7003 If ISIG is set, each input character shall be checked against the special control characters INTR,  
7004 QUIT, and SUSP. If an input character matches one of these control characters, the function  
7005 associated with that character shall be performed. If ISIG is not set, no checking shall be done.  
7006 Thus these special input functions are possible only if ISIG is set.

7007 If NOFLSH is set, the normal flush of the input and output queues associated with the INTR,  
7008 QUIT, and SUSP characters shall not be done.

7009 If TOSTOP is set, the signal SIGTTOU shall be sent to the process group of a process that tries to  
7010 write to its controlling terminal if it is not in the foreground process group for that terminal. This  
7011 signal, by default, stops the members of the process group. Otherwise, the output generated by |  
7012 that process shall be output to the current output stream. Processes that are blocking or ignoring |  
7013 SIGTTOU signals are excepted and allowed to produce output, and the SIGTTOU signal shall |  
7014 not be sent. |

7015 The initial local control value after *open()* is implementation-defined.

### 7016 11.2.6 Special Control Characters

7017 The special control character values shall be defined by the array `c_cc`. The subscript name and |  
7018 description for each element in both canonical and non-canonical modes are as follows:

7019  
7020  
7021  
7022  
7023  
7024  
7025  
7026  
7027  
7028  
7029  
7030  
7031  
7032  
7033

Subscript Usage		Description
Canonical Mode	Non-Canonical Mode	
VEOF		EOF character
VEOL		EOL character
VERASE		ERASE character
VINTR	VINTR	INTR character
VKILL		KILL character
	VMIN	MIN value
VQUIT	VQUIT	QUIT character
VSUSP	VSUSP	SUSP character
	VTIME	TIME value
VSTART	VSTART	START character
VSTOP	VSTOP	STOP character

7034  
7035

The subscript values are unique, except that the VMIN and VTIME subscripts may have the same values as the VEOF and VEOL subscripts, respectively.

7036  
7037  
7038

Implementations that do not support changing the START and STOP characters may ignore the character values in the `c_cc` array indexed by the VSTART and VSTOP subscripts when `tcsetattr()` is called, but shall return the value in use when `tcgetattr()` is called.

7039

The initial values of all control characters are implementation-defined.

7040  
7041  
7042  
7043

If the value of one of the changeable special control characters (see Section 11.1.9 (on page 187)) is `_POSIX_VDISABLE`, that function shall be disabled; that is, no input data is recognized as the disabled special character. If ICANON is not set, the value of `_POSIX_VDISABLE` has no special meaning for the VMIN and VTIME entries of the `c_cc` array.



7045

7046 **12.1 Utility Argument Syntax**

7047 This section describes the argument syntax of the standard utilities and introduces terminology  
 7048 used throughout IEEE Std 1003.1-200x for describing the arguments processed by the utilities.

7049 Within IEEE Std 1003.1-200x, a special notation is used for describing the syntax of a utility's  
 7050 arguments. Unless otherwise noted, all utility descriptions use this notation, which is illustrated  
 7051 by this example (see the Shell and Utilities volume of IEEE Std 1003.1-200x, Section 2.9.1, Simple  
 7052 Commands):

```
7053 utility_name[-a][-b][-c option_argument]
7054 [-d|-e][-foption_argument][operand...]
```

7055 The notation used for the SYNOPSIS sections imposes requirements on the implementors of the  
 7056 standard utilities and provides a simple reference for the application developer or system user.

7057 1. The utility in the example is named *utility\_name*. It is followed by *options*, *option-*  
 7058 *arguments*, and *operands*. The arguments that consist of hyphens and single letters or  
 7059 digits, such as 'a', are known as *options* (or, historically, *flags*). Certain options are  
 7060 followed by an *option-argument*, as shown with [-c *option\_argument*]. The arguments  
 7061 following the last options and option-arguments are named *operands*.

7062 2. Option-arguments are sometimes shown separated from their options by <blank>s,  
 7063 sometimes directly adjacent. This reflects the situation that in some cases an option-  
 7064 argument is included within the same argument string as the option; in most cases it is the  
 7065 next argument. The Utility Syntax Guidelines in Section 12.2 (on page 199) require that the  
 7066 option be a separate argument from its option-argument, but there are some exceptions in  
 7067 IEEE Std 1003.1-200x to ensure continued operation of historical applications:

7068 a. If the SYNOPSIS of a standard utility shows a space character between an option and  
 7069 option-argument (as with [-c *option\_argument*] in the example), a conforming  
 7070 application shall use separate arguments for that option and its option-argument. |

7071 b. If a space character is not shown (as with [-*foption\_argument*] in the example), a |  
 7072 conforming application shall place an option and its option-argument directly |  
 7073 adjacent in the same argument string, without intervening <blank>s.

7074 c. Notwithstanding the preceding requirements on conforming applications, a |  
 7075 conforming system shall permit, but shall not require, an application to specify  
 7076 options and option-arguments as separate arguments whether or not a space  
 7077 XSI character is shown on the synopsis line, except in those cases (marked with the XSI  
 7078 portability warning) where an option-argument is optional and no separation can be  
 7079 used.

7080 d. A standard utility may also be implemented to operate correctly when the required  
 7081 separation into multiple arguments is violated by a non-conforming application. |

7082 In summary, the following table shows allowable combinations:

7083  
7084  
7085  
7086  
7087  
7088

	SYNOPSIS Shows:		
	-a <i>arg</i>	-b <i>arg</i>	-c[ <i>arg</i> ]
Conforming application shall use:	-a <i>arg</i>	-b <i>arg</i>	N/A
System shall support:	-a <i>arg</i>	-b <i>arg</i>	-c <i>arg</i> or -c
System may support:	-a <i>arg</i>	-b <i>arg</i>	

7089  
7090  
7091  
7092  
7093  
7094  
7095  
7096  
7097

- Options are usually listed in alphabetical order unless this would make the utility description more confusing. There are no implied relationships between the options based upon the order in which they appear, unless otherwise stated in the OPTIONS section, or unless the exception in Guideline 11 of Section 12.2 (on page 199) applies. If an option that does not have option-arguments is repeated, the results are undefined, unless otherwise stated.
- Frequently, names of parameters that require substitution by actual values are shown with embedded underscores. Alternatively, parameters are shown as follows:

*<parameter name>*

7098  
7099  
7100

The angle brackets are used for the symbolic grouping of a phrase representing a single parameter and conforming applications shall not include them in data submitted to the utility.

7101  
7102  
7103

- When a utility has only a few permissible options, they are sometimes shown individually, as in the example. Utilities with many flags generally show all of the individual flags (that do not take option-arguments) grouped, as in:

*utility\_name* [-abcDxyz][*-p arg*][*operand*]

7104

Utilities with very complex arguments may be shown as follows:

7105

*utility\_name* [*options*][*operands*]

7106  
7107  
7108

- Unless otherwise specified, whenever an operand or option-argument is, or contains, a numeric value:

7109

- The number is interpreted as a decimal integer.

7110

- Numerals in the range 0 to 2 147 483 647 are syntactically recognized as numeric values.

7111

- When the utility description states that it accepts negative numbers as operands or option-arguments, numerals in the range -2 147 483 647 to 2 147 483 647 are syntactically recognized as numeric values.

7112

7113

- Ranges greater than those listed here are allowed.

7114

This does not mean that all numbers within the allowable range are necessarily semantically correct. A standard utility that accepts an option-argument or operand that is to be interpreted as a number, and for which a range of values smaller than that shown above is permitted by the IEEE Std 1003.1-200x, describes that smaller range along with the description of the option-argument or operand. If an error is generated, the utility's diagnostic message shall indicate that the value is out of the supported range, not that it is syntactically incorrect.

7115

7116

7117

- Arguments or option-arguments enclosed in the '[' and ']' notation are optional and can be omitted. Conforming applications shall not include the '[' and ']' symbols in data submitted to the utility.

7118

7119

7120

7121

- Arguments separated by the '|' vertical bar notation are mutually-exclusive. Conforming applications shall not include the '|' symbol in data submitted to the utility.

7122

7123

7124

7125

7126

7127 Alternatively, mutually-exclusive options and operands may be listed with multiple  
7128 synopsis lines. For example:

```
7129     utility_name -d[-a][-c option_argument][operand...]  
7130     utility_name[-a][-b][operand...]
```

7131 When multiple synopsis lines are given for a utility, it is an indication that the utility has  
7132 mutually-exclusive arguments. These mutually-exclusive arguments alter the functionality  
7133 of the utility so that only certain other arguments are valid in combination with one of the  
7134 mutually-exclusive arguments. Only one of the mutually-exclusive arguments is allowed  
7135 for invocation of the utility. Unless otherwise stated in an accompanying OPTIONS  
7136 section, the relationships between arguments depicted in the SYNOPSIS sections are  
7137 mandatory requirements placed on conforming applications. The use of conflicting  
7138 mutually-exclusive arguments produces undefined results, unless a utility description  
7139 specifies otherwise. When an option is shown without the '[' and ']' brackets, it means  
7140 that option is required for that version of the SYNOPSIS. However, it is not required to be  
7141 the first argument, as shown in the example above, unless otherwise stated.

7142 9. Ellipses ("...") are used to denote that one or more occurrences of an option or operand  
7143 are allowed. When an option or an operand followed by ellipses is enclosed in brackets,  
7144 zero or more options or operands can be specified. The forms:

```
7145     utility_name -f option_argument...[operand...]  
7146     utility_name [-g option_argument]...[operand...]
```

7147 indicate that multiple occurrences of the option and its option-argument preceding the  
7148 ellipses are valid, with semantics as indicated in the OPTIONS section of the utility. (See  
7149 also Guideline 11 in Section 12.2.) In the first example, each option-argument requires a  
7150 preceding `-f` and at least one `-f option_argument` must be given.

7151 10. When the synopsis line is too long to be printed on a single line in the Shell and Utilities  
7152 volume of IEEE Std 1003.1-200x, the indented lines following the initial line are  
7153 continuation lines. An actual use of the command would appear on a single logical line.

## 7154 12.2 Utility Syntax Guidelines

7155 The following guidelines are established for the naming of utilities and for the specification of  
7156 options, option-arguments, and operands. The `getopt()` function in the System Interfaces volume  
7157 of IEEE Std 1003.1-200x assists utilities in handling options and operands that conform to these  
7158 guidelines.

7159 Operands and option-arguments can contain characters not specified in the portable character  
7160 set.

7161 The guidelines are intended to provide guidance to the authors of future utilities, such as those  
7162 written specific to a local system or that are components of a larger application. Some of the  
7163 standard utilities do not conform to all of these guidelines; in those cases, the OPTIONS sections  
7164 describe the deviations.

7165 **Guideline 1:** Utility names should be between two and nine characters, inclusive.

7166 **Guideline 2:** Utility names should include lowercase letters (the **lower** character  
7167 classification) and digits only from the portable character set.

7168 **Guideline 3:** Each option name should be a single alphanumeric character (the **alnum**  
7169 character classification) from the portable character set. The `-W` (capital-W)  
7170 option shall be reserved for vendor options.

- 7171 Multi-digit options should not be allowed. |
- 7172 **Guideline 4:** All options should be preceded by the '-' delimiter character.
- 7173 **Guideline 5:** Options without option-arguments should be accepted when grouped behind  
7174 one '-' delimiter.
- 7175 **Guideline 6:** Each option and option-argument should be a separate argument, except as  
7176 noted in Section 12.1 (on page 197), item (2).
- 7177 **Guideline 7:** Option-arguments should not be optional.
- 7178 **Guideline 8:** When multiple option-arguments are specified to follow a single option, they  
7179 should be presented as a single argument, using commas within that  
7180 argument or <blank>s within that argument to separate them.
- 7181 **Guideline 9:** All options should precede operands on the command line.
- 7182 **Guideline 10:** The argument -- should be accepted as a delimiter indicating the end of  
7183 options. Any following arguments should be treated as operands, even if they  
7184 begin with the '-' character. The -- argument should not be used as an  
7185 option or as an operand.
- 7186 **Guideline 11:** The order of different options relative to one another should not matter,  
7187 unless the options are documented as mutually-exclusive and such an option  
7188 is documented to override any incompatible options preceding it. If an option  
7189 that has option-arguments is repeated, the option and option-argument  
7190 combinations should be interpreted in the order specified on the command  
7191 line.
- 7192 **Guideline 12:** The order of operands may matter and position-related interpretations should  
7193 be determined on a utility-specific basis.
- 7194 **Guideline 13:** For utilities that use operands to represent files to be opened for either reading  
7195 or writing, the '-' operand should be used only to mean standard input (or  
7196 standard output when it is clear from context that an output file is being  
7197 specified).
- 7198 The utilities in the Shell and Utilities volume of IEEE Std 1003.1-200x that claim conformance to  
7199 these guidelines shall conform completely to these guidelines as if these guidelines contained the |  
7200 term "shall" instead of "should". On some implementations, the utilities accept usage in |  
7201 violation of these guidelines for backward compatibility as well as accepting the required form. |
- 7202 It is recommended that all future utilities and applications use these guidelines to enhance user  
7203 portability. The fact that some historical utilities could not be changed (to avoid breaking |  
7204 existing applications) should not deter this future goal. |

7205

7206 This chapter describes the contents of headers.

7207 Headers contain function prototypes, the definition of symbolic constants, common structures,  
7208 preprocessor macros, and defined types. Each function in the System Interfaces volume of  
7209 IEEE Std 1003.1-200x specifies the headers that an application shall include in order to use that  
7210 function. In most cases, only one header is required. These headers are present on an application  
7211 development system; they need not be present on the target execution system.

### 7212 **13.1 Format of Entries**

7213 The entries in this chapter are based on a common format as follows. The only sections relating  
7214 to conformance are the SYNOPSIS and DESCRIPTION.

#### 7215 **NAME**

7216 This section gives the name or names of the entry and briefly states its purpose.

#### 7217 **SYNOPSIS**

7218 This section summarizes the use of the entry being described.

#### 7219 **DESCRIPTION**

7220 This section describes the functionality of the header.

#### 7221 **APPLICATION USAGE**

7222 This section is non-normative.

7223 This section gives warnings and advice to application writers about the entry. In the  
7224 event of conflict between warnings and advice and a normative part of this volume of  
7225 IEEE Std 1003.1-200x, the normative material is to be taken as correct.

#### 7226 **RATIONALE**

7227 This section is non-normative.

7228 This section contains historical information concerning the contents of this volume of  
7229 IEEE Std 1003.1-200x and why features were included or discarded by the standard  
7230 developers.

#### 7231 **FUTURE DIRECTIONS**

7232 This section is non-normative.

7233 This section provides comments which should be used as a guide to current thinking;  
7234 there is not necessarily a commitment to adopt these future directions.

#### 7235 **SEE ALSO**

7236 This section is non-normative.

7237 This section gives references to related information.

#### 7238 **CHANGE HISTORY**

7239 This section is non-normative.

7240 This section shows the derivation of the entry and any significant changes that have  
7241 been made to it.

7242 **NAME**7243 aio.h — asynchronous input and output (**REALTIME**)7244 **SYNOPSIS**

7245 AIO #include &lt;aio.h&gt;

7246

7247 **DESCRIPTION**7248 The <aio.h> header shall define the **aio** structure which shall include at least the following  
7249 members:

7250	int	aio_fildes	File descriptor.
7251	off_t	aio_offset	File offset.
7252	volatile void *	aio_buf	Location of buffer.
7253	size_t	aio_nbytes	Length of transfer.
7254	int	aio_reqprio	Request priority offset.
7255	struct sigevent	aio_sigevent	Signal number and value.
7256	int	aio_lio_opcode	Operation to be performed.

7257 This header shall also include the following constants:

7258 **AIO\_CANCELED** A return value indicating that all requested operations have been  
7259 canceled.7260 **AIO\_NOTCANCELED**7261 A return value indicating that some of the requested operations could not  
7262 be canceled since they are in progress.7263 **AIO\_ALLDONE** A return value indicating that none of the requested operations could be  
7264 canceled since they are already complete.7265 **LIO\_WAIT** A *lio\_listio()* synchronization operation indicating that the calling thread  
7266 is to suspend until the *lio\_listio()* operation is complete.7267 **LIO\_NOWAIT** A *lio\_listio()* synchronization operation indicating that the calling thread  
7268 is to continue execution while the *lio\_listio()* operation is being  
7269 performed, and no notification is given when the operation is complete.7270 **LIO\_READ** A *lio\_listio()* element operation option requesting a read.7271 **LIO\_WRITE** A *lio\_listio()* element operation option requesting a write.7272 **LIO\_NOP** A *lio\_listio()* element operation option indicating that no transfer is  
7273 requested.7274 The following shall be declared as functions and may also be defined as macros. Function  
7275 prototypes shall be provided.

```

7276 int aio_cancel(int, struct aiocb *);
7277 int aio_error(const struct aiocb *);
7278 int aio_fsync(int, struct aiocb *);
7279 int aio_read(struct aiocb *);
7280 ssize_t aio_return(struct aiocb *);
7281 int aio_suspend(const struct aiocb *const[], int,
7282 const struct timespec *);
7283 int aio_write(struct aiocb *);
7284 int lio_listio(int, struct aiocb *restrict const[restrict], int,
7285 struct sigevent *restrict);

```

7286 Inclusion of the <aio.h> header may make visible symbols defined in the headers <fcntl.h>  
7287 <signal.h>, <sys/types.h>, and <time.h>.

7288 **APPLICATION USAGE**

7289 None.

7290 **RATIONALE**

7291 None.

7292 **FUTURE DIRECTIONS**

7293 None.

7294 **SEE ALSO**

7295 <fcntl.h>, <signal.h>, <sys/types.h>, <time.h>, the System Interfaces volume of  
7296 IEEE Std 1003.1-200x, *fsync()*, *lseek()*, *read()*, *write()*

7297 **CHANGE HISTORY**

7298 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

7299 **Issue 6**

7300 The <aio.h> header is marked as part of the Asynchronous Input and Output option.

7301 The description of the constants is expanded.

7302 The **restrict** keyword is added to the prototype for *lio\_listio()*.

7303 **NAME**

7304 arpa/inet.h — definitions for internet operations

7305 **SYNOPSIS**

7306 #include &lt;arpa/inet.h&gt;

7307 **DESCRIPTION**7308 The **in\_port\_t** and **in\_addr\_t** types shall be defined as described in <netinet/in.h>.7309 The **in\_addr** structure shall be defined as described in <netinet/in.h>.7310 IP6 The **INET\_ADDRSTRLEN** and **INET6\_ADDRSTRLEN** macros shall be defined as described in |  
7311 <netinet/in.h>.7312 The following shall either be declared as functions, defined as macros, or both. If functions are |  
7313 declared, function prototypes shall be provided.

7314 uint32\_t htonl(uint32\_t);

7315 uint16\_t htons(uint16\_t);

7316 uint32\_t ntohl(uint32\_t);

7317 uint16\_t ntohs(uint16\_t);

7318 The **uint32\_t** and **uint16\_t** types shall be defined as described in <inttypes.h>.7319 The following shall be declared as functions and may also be defined as macros. Function |  
7320 prototypes shall be provided.

7321 in\_addr\_t inet\_addr(const char \*);

7322 char \*inet\_ntoa(struct in\_addr);

7323 const char \*inet\_ntop(int, const void \*restrict, char \*restrict,  
7324 socklen\_t);

7325 int inet\_pton(int, const char \*restrict, void \*restrict);

7326 Inclusion of the <arpa/inet.h> header may also make visible all symbols from <netinet/in.h> |  
7327 and <inttypes.h>.7328 **APPLICATION USAGE**

7329 None.

7330 **RATIONALE**

7331 None.

7332 **FUTURE DIRECTIONS**

7333 None.

7334 **SEE ALSO**7335 <netinet/in.h>, <inttypes.h>, the System Interfaces volume of IEEE Std 1003.1-200x, *htonl()*,  
7336 *inet\_addr()*7337 **CHANGE HISTORY**

7338 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

7339 The **restrict** keyword is added to the prototypes for *inet\_ntop()* and *inet\_pton()*.

7340 **NAME**

7341           assert.h — verify program assertion

7342 **SYNOPSIS**

7343           #include &lt;assert.h&gt;

7344 **DESCRIPTION**

7345 cx       The functionality described on this reference page is aligned with the ISO C standard. Any  
7346 conflict between the requirements described here and the ISO C standard is unintentional. This  
7347 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

7348       The <assert.h> header shall define the *assert()* macro. It refers to the macro NDEBUG which is  
7349 not defined in the header. If NDEBUG is defined as a macro name before the inclusion of this  
7350 header, the *assert()* macro shall be defined simply as:

7351       #define assert(ignore)((void) 0)

7352       Otherwise, the macro behaves as described in *assert()*.

7353       The *assert()* macro shall be redefined according to the current state of NDEBUG each time  
7354 <assert.h> is included.

7355       The *assert()* macro shall be implemented as a macro, not as a function. If the macro definition is  
7356 suppressed in order to access an actual function, the behavior is undefined.

7357 **APPLICATION USAGE**

7358       None.

7359 **RATIONALE**

7360       None.

7361 **FUTURE DIRECTIONS**

7362       None.

7363 **SEE ALSO**7364       The System Interfaces volume of IEEE Std 1003.1-200x, *assert()*7365 **CHANGE HISTORY**

7366       First released in Issue 1. Derived from Issue 1 of the SVID.

7367 **Issue 6**

7368       The definition of the *assert()* macro is changed for alignment with the ISO/IEC 9899:1999  
7369 standard.

7370 **NAME**

7371 complex.h — complex arithmetic

7372 **SYNOPSIS**

7373 #include &lt;complex.h&gt;

7374 **DESCRIPTION**

7375 **cx** The functionality described on this reference page is aligned with the ISO C standard. Any  
 7376 conflict between the requirements described here and the ISO C standard is unintentional. This  
 7377 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

7378 The **<complex.h>** header shall define the following macros:7379 **complex** Expands to **\_Complex**.7380 **\_Complex\_I** Expands to a constant expression of type **const float \_Complex**, with the  
7381 value of the imaginary unit (that is, a number such that  $i^2=-1$ ).7382 **imaginary** Expands to **\_Imaginary**.7383 **\_Imaginary\_I** Expands to a constant expression of type **const float \_Imaginary** with the  
7384 value of the imaginary unit.7385 **I** Expands to either **\_Imaginary\_I** or **\_Complex\_I**. If **\_Imaginary\_I** is not defined,  
7386 **I** expands to **\_Complex\_I**.7387 The macros **imaginary** and **\_Imaginary\_I** shall be defined if and only if the implementation  
7388 supports imaginary types.7389 An application may undefine and then, perhaps, redefine the **complex**, **imaginary**, and **I** macros.7390 The following shall be declared as functions and may also be defined as macros. Function  
7391 prototypes shall be provided.

7392	double	cabs(double complex);
7393	float	cabsf(float complex);
7394	long double	cabsl(long double complex);
7395	double complex	acos(double complex);
7396	float complex	acosf(float complex);
7397	double complex	acosh(double complex);
7398	float complex	acoshf(float complex);
7399	long double complex	acoshl(long double complex);
7400	long double complex	acosl(long double complex);
7401	double	carg(double complex);
7402	float	cargf(float complex);
7403	long double	cargl(long double complex);
7404	double complex	casin(double complex);
7405	float complex	casinf(float complex);
7406	double complex	casinh(double complex);
7407	float complex	casinhf(float complex);
7408	long double complex	casinhl(long double complex);
7409	long double complex	casinl(long double complex);
7410	double complex	catan(double complex);
7411	float complex	catanf(float complex);
7412	double complex	catanh(double complex);
7413	float complex	catanhf(float complex);
7414	long double complex	catanhl(long double complex);
7415	long double complex	catanl(long double complex);

```

7416     double complex      ccos(double complex);
7417     float complex       ccosf(float complex);
7418     double complex      ccosh(double complex);
7419     float complex       ccoshf(float complex);
7420     long double complex ccoshl(long double complex);
7421     long double complex ccosl(long double complex);
7422     double complex      cexp(double complex);
7423     float complex       cexpf(float complex);
7424     long double complex cexpl(long double complex);
7425     double              cimag(double complex);
7426     float              cimagf(float complex);
7427     long double        cimagl(long double complex);
7428     double complex     clog(double complex);
7429     float complex      clogf(float complex);
7430     long double complex clogl(long double complex);
7431     double complex     conj(double complex);
7432     float complex      conjf(float complex);
7433     long double complex conjl(long double complex);
7434     double complex     cpow(double complex, double complex);
7435     float complex      cpowf(float complex, float complex);
7436     long double complex cpowl(long double complex, long double complex);
7437     double complex     cproj(double complex);
7438     float complex      cprojf(float complex);
7439     long double complex cprojl(long double complex);
7440     double             creal(double complex);
7441     float             crealf(float complex);
7442     long double        creall(long double complex);
7443     double complex     csin(double complex);
7444     float complex      csinf(float complex);
7445     double complex     csinh(double complex);
7446     float complex      csinhf(float complex);
7447     long double complex csinhl(long double complex);
7448     long double complex csinl(long double complex);
7449     double complex     csqrt(double complex);
7450     float complex      csqrtf(float complex);
7451     long double complex csqrtl(long double complex);
7452     double complex     ctan(double complex);
7453     float complex      ctanf(float complex);
7454     double complex     ctanh(double complex);
7455     float complex      ctanhf(float complex);
7456     long double complex ctanhl(long double complex);
7457     long double complex ctanl(long double complex);

```

#### 7458 APPLICATION USAGE

7459 Values are interpreted as radians, not degrees.

#### 7460 RATIONALE

7461 The choice of *I* instead of *i* for the imaginary unit concedes to the widespread use of the  
7462 identifier *i* for other purposes. The application can use a different identifier, say *j*, for the  
7463 imaginary unit by following the inclusion of the <complex.h> header with:

```

7464     #undef I
7465     #define j _Imaginary_I

```

7466 An *I* suffix to designate imaginary constants is not required, as multiplication by *I* provides a  
7467 sufficiently convenient and more generally useful notation for imaginary terms. The  
7468 corresponding real type for the imaginary unit is **float**, so that use of *I* for algorithmic or  
7469 notational convenience will not result in widening types.

7470 On systems with imaginary types, the application has the ability to control whether use of the  
7471 macro **I** introduces an imaginary type, by explicitly defining **I** to be `_Imaginary_I` or `_Complex_I`.  
7472 Disallowing imaginary types is useful for some applications intended to run on implementations  
7473 without support for such types.

7474 The macro `_Imaginary_I` provides a test for whether imaginary types are supported.

7475 The `cis()` function ( $\cos(x) + I\sin(x)$ ) was considered but rejected because its implementation is  
7476 easy and straightforward, even though some implementations could compute sine and cosine  
7477 more efficiently in tandem.

#### 7478 **FUTURE DIRECTIONS**

7479 The following function names and the same names suffixed with *f* or *l* are reserved for future  
7480 use, and may be added to the declarations in the **<complex.h>** header.

7481	<code>cerf()</code>	<code>cexpm1()</code>	<code>clog2()</code>
7482	<code>cerfc()</code>	<code>clog10()</code>	<code>clgamma()</code>
7483	<code>cexp2()</code>	<code>clog1p()</code>	<code>ctgamma()</code>

#### 7484 **SEE ALSO**

7485 The System Interfaces volume of IEEE Std 1003.1-200x, `cabs()`, `cacos()`, `cacosh()`, `carg()`, `casin()`,  
7486 `casinh()`, `catan()`, `catanh()`, `ccos()`, `ccosh()`, `cexp()`, `cimag()`, `clog()`, `conj()`, `cpow()`, `cproj()`, `creal()`,  
7487 `csin()`, `csinh()`, `csqrt()`, `ctan()`, `ctanh()`

#### 7488 **CHANGE HISTORY**

7489 First released in Issue 6. Included for alignment with the ISO/IEC 9899:1999 standard.

7490 **NAME**

7491 cpio.h — cpio archive values

7492 **SYNOPSIS**

7493 xSI #include <cpio.h>

7494

7495 **DESCRIPTION**

7496 Values needed by the *c\_mode* field of the *cpio* archive format are described as follows:

7497

7498

Name	Description	Value (Octal)
C_IRUSR	Read by owner.	0000400
C_IWUSR	Write by owner.	0000200
C_IXUSR	Execute by owner.	0000100
C_IRGRP	Read by group.	0000040
C_IWGRP	Write by group.	0000020
C_IXGRP	Execute by group.	0000010
C_IROTH	Read by others.	0000004
C_IWOTH	Write by others.	0000002
C_IXOTH	Execute by others.	0000001
C_ISUID	Set user ID.	0004000
C_ISGID	Set group ID.	0002000
C_ISVTX	On directories, restricted deletion flag.	0001000
C_ISDIR	Directory.	0040000
C_ISFIFO	FIFO.	0010000
C_ISREG	Regular file.	0100000
C_ISBLK	Block special.	0060000
C_ISCHR	Character special.	0020000
C_ISCTG	Reserved.	0110000
C_ISLNK	Symbolic link.	0120000
C_ISSOCK	Socket.	0140000

7519 The header shall define the symbolic constant:

7520 MAGIC "070707"

7521 **APPLICATION USAGE**

7522 None.

7523 **RATIONALE**

7524 None.

7525 **FUTURE DIRECTIONS**

7526 None.

7527 **SEE ALSO**

7528 The Shell and Utilities volume of IEEE Std 1003.1-200x, *pax*

7529 **CHANGE HISTORY**

7530 First released in Issue 3 of the Headers Interface, Issue 3 specification. Derived from the  
7531 POSIX.1-1988 standard.

7532 **Issue 6**

7533 The SEE ALSO is updated to refer to *pax*, since the *cpio* utility is not included in the Shell and  
7534 Utilities volume of IEEE Std 1003.1-200x.

7535 **NAME**

7536         ctype.h — character types

7537 **SYNOPSIS**

7538         #include &lt;ctype.h&gt;

7539 **DESCRIPTION**

7540 **CX**         Some of the functionality described on this reference page extends the ISO C standard.  
7541         Applications shall define the appropriate feature test macro (see the System Interfaces volume of  
7542         IEEE Std 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of these  
7543         symbols in this header.

7544         The following shall be declared as functions and may also be defined as macros. Function  
7545         prototypes shall be provided.

7546         int    isalnum(int);

7547         int    isalpha(int);

7548 **XSI**         int    isascii(int);

7549         int    isblank(int);

7550         int    iscntrl(int);

7551         int    isdigit(int);

7552         int    isgraph(int);

7553         int    islower(int);

7554         int    isprint(int);

7555         int    ispunct(int);

7556         int    isspace(int);

7557         int    isupper(int);

7558         int    isxdigit(int);

7559 **XSI**         int    toascii(int);

7560         int    tolower(int);

7561         int    toupper(int);

7562         The following are defined as macros:

7563 **XSI**         int    \_toupper(int);

7564         int    \_tolower(int);

7565

7566 **APPLICATION USAGE**

7567         None.

7568 **RATIONALE**

7569         None.

7570 **FUTURE DIRECTIONS**

7571         None.

7572 **SEE ALSO**

7573         <locale.h>, the System Interfaces volume of IEEE Std 1003.1-200x, *isalnum()*, *isalpha()*, *isascii()*,  
7574         *iscntrl()*, *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isspace()*, *isupper()*, *isxdigit()*, *mblen()*,  
7575         *mbstowcs()*, *mbtowc()*, *setlocale()*, *toascii()*, *tolower()*, *\_tolower()*, *toupper()*, *\_toupper()*, *wcstombs()*,  
7576         *wctomb()*

7577 **CHANGE HISTORY**

7578         First released in Issue 1. Derived from Issue 1 of the SVID.

7579 **Issue 6**

7580 Extensions beyond the ISO C standard are now marked.

7581 **NAME**7582 `dirent.h` — format of directory entries7583 **SYNOPSIS**7584 `#include <dirent.h>`7585 **DESCRIPTION**

7586 The internal format of directories is unspecified.

7587 The **<dirent.h>** header shall define the following type:7588 **DIR** A type representing a directory stream.7589 It shall also define the structure **dirent** which shall include the following members:7590 XSI `ino_t d_ino` File serial number.7591 `char d_name[]` Name of entry.7592 XSI The type `ino_t` shall be defined as described in **<sys/types.h>**.7593 The character array `d_name` is of unspecified size, but the number of bytes preceding the  
7594 terminating null byte shall not exceed `{NAME_MAX}`.7595 The following shall be declared as functions and may also be defined as macros. Function  
7596 prototypes shall be provided. |7597 `int closedir(DIR *);`7598 `DIR *opendir(const char *);`7599 `struct dirent *readdir(DIR *);`7600 TSF `int readdir_r(DIR *restrict, struct dirent *restrict,  
7601 struct dirent **restrict);`7602 `void rewinddir(DIR *);`7603 XSI `void seekdir(DIR *, long);`7604 `long telldir(DIR *);`

7605

7606 **APPLICATION USAGE**

7607 None.

7608 **RATIONALE**7609 Information similar to that in the **<dirent.h>** header is contained in a file **<sys/dir.h>** in 4.2 BSD  
7610 and 4.3 BSD. The equivalent in these implementations of **struct dirent** from this volume of  
7611 IEEE Std 1003.1-200x is **struct direct**. The filename was changed because the name **<sys/dir.h>**  
7612 was also used in earlier implementations to refer to definitions related to the older access  
7613 method; this produced name conflicts. The name of the structure was changed because this  
7614 volume of IEEE Std 1003.1-200x does not completely define what is in the structure, so it could  
7615 be different on some implementations from **struct direct**.7616 The name of an array of **char** of an unspecified size should not be used as an lvalue. Use of: |7617 `sizeof(d_name)`

7618 is incorrect; use:

7619 `strlen(d_name)`

7620 instead.

7621 The array of **char** `d_name` is not a fixed size. Implementations may need to declare **struct dirent**  
7622 with an array size for `d_name` of 1, but the actual number of characters provided matches (or  
7623 only slightly exceeds) the length of the filename.

7624 **FUTURE DIRECTIONS**

7625 None.

7626 **SEE ALSO**7627 <sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-200x, *closedir()*, *opendir()*,  
7628 *readdir()*, *readdir\_r()*, *rewinddir()*, *seekdir()*, *telldir()*7629 **CHANGE HISTORY**

7630 First released in Issue 2.

7631 **Issue 5**

7632 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

7633 **Issue 6**7634 The Open Group Corrigendum U026/7 is applied, correcting the prototype for *readdir\_r()*.7635 The **restrict** keyword is added to the prototype for *readdir\_r()*.

7636 **NAME**7637 `dlfcn.h` — dynamic linking7638 **SYNOPSIS**7639 XSI `#include <dlfcn.h>`

7640

7641 **DESCRIPTION**7642 The `<dlfcn.h>` header shall define at least the following macros for use in the construction of a  
7643 `dlopen()` *mode* argument:7644 `RTLD_LAZY` Relocations are performed at an implementation-defined time.7645 `RTLD_NOW` Relocations are performed when the object is loaded.7646 `RTLD_GLOBAL` All symbols are available for relocation processing of other modules.7647 `RTLD_LOCAL` All symbols are not made available for relocation processing by other  
7648 modules.7649 The following shall be declared as functions and may also be defined as macros. Function |  
7650 prototypes shall be provided. |7651 `int dlclose(void *);`7652 `char *dlerror(void);`7653 `void *dlopen(const char *, int);`7654 `void *dlsym(void *restrict, const char *restrict);`7655 **APPLICATION USAGE**

7656 None.

7657 **RATIONALE**

7658 None.

7659 **FUTURE DIRECTIONS**

7660 None.

7661 **SEE ALSO**7662 The System Interfaces volume of IEEE Std 1003.1-200x, `dlopen()`, `dlclose()`, `dlsym()`, `dlerror()`7663 **CHANGE HISTORY**

7664 First released in Issue 5.

7665 **Issue 6**7666 The `restrict` keyword is added to the prototype for `dlsym()`.

7667 **NAME**

7668            errno.h — system error numbers

7669 **SYNOPSIS**

7670            #include &lt;errno.h&gt;

7671 **DESCRIPTION**

7672 **CX**        Some of the functionality described on this reference page extends the ISO C standard. Any  
 7673 conflict between the requirements described here and the ISO C standard is unintentional. This  
 7674 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

7675 **CX**        The ISO C standard only requires the symbols [EDOM], [EILSEQ], and [ERANGE] to be defined.

7676            The <errno.h> header shall provide a declaration for *errno* and give positive values for the  
 7677 following symbolic constants. Their values shall be unique except as noted below:

7678	[E2BIG]	Argument list too long.
7679	[EACCES]	Permission denied.
7680	[EADDRINUSE]	Address in use.
7681	[EADDRNOTAVAIL]	Address not available.
7682	[EAFNOSUPPORT]	Address family not supported.
7683	[EAGAIN]	Resource unavailable, try again (may be the same value as
7684		[EWOULDBLOCK]).
7685	[EALREADY]	Connection already in progress.
7686	[EBADF]	Bad file descriptor.
7687	[EBADMSG]	Bad message.
7688	[EBUSY]	Device or resource busy.
7689	[ECANCELED]	Operation canceled.
7690	[ECHILD]	No child processes.
7691	[ECONNABORTED]	Connection aborted.
7692	[ECONNREFUSED]	Connection refused.
7693	[ECONNRESET]	Connection reset.
7694	[EDEADLK]	Resource deadlock would occur.
7695	[EDESTADDRREQ]	Destination address required.
7696	[EDOM]	Mathematics argument out of domain of function.
7697	[EDQUOT]	Reserved.
7698	[EEXIST]	File exists.
7699	[EFAULT]	Bad address.
7700	[EFBIG]	File too large.
7701	[EHOSTUNREACH]	Host is unreachable.
7702	[EIDRM]	Identifier removed.
7703	[EILSEQ]	Illegal byte sequence.

7704		[EINPROGRESS]	Operation in progress.
7705		[EINTR]	Interrupted function.
7706		[EINVAL]	Invalid argument.
7707		[EIO]	I/O error.
7708		[EISCONN]	Socket is connected.
7709		[EISDIR]	Is a directory.
7710		[ELOOP]	Too many levels of symbolic links.
7711		[EMFILE]	Too many open files.
7712		[EMLINK]	Too many links.
7713		[EMSGSIZE]	Message too large.
7714		[EMULTIHOP]	Reserved.
7715		[ENAMETOOLONG]	Filename too long.
7716		[ENETDOWN]	Network is down.
7717		[ENETUNREACH]	Network unreachable.
7718		[ENFILE]	Too many files open in system.
7719		[ENOBUFS]	No buffer space available.
7720	XSR	[ENODATA]	No message is available on the STREAM head read queue.
7721		[ENODEV]	No such device.
7722		[ENOENT]	No such file or directory.
7723		[ENOEXEC]	Executable file format error.
7724		[ENOLCK]	No locks available.
7725		[ENOLINK]	Reserved.
7726		[ENOMEM]	Not enough space.
7727		[ENOMSG]	No message of the desired type.
7728		[ENOPROTOPT]	Protocol not available.
7729		[ENOSPC]	No space left on device.
7730	XSR	[ENOSR]	No STREAM resources.
7731	XSR	[ENOSTR]	Not a STREAM.
7732		[ENOSYS]	Function not supported.
7733		[ENOTCONN]	The socket is not connected.
7734		[ENOTDIR]	Not a directory.
7735		[ENOTEMPTY]	Directory not empty.
7736		[ENOTSOCK]	Not a socket.
7737		[ENOTSUP]	Not supported.

7738	[ENOTTY]	Inappropriate I/O control operation.
7739	[ENXIO]	No such device or address.
7740	[EOPNOTSUPP]	Operation not supported on socket.
7741	[EOVERFLOW]	Value too large to be stored in data type.
7742	[EPERM]	Operation not permitted.
7743	[EPIPE]	Broken pipe.
7744	[EPROTO]	Protocol error.
7745	[EPROTONOSUPPORT]	
7746		Protocol not supported.
7747	[EPROTOTYPE]	Protocol wrong type for socket.
7748	[ERANGE]	Result too large.
7749	[EROFS]	Read-only file system.
7750	[ESPIPE]	Invalid seek.
7751	[ESRCH]	No such process.
7752	[ESTALE]	Reserved.
7753	XSR [ETIME]	Stream <i>ioctl()</i> timeout.
7754	[ETIMEDOUT]	Connection timed out.
7755	[ETXTBSY]	Text file busy.
7756	[EWOULDBLOCK]	Operation would block (may be the same value as [EAGAIN]).
7757	[EXDEV]	Cross-device link.
7758	<b>APPLICATION USAGE</b>	
7759	Additional error numbers may be defined on conforming systems; see the System Interfaces	
7760	volume of IEEE Std 1003.1-200x.	
7761	<b>RATIONALE</b>	
7762	None.	
7763	<b>FUTURE DIRECTIONS</b>	
7764	None.	
7765	<b>SEE ALSO</b>	
7766	The System Interfaces volume of IEEE Std 1003.1-200x, Section 2.3, Error Numbers	
7767	<b>CHANGE HISTORY</b>	
7768	First released in Issue 1. Derived from Issue 1 of the SVID.	
7769	<b>Issue 5</b>	
7770	Updated for alignment with the POSIX Realtime Extension.	
7771	<b>Issue 6</b>	
7772	The following new requirements on POSIX implementations derive from alignment with the	
7773	Single UNIX Specification:	
7774	<ul style="list-style-type: none"> <li>• The majority of the error conditions previously marked as extensions are now mandatory,</li> </ul>	
7775	except for the STREAMS-related error conditions.	

7776  
7777

Values for *errno* are now required to be distinct positive values rather than non-zero values. This change is for alignment with the ISO/IEC 9899:1999 standard.

7778 **NAME**

7779       fcntl.h — file control options

7780 **SYNOPSIS**

7781       #include <fcntl.h>

7782 **DESCRIPTION**

7783       The <fcntl.h> header shall define the following requests and arguments for use by the functions  
7784       *fcntl()* and *open()*.

7785       Values for *cmd* used by *fcntl()* (the following values are unique) are as follows:

- 7786       F\_DUPFD       Duplicate file descriptor.
- 7787       F\_GETFD       Get file descriptor flags.
- 7788       F\_SETFD       Set file descriptor flags.
- 7789       F\_GETFL       Get file status flags and file access modes.
- 7790       F\_SETFL       Set file status flags.
- 7791       F\_GETLK       Get record locking information.
- 7792       F\_SETLK       Set record locking information.
- 7793       F\_SETLKW      Set record locking information; wait if blocked.
- 7794       F\_GETOWN      Get process or process group ID to receive SIGURG signals.
- 7795       F\_SETOWN      Set process or process group ID to receive SIGURG signals.

7796       File descriptor flags used for *fcntl()* are as follows:

- 7797       FD\_CLOEXEC   Close the file descriptor upon execution of an *exec* family function.

7798       Values for *l\_type* used for record locking with *fcntl()* (the following values are unique) are as  
7799       follows:

- 7800       F\_RDLCK      Shared or read lock.
- 7801       F\_UNLCK      Unlock.
- 7802       F\_WRLCK      Exclusive or write lock.

7803 XSI       The values used for *l\_whence*, *SEEK\_SET*, *SEEK\_CUR*, and *SEEK\_END* shall be defined as  
7804       described in <unistd.h>.

7805       The following values are file creation flags and are used in the *oflag* value to *open()*. They shall  
7806       be bitwise-distinct. |

- 7807       O\_CREAT       Create file if it does not exist.
- 7808       O\_EXCL       Exclusive use flag.
- 7809       O\_NOCTTY      Do not assign controlling terminal.
- 7810       O\_TRUNC       Truncate flag.

7811       File status flags used for *open()* and *fcntl()* are as follows:

- 7812       O\_APPEND      Set append mode.
- 7813 SIO       O\_DSYNC       Write according to synchronized I/O data integrity completion.
- 7814       O\_NONBLOCK    Non-blocking mode.

7815 SIO O\_RSYNC Synchronized read I/O operations.

7816 O\_SYNC Write according to synchronized I/O file integrity completion.

7817 Mask for use with file access modes is as follows:

7818 O\_ACCMODE Mask for file access modes.

7819 File access modes used for *open()* and *fcntl()* are as follows:

7820 O\_RDONLY Open for reading only.

7821 O\_RDWR Open for reading and writing.

7822 O\_WRONLY Open for writing only.

7823 XSI The symbolic names for file modes for use as values of **mode\_t** shall be defined as described in  
7824 <sys/stat.h>.

7825 ADV Values for *advice* used by *posix\_fadvise()* are as follows:

7826 POSIX\_FADV\_NORMAL  
7827 The application has no advice to give on its behavior with respect to the specified data. It is  
7828 the default characteristic if no advice is given for an open file.

7829 POSIX\_FADV\_SEQUENTIAL  
7830 The application expects to access the specified data sequentially from lower offsets to  
7831 higher offsets.

7832 POSIX\_FADV\_RANDOM  
7833 The application expects to access the specified data in a random order.

7834 POSIX\_FADV\_WILLNEED  
7835 The application expects to access the specified data in the near future.

7836 POSIX\_FADV\_DONTNEED  
7837 The application expects that it will not access the specified data in the near future.

7838 POSIX\_FADV\_NOREUSE  
7839 The application expects to access the specified data once and then not reuse it thereafter.  
7840

7841 The structure  **flock** describes a file lock. It shall include the following members:

7842 short l\_type Type of lock; F\_RDLCK, F\_WRLCK, F\_UNLCK.  
7843 short l\_whence Flag for starting offset.  
7844 off\_t l\_start Relative offset in bytes.  
7845 off\_t l\_len Size; if 0 then until EOF.  
7846 pid\_t l\_pid Process ID of the process holding the lock; returned with F\_GETLK.

7847 The **mode\_t**, **off\_t**, and **pid\_t** types shall be defined as described in <sys/types.h>.

7848 The following shall be declared as functions and may also be defined as macros. Function |  
7849 prototypes shall be provided. |

7850 int creat(const char \*, mode\_t);  
7851 int fcntl(int, int, ...);  
7852 int open(const char \*, int, ...);  
7853 ADV int posix\_fadvise(int, off\_t, size\_t, int);  
7854 int posix\_fallocate(int, off\_t, size\_t);  
7855

7856 XSI Inclusion of the <fcntl.h> header may also make visible all symbols from <sys/stat.h> and  
7857 <unistd.h>.

7858 **APPLICATION USAGE**

7859 None.

7860 **RATIONALE**

7861 None.

7862 **FUTURE DIRECTIONS**

7863 None.

7864 **SEE ALSO**

7865 <sys/stat.h>, <sys/types.h>, <unistd.h>, the System Interfaces volume of IEEE Std 1003.1-200x,  
7866 *creat()*, *exec()*, *fcntl()*, *open()*, *posix\_fadvise()*, *posix\_fallocate()*, *posix\_madvise()*

7867 **CHANGE HISTORY**

7868 First released in Issue 1. Derived from Issue 1 of the SVID.

7869 **Issue 5**

7870 The DESCRIPTION is updated for alignment with POSIX Realtime Extension.

7871 **Issue 6**

7872 The following changes are made for alignment with the ISO POSIX-1: 1996 standard:

7873 • O\_DSYNC and O\_RSYNC are marked as part of the Synchronized Input and Output option.

7874 The following new requirements on POSIX implementations derive from alignment with the  
7875 Single UNIX Specification:

7876 • The definition of the **mode\_t**, **off\_t**, and **pid\_t** types is mandated.

7877 The F\_GETOWN and F\_SETOWN values are added for sockets.

7878 The *posix\_fadvise()*, *posix\_fallocate()*, and *posix\_madvise()* functions are added for alignment with  
7879 IEEE Std 1003.1d-1999.

7880 IEEE PASC Interpretation 1003.1 #102 is applied moving the prototype for *posix\_madvise()* to  
7881 <sys/mman.h>.

7882 **NAME**7883 `fenv.h` — floating-point environment7884 **SYNOPSIS**7885 `#include <fenv.h>`7886 **DESCRIPTION**7887 `cx` The functionality described on this reference page is aligned with the ISO C standard. Any  
7888 conflict between the requirements described here and the ISO C standard is unintentional. This  
7889 volume of IEEE Std 1003.1-200x defers to the ISO C standard.7890 The **<fenv.h>** header shall define the following data types through **typedef**:7891 **fenv\_t** Represents the entire floating-point environment. The floating-point environment  
7892 refers collectively to any floating-point status flags and control modes supported  
7893 by the implementation.7894 **fexcept\_t** Represents the floating-point status flags collectively, including any status the  
7895 implementation associates with the flags. A floating-point status flag is a system  
7896 variable whose value is set (but never cleared) when a floating-point exception is  
7897 raised, which occurs as a side effect of exceptional floating-point arithmetic to  
7898 provide auxiliary information. A floating-point control mode is a system variable  
7899 whose value may be set by the user to affect the subsequent behavior of floating-  
7900 point arithmetic.7901 The **<fenv.h>** header shall define the following constants if and only if the implementation  
7902 supports the floating-point exception by means of the floating-point functions *fwclearexcept()*,  
7903 *fegetexceptflag()*, *feraiseexcept()*, *fesetexceptflag()*, and *fetestexcept()*. Each expands to an integer  
7904 constant expression with values such that bitwise-inclusive ORs of all combinations of the  
7905 constants result in distinct values.7906 `FE_DIVBYZERO`  
7907 `FE_INEXACT`  
7908 `FE_INVALID`  
7909 `FE_OVERFLOW`  
7910 `FE_UNDERFLOW`7911 The **<fenv.h>** header shall define the following constant, which is simply the bitwise-inclusive  
7912 OR of all floating-point exception constants defined above:7913 `FE_ALL_EXCEPT`7914 The **<fenv.h>** header shall define the following constants if and only if the implementation  
7915 supports getting and setting the represented rounding direction by means of the *fegetround()*  
7916 and *fesetround()* functions. Each expands to an integer constant expression whose values are  
7917 distinct non-negative values.7918 `FE_DOWNWARD`  
7919 `FE_TONEAREST`  
7920 `FE_TOWARDZERO`  
7921 `FE_UPWARD`7922 The **<fenv.h>** header shall define the following constant, which represents the default floating-  
7923 point environment (that is, the one installed at program startup) and has type pointer to const-  
7924 qualified **fenv\_t**. It can be used as an argument to the functions within the **<fenv.h>** header that  
7925 manage the floating-point environment.7926 `FE_DFL_ENV`

7927 The following shall be declared as functions and may also be defined as macros. Function |  
 7928 prototypes shall be provided. |

```
7929 int feclearexcept(int);
7930 int fegetexceptflag(fexcept_t *, int);
7931 int feraiseexcept(int);
7932 int fesetexceptflag(const fexcept_t *, int);
7933 int fetestexcept(int);
7934 int fegetround(void);
7935 int fesetround(int);
7936 int fegetenv(fenv_t *);
7937 int feholdexcept(fenv_t *);
7938 int fesetenv(const fenv_t *);
7939 int feupdateenv(const fenv_t *);
```

7940 The FENV\_ACCESS pragma provides a means to inform the implementation when an  
 7941 application might access the floating-point environment to test floating-point status flags or run  
 7942 under non-default floating-point control modes. The pragma shall occur either outside external  
 7943 declarations or preceding all explicit declarations and statements inside a compound statement.  
 7944 When outside external declarations, the pragma takes effect from its occurrence until another  
 7945 FENV\_ACCESS pragma is encountered, or until the end of the translation unit. When inside a  
 7946 compound statement, the pragma takes effect from its occurrence until another FENV\_ACCESS  
 7947 pragma is encountered (including within a nested compound statement), or until the end of the  
 7948 compound statement; at the end of a compound statement the state for the pragma is restored to  
 7949 its condition just before the compound statement. If this pragma is used in any other context, the  
 7950 behavior is undefined. If part of an application tests floating-point status flags, sets floating-  
 7951 point control modes, or runs under non-default mode settings, but was translated with the state  
 7952 for the FENV\_ACCESS pragma off, the behavior is undefined. The default state (on or off) for  
 7953 the pragma is implementation-defined. (When execution passes from a part of the application  
 7954 translated with FENV\_ACCESS off to a part translated with FENV\_ACCESS on, the state of the  
 7955 floating-point status flags is unspecified and the floating-point control modes have their default  
 7956 settings.)

#### 7957 APPLICATION USAGE

7958 This header is designed to support the floating-point exception status flags and directed-  
 7959 rounding control modes required by the IEC 60559:1989 standard, and other similar floating-  
 7960 point state information. Also it is designed to facilitate code portability among all systems.

7961 Certain application programming conventions support the intended model of use for the  
 7962 floating-point environment:

- 7963 • A function call does not alter its caller's floating-point control modes, clear its caller's  
 7964 floating-point status flags, nor depend on the state of its caller's floating-point status flags  
 7965 unless the function is so documented.
- 7966 • A function call is assumed to require default floating-point control modes, unless its  
 7967 documentation promises otherwise.
- 7968 • A function call is assumed to have the potential for raising floating-point exceptions, unless  
 7969 its documentation promises otherwise.

7970 With these conventions, an application can safely assume default floating-point control modes  
 7971 (or be unaware of them). The responsibilities associated with accessing the floating-point  
 7972 environment fall on the application that does so explicitly.

7973 Even though the rounding direction macros may expand to constants corresponding to the  
 7974 values of FLT\_ROUNDS, they are not required to do so.

```

7975     For example:
7976     #include <fenv.h>
7977     void f(double x)
7978     {
7979         #pragma STDC FENV_ACCESS ON
7980         void g(double);
7981         void h(double);
7982         /* ... */
7983         g(x + 1);
7984         h(x + 1);
7985         /* ... */
7986     }

```

7987 If the function *g()* might depend on status flags set as a side effect of the first *x+1*, or if the  
7988 second *x+1* might depend on control modes set as a side effect of the call to function *g()*, then  
7989 the application shall contain an appropriately placed invocation as follows:

```

7990     #pragma STDC FENV_ACCESS ON

```

7991 **RATIONALE**

7992 **The *fexcept\_t* Type**

7993 **fexcept\_t** does not have to be an integer type. Its values must be obtained by a call to  
7994 *fegetexceptflag()*, and cannot be created by logical operations from the exception macros. An  
7995 implementation might simply implement **fexcept\_t** as an **int** and use the representations  
7996 reflected by the exception macros, but is not required to; other representations might contain  
7997 extra information about the exceptions. **fexcept\_t** might be a **struct** with a member for each  
7998 exception (that might hold the address of the first or last floating-point instruction that caused  
7999 that exception). The ISO/IEC 9899:1999 standard makes no claims about the internals of an  
8000 **fexcept\_t**, and so the user cannot inspect it.

8001 **Exception and Rounding Macros**

8002 Macros corresponding to unsupported modes and rounding directions are not defined by the  
8003 implementation and must not be defined by the application. An application might use **#ifdef** to  
8004 test for this.

8005 **FUTURE DIRECTIONS**

8006 None.

8007 **SEE ALSO**

8008 The System Interfaces volume of IEEE Std 1003.1-200x, *feclearexcept()*, *fegetenv()*, *fegetexceptflag()*,  
8009 *fegetround()*, *fehldexcept()*, *feraiseexcept()*, *fesetenv()*, *fesetexceptflag()*, *fesetround()*, *fetestexcept()*,  
8010 *feupdateenv()*

8011 **CHANGE HISTORY**

8012 First released in Issue 6. Included for alignment with the ISO/IEC 9899:1999 standard.

8013 The return types for *feclearexcept()*, *fegetexceptflag()*, *feraiseexcept()*, *fesetexceptflag()*, *fegetenv()*,  
8014 *fesetenv()*, and *feupdateenv()* are changed from **void** to **int** for alignment with the  
8015 ISO/IEC 9899:1999 standard, Defect Report 202.

8016 **NAME**

8017 float.h — floating types

8018 **SYNOPSIS**

8019 #include <float.h>

8020 **DESCRIPTION**

8021 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
 8022 conflict between the requirements described here and the ISO C standard is unintentional. This  
 8023 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

8024 The characteristics of floating types are defined in terms of a model that describes a  
 8025 representation of floating-point numbers and values that provide information about an  
 8026 implementation's floating-point arithmetic.

8027 The following parameters are used to define the model for each floating-point type:

8028 *s* Sign ( $\pm 1$ ).

8029 *b* Base or radix of exponent representation (an integer  $> 1$ ).

8030 *e* Exponent (an integer between a minimum  $e_{\min}$  and a maximum  $e_{\max}$ ).

8031 *p* Precision (the number of base-*b* digits in the significand).

8032  $f_k$  Non-negative integers less than *b* (the significand digits).

8033 A floating-point number *x* is defined by the following model:

8034 
$$x = sb^e \sum_{k=1}^p f_k b^{-k}, e_{\min} \leq e \leq e_{\max}$$

8035 In addition to normalized floating-point numbers ( $f_1 > 0$  if  $x \neq 0$ ), floating types may be able to  
 8036 contain other kinds of floating-point numbers, such as subnormal floating-point numbers ( $x \neq 0$ ,  
 8037  $e = e_{\min}$ ,  $f_1 = 0$ ) and unnormalized floating-point numbers ( $x \neq 0$ ,  $e > e_{\min}$ ,  $f_1 = 0$ ), and values that are  
 8038 not floating-point numbers, such as infinities and NaNs. A NaN is an encoding signifying Not-  
 8039 a-Number. A *quiet NaN* propagates through almost every arithmetic operation without raising a  
 8040 floating-point exception; a *signaling NaN* generally raises a floating-point exception when  
 8041 occurring as an arithmetic operand.

8042 The accuracy of the floating-point operations ('+', '-', '\*', '/') and of the library functions  
 8043 in <math.h> and <complex.h> that return floating-point results is implementation-defined. The  
 8044 implementation may state that the accuracy is unknown.

8045 All integer values in the <float.h> header, except FLT\_ROUNDS, shall be constant expressions  
 8046 suitable for use in #if preprocessing directives; all floating values shall be constant expressions.  
 8047 All except DECIMAL\_DIG, FLT\_EVAL\_METHOD, FLT\_RADIX, and FLT\_ROUNDS have  
 8048 separate names for all three floating-point types. The floating-point model representation is  
 8049 provided for all values except FLT\_EVAL\_METHOD and FLT\_ROUNDS.

8050 The rounding mode for floating-point addition is characterized by the implementation-defined  
 8051 value of FLT\_ROUNDS:

8052 -1 Indeterminable.

8053 0 Toward zero.

8054 1 To nearest.

8055 2 Toward positive infinity.

8056           3   Toward negative infinity.

8057           All other values for FLT\_ROUNDS characterize implementation-defined rounding behavior.

8058           The values of operations with floating operands and values subject to the usual arithmetic  
8059           conversions and of floating constants are evaluated to a format whose range and precision may  
8060           be greater than required by the type. The use of evaluation formats is characterized by the  
8061           implementation-defined value of FLT\_EVAL\_METHOD:

8062           -1   Indeterminable.

8063           0   Evaluate all operations and constants just to the range and precision of the type.

8064           1   Evaluate operations and constants of type **float** and **double** to the range and precision of the  
8065           **double** type, evaluate **long double** operations and constants to the range and precision of  
8066           the **long double** type.

8067           2   Evaluate all operations and constants to the range and precision of the **long double** type.

8068           All other negative values for FLT\_EVAL\_METHOD characterize implementation-defined  
8069           behavior.

8070           The values given in the following list shall be defined as constant expressions with  
8071           implementation-defined values that are greater or equal in magnitude (absolute value) to those  
8072           shown, with the same sign.

8073           • Radix of exponent representation, *b*.

8074           FLT\_RADIX           2

8075           • Number of base-FLT\_RADIX digits in the floating-point significand, *p*.

8076           FLT\_MANT\_DIG

8077           DBL\_MANT\_DIG

8078           LDBL\_MANT\_DIG

8079           • Number of decimal digits, *n*, such that any floating-point number in the widest supported  
8080           floating type with  $p_{\max}$  radix *b* digits can be rounded to a floating-point number with *n*  
8081           decimal digits and back again without change to the value.

8082           
$$\left\{ \begin{array}{ll} p_{\max} \log_{10} b & \text{if } b \text{ is a power of } 10 \\ \left\lceil 1 + p_{\max} \log_{10} b \right\rceil & \text{otherwise} \end{array} \right.$$

8083           DECIMAL\_DIG       10

8084           • Number of decimal digits, *q*, such that any floating-point number with *q* decimal digits can  
8085           be rounded into a floating-point number with *p* radix *b* digits and back again without change  
8086           to the *q* decimal digits.

8087           
$$\left\{ \begin{array}{ll} p \log_{10} b & \text{if } b \text{ is a power of } 10 \\ \left\lceil (p - 1) \log_{10} b \right\rceil & \text{otherwise} \end{array} \right.$$

8088           FLT\_DIG           6

8089           DBL\_DIG           10

8090           LDBL\_DIG           10

8091           • Minimum negative integer such that FLT\_RADIX raised to that power minus 1 is a  
8092           normalized floating-point number,  $e_{\min}$ .

8093           FLT\_MIN\_EXP

8094           DBL\_MIN\_EXP

8095           LDBL\_MIN\_EXP

8096           • Minimum negative integer such that 10 raised to that power is in the range of normalized  
8097           floating-point numbers.

8098            $\left\lceil \log_{10} b^{e_{\min} - 1} \right\rceil$

8099           FLT\_MIN\_10\_EXP   -37

8100           DBL\_MIN\_10\_EXP   -37

8101           LDBL\_MIN\_10\_EXP   -37

8102           • Maximum integer such that FLT\_RADIX raised to that power minus 1 is a representable  
8103           finite floating-point number,  $e_{\max}$ .

8104           FLT\_MAX\_EXP

8105           DBL\_MAX\_EXP

8106           LDBL\_MAX\_EXP

8107           • Maximum integer such that 10 raised to that power is in the range of representable finite  
8108           floating-point numbers.

8109            $\left\lceil \log_{10} ((1 - b^{-p}) b^{e_{\max}}) \right\rceil$

8110           FLT\_MAX\_10\_EXP   +37

8111           DBL\_MAX\_10\_EXP   +37

8112           LDBL\_MAX\_10\_EXP   +37

8113           The values given in the following list shall be defined as constant expressions with  
8114           implementation-defined values that are greater than or equal to those shown:

8115           • Maximum representable finite floating-point number.

8116            $(1 - b^{-p}) b^{e_{\max}}$

8117           FLT\_MAX           1E+37

8118           DBL\_MAX           1E+37

8119           LDBL\_MAX          1E+37

8120           The values given in the following list shall be defined as constant expressions with  
8121           implementation-defined (positive) values that are less than or equal to those shown:

8122           • The difference between 1 and the least value greater than 1 that is representable in the given  
8123           floating-point type,  $b^{1-p}$ .

8124           FLT\_EPSILON       1E-5

8125           DBL\_EPSILON       1E-9

8126	LDBL_EPSILON	1E-9
8127	•	Minimum normalized positive floating-point number, $b^{e_{\min} - 1}$ .
8128	FLT_MIN	1E-37
8129	DBL_MIN	1E-37
8130	LDBL_MIN	1E-37
8131	<b>APPLICATION USAGE</b>	
8132	None.	
8133	<b>RATIONALE</b>	
8134	None.	
8135	<b>FUTURE DIRECTIONS</b>	
8136	None.	
8137	<b>SEE ALSO</b>	
8138	<b>&lt;complex.h&gt;</b> , <b>&lt;math.h&gt;</b>	
8139	<b>CHANGE HISTORY</b>	
8140	First released in Issue 4. Derived from the ISO C standard.	
8141	<b>Issue 6</b>	
8142	The description of the operations with floating-point values is updated for alignment with the	
8143	ISO/IEC 9899:1999 standard.	

8144 **NAME**

8145         fmtmsg.h — message display structures

8146 **SYNOPSIS**

8147 xSI       #include <fmtmsg.h>

8148

8149 **DESCRIPTION**

8150         The <fmtmsg.h> header shall define the following macros, which expand to constant integer  
8151         expressions:

- 8152         MM\_HARD                 Source of the condition is hardware.
- 8153         MM\_SOFT                Source of the condition is software.
- 8154         MM\_FIRM                Source of the condition is firmware.
- 8155         MM\_APPL                Condition detected by application.
- 8156         MM\_UTIL                Condition detected by utility.
- 8157         MM\_OPSYS               Condition detected by operating system.
- 8158         MM\_RECOVER            Recoverable error.
- 8159         MM\_NRECOV            Non-recoverable error.
- 8160         MM\_HALT                Error causing application to halt.
- 8161         MM\_ERROR               Application has encountered a non-fatal fault.
- 8162         MM\_WARNING            Application has detected unusual non-error condition.
- 8163         MM\_INFO                Informative message.
- 8164         MM\_NOSEV              No severity level provided for the message.
- 8165         MM\_PRINT               Display message on standard error.
- 8166         MM\_CONSOLE            Display message on system console.

8167         The table below indicates the null values and identifiers for *fmtmsg()* arguments. The  
8168         <fmtmsg.h> header shall define the macros in the **Identifier** column, which expand to constant  
8169         expressions that expand to expressions of the type indicated in the **Type** column:

8170

8171

Argument	Type	Null-Value	Identifier
<i>label</i>	char *	(char*)0	MM_NULLLBL
<i>severity</i>	int	0	MM_NULLSEV
<i>class</i>	long	0L	MM_NULLMC
<i>text</i>	char *	(char*)0	MM_NULLTXT
<i>action</i>	char *	(char*)0	MM_NULLACT
<i>tag</i>	char *	(char*)0	MM_NULLTAG

8172

8173

8174

8175

8176

8177

8178         The <fmtmsg.h> header shall also define the following macros for use as return values for  
8179         *fmtmsg()*:

- 8180         MM\_OK                   The function succeeded.
- 8181         MM\_NOTOK               The function failed completely.
- 8182         MM\_NOMSG               The function was unable to generate a message on standard error, but  
8183         otherwise succeeded.



8200 **NAME**

8201       fnmatch.h — filename-matching types

8202 **SYNOPSIS**

8203       #include &lt;fnmatch.h&gt;

8204 **DESCRIPTION**

8205       The &lt;fnmatch.h&gt; header shall define the following constants:

8206       FNM\_NOMATCH     The string does not match the specified pattern.

8207       FNM\_PATHNAME    Slash in *string* only matches slash in *pattern*.8208       FNM\_PERIOD      Leading period in *string* must be exactly matched by period in *pattern*.

8209       FNM\_NOESCAPE    Disable backslash escaping.

8210   OB XSI   FNM\_NOSYS     Reserved.

8211       The following shall be declared as a function and may also be defined as a macro. A function |  
8212       prototype shall be provided. |

8213       int fnmatch(const char \*, const char \*, int);

8214 **APPLICATION USAGE**

8215       None.

8216 **RATIONALE**

8217       None.

8218 **FUTURE DIRECTIONS**

8219       None.

8220 **SEE ALSO**8221       The System Interfaces volume of IEEE Std 1003.1-200x, *fnmatch()*, the Shell and Utilities volume  
8222       of IEEE Std 1003.1-200x8223 **CHANGE HISTORY**

8224       First released in Issue 4. Derived from the ISO POSIX-2 standard.

8225 **Issue 6**

8226       The constant FNM\_NOSYS is marked obsolescent.

8227 **NAME**8228 `ftw.h` — file tree traversal8229 **SYNOPSIS**8230 XSI `#include <ftw.h>`

8231

8232 **DESCRIPTION**8233 The `<ftw.h>` header shall define the **FTW** structure that includes at least the following members:8234 `int base`8235 `int level`8236 The `<ftw.h>` header shall define macros for use as values of the third argument to the  
8237 application-supplied function that is passed as the second argument to `ftw()` and `nftw()`:8238 `FTW_F` File.8239 `FTW_D` Directory.8240 `FTW_DNR` Directory without read permission.8241 `FTW_DP` Directory with subdirectories visited.8242 `FTW_NS` Unknown type; `stat()` failed.8243 `FTW_SL` Symbolic link.8244 `FTW_SLN` Symbolic link that names a nonexistent file.8245 The `<ftw.h>` header shall define macros for use as values of the fourth argument to `nftw()`:8246 `FTW_PHYS` Physical walk, does not follow symbolic links. Otherwise, `nftw()` follows  
8247 links but does not walk down any path that crosses itself.8248 `FTW_MOUNT` The walk does not cross a mount point.8249 `FTW_DEPTH` All subdirectories are visited before the directory itself.8250 `FTW_CHDIR` The walk changes to each directory before reading it.8251 The following shall be declared as functions and may also be defined as macros. Function |  
8252 prototypes shall be provided. |8253 `int ftw(const char *,`  
8254 `int (*)(const char *, const struct stat *, int), int);`  
8255 `int nftw(const char *, int (*)`  
8256 `(const char *, const struct stat *, int, struct FTW*),`  
8257 `int, int);`8258 The `<ftw.h>` header shall define the **stat** structure and the symbolic names for `st_mode` and the  
8259 file type test macros as described in `<sys/stat.h>`.8260 Inclusion of the `<ftw.h>` header may also make visible all symbols from `<sys/stat.h>`.

8261 **APPLICATION USAGE**

8262       None.

8263 **RATIONALE**

8264       None.

8265 **FUTURE DIRECTIONS**

8266       None.

8267 **SEE ALSO**8268       <sys/stat.h>, the System Interfaces volume of IEEE Std 1003.1-200x, *ftw()*, *nftw()*8269 **CHANGE HISTORY**

8270       First released in Issue 1. Derived from Issue 1 of the SVID.

8271 **Issue 5**

8272       A description of FTW\_DP is added.

8273 **NAME**

8274 glob.h — pathname pattern-matching types |

8275 **SYNOPSIS**

8276 #include &lt;glob.h&gt;

8277 **DESCRIPTION**8278 The <glob.h> header shall define the structures and symbolic constants used by the *glob()*  
8279 function.8280 The structure type **glob\_t** shall contain at least the following members:8281 size\_t gl\_pathc Count of paths matched by *pattern*.

8282 char \*\*gl\_pathv Pointer to a list of matched pathnames. |

8283 size\_t gl\_offs Slots to reserve at the beginning of *gl\_pathv*. |8284 The following constants shall be provided as values for the *flags* argument:

8285 GLOB\_APPEND Append generated pathnames to those previously obtained. |

8286 GLOB\_DOOFFS Specify how many null pointers to add to the beginning of *pglob-*  
8287 *>gl\_pathv*.8288 GLOB\_ERR Cause *glob()* to return on error.8289 GLOB\_MARK Each pathname that is a directory that matches *pattern* has a slash |  
8290 appended.8291 GLOB\_NOCHECK If *pattern* does not match any pathname, then return a list consisting of |  
8292 only *pattern*.

8293 GLOB\_NOESCAPE Disable backslash escaping.

8294 GLOB\_NOSORT Do not sort the pathnames returned. |

8295 The following constants shall be defined as error return values:

8296 GLOB\_ABORTED The scan was stopped because GLOB\_ERR was set or (*errfunc*())  
8297 returned non-zero.8298 GLOB\_NOMATCH The pattern does not match any existing pathname, and |  
8299 GLOB\_NOCHECK was not set in flags.

8300 GLOB\_NOSPACE An attempt to allocate memory failed.

8301 OB XSI GLOB\_NOSYS Reserved.

8302 The following shall be declared as functions and may also be defined as macros. Function |  
8303 prototypes shall be provided.8304 int glob(const char \*restrict, int, int (\*restrict)(const char \*, int),  
8305 glob\_t \*restrict);

8306 void globfree (glob\_t \*);

8307 The implementation may define additional macros or constants using names beginning with  
8308 GLOB\_.

8309 **APPLICATION USAGE**

8310 None.

8311 **RATIONALE**

8312 None.

8313 **FUTURE DIRECTIONS**

8314 None.

8315 **SEE ALSO**8316 The System Interfaces volume of IEEE Std 1003.1-200x, *glob()*, the Shell and Utilities volume of  
8317 IEEE Std 1003.1-200x8318 **CHANGE HISTORY**

8319 First released in Issue 4. Derived from the ISO POSIX-2 standard.

8320 **Issue 6**8321 The **restrict** keyword is added to the prototype for *glob()*.

8322 The constant GLOB\_NOSYS is marked obsolescent.

8323 **NAME**

8324 grp.h — group structure

8325 **SYNOPSIS**

8326 #include <grp.h>

8327 **DESCRIPTION**

8328 The <grp.h> header shall declare the structure **group** which shall include the following  
8329 members:

- 8330 char \*gr\_name The name of the group.
- 8331 gid\_t gr\_gid Numerical group ID.
- 8332 char \*\*gr\_mem Pointer to a null-terminated array of character  
8333 pointers to member names.

8334 The **gid\_t** type shall be defined as described in <sys/types.h>.

8335 The following shall be declared as functions and may also be defined as macros. Function  
8336 prototypes shall be provided.

```

8337 struct group *getgrgid(gid_t);
8338 struct group *getgrnam(const char *);
8339 TSF int getgrgid_r(gid_t, struct group *, char *,
8340 size_t, struct group **);
8341 int getgrnam_r(const char *, struct group *, char *,
8342 size_t, struct group **);
8343 XSI struct group *getgrent(void);
8344 void endgrent(void);
8345 void setgrent(void);
8346

```

8347 **APPLICATION USAGE**

8348 None.

8349 **RATIONALE**

8350 None.

8351 **FUTURE DIRECTIONS**

8352 None.

8353 **SEE ALSO**

8354 <sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-200x, *endgrent()*, *getgrgid()*,  
8355 *getgrnam()*

8356 **CHANGE HISTORY**

8357 First released in Issue 1.

8358 **Issue 5**

8359 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

8360 **Issue 6**

8361 The following new requirements on POSIX implementations derive from alignment with the  
8362 Single UNIX Specification:

- 8363 • The definition of **gid\_t** is mandated.
- 8364 • The *getgrgid\_r()* and *getgrnam\_r()* functions are marked as part of the Thread-Safe Functions  
8365 option.

8366 **NAME**

8367 iconv.h — codeset conversion facility

8368 **SYNOPSIS**8369 XSI `#include <iconv.h>`

8370

8371 **DESCRIPTION**

8372 The &lt;iconv.h&gt; header shall define the following type:

8373 **iconv\_t** Identifies the conversion from one codeset to another.8374 The following shall be declared as functions and may also be defined as macros. Function |  
8375 prototypes shall be provided. |8376 `iconv_t iconv_open(const char *, const char *);`8377 `size_t iconv(iconv_t, char **restrict, size_t *restrict, char **restrict,`  
8378 `size_t *restrict);`8379 `int iconv_close(iconv_t);`8380 **APPLICATION USAGE**

8381 None.

8382 **RATIONALE**

8383 None.

8384 **FUTURE DIRECTIONS**

8385 None.

8386 **SEE ALSO**8387 The System Interfaces volume of IEEE Std 1003.1-200x, *iconv()*, *iconv\_close()*, *iconv\_open()*8388 **CHANGE HISTORY**

8389 First released in Issue 4.

8390 **Issue 6**8391 The **restrict** keyword is added to the prototype for *iconv()*.

8392 NAME

8393 inttypes.h — fixed size integer types

8394 SYNOPSIS

8395 #include <inttypes.h>

8396 DESCRIPTION

8397 cx Some of the functionality described on this reference page extends the ISO C standard. Applications shall define the appropriate feature test macro (see the System Interfaces volume of IEEE Std 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of these symbols in this header.

8401 The <inttypes.h> header shall include the <stdint.h> header.

8402 The <inttypes.h> header shall include a definition of at least the following type:

8403 **imaxdiv\_t** Structure type that is the type of the value returned by the *imaxdiv()* function.

8404 The following macros shall be defined. Each expands to a character string literal containing a conversion specifier, possibly modified by a length modifier, suitable for use within the *format* argument of a formatted input/output function when converting the corresponding integer type. These macros have the general form of PRI (character string literals for the *fprintf()* and *fwprintf()* family of functions) or SCN (character string literals for the *fscanf()* and *fwscanf()* family of functions), followed by the conversion specifier, followed by a name corresponding to a similar type name in <stdint.h>. In these names, *N* represents the width of the type as described in <stdint.h>. For example, *PRIdFAST32* can be used in a format string to print the value of an integer of type **int\_fast32\_t**.

8413 The *fprintf()* macros for signed integers are:

8414	PRIdN	PRIdLEASTN	PRIdFASTN	PRIdMAX	PRIdPTR
8415	PRiN	PRiLEASTN	PRiFASTN	PRiMAX	PRiPTR

8416 The *fprintf()* macros for unsigned integers are:

8417	PRIoN	PRIoLEASTN	PRIoFASTN	PRIoMAX	PRIoPTR
8418	PRiUN	PRiULEASTN	PRiUFASTN	PRiUMAX	PRiUPTR
8419	PRIxN	PRiXLEASTN	PRiXFASTN	PRiXMAX	PRiXPTR
8420	PRiXN	PRiXLEASTN	PRiXFASTN	PRiXMAX	PRiXPTR

8421 The *fscanf()* macros for signed integers are:

8422	SCNdN	SCNdLEASTN	SCNdFASTN	SCNdMAX	SCNdPTR
8423	SCNiN	SCNiLEASTN	SCNiFASTN	SCNiMAX	SCNiPTR

8424 The *fscanf()* macros for unsigned integers are:

8425	SCNoN	SCNoLEASTN	SCNoFASTN	SCNoMAX	SCNoPTR
8426	SCNuN	SCNuLEASTN	SCNuFASTN	SCNuMAX	SCNuPTR
8427	SCNxN	SCNxLEASTN	SCNxFASTN	SCNxMAX	SCNxPTR

8428 For each type that the implementation provides in <stdint.h>, the corresponding *fprintf()* macros shall be defined and the corresponding *fscanf()* macros shall be defined unless the implementation does not have a suitable *fscanf* length modifier for the type.

8431 The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided. |

```
8433 intmax_t imaxabs(intmax_t);
8434 imaxdiv_t imaxdiv(intmax_t, intmax_t);
8435 intmax_t strtoumax(const char *restrict, char **restrict, int);
```

```

8436     uintmax_t strtoumax(const char *restrict, char **restrict, int);
8437     intmax_t wcstoimax(const wchar_t *restrict, wchar_t **restrict, int);
8438     uintmax_t wcstoumax(const wchar_t *restrict, wchar_t **restrict, int);

```

**8439 EXAMPLES**

```

8440     #include <inttypes.h>
8441     #include <wchar.h>
8442     int main(void)
8443     {
8444         uintmax_t i = UINTMAX_MAX; // This type always exists.
8445         wprintf(L"The largest integer value is %020"
8446             PRIxMAX "\n", i);
8447         return 0;
8448     }

```

**8449 APPLICATION USAGE**

8450 None.

**8451 RATIONALE**

8452 The ISO/IEC 9899:1990 standard specifies that the language should support four signed and  
8453 unsigned integer data types—**char**, **short**, **int**, and **long**—but places very little requirement on  
8454 their size other than that **int** and **short** be at least 16 bits and **long** be at least as long as **int** and  
8455 not smaller than 32 bits. For 16-bit systems, most implementations assign 8, 16, 16, and 32 bits to  
8456 **char**, **short**, **int**, and **long**, respectively. For 32-bit systems, the common practice is to assign 8, 16,  
8457 32, and 32 bits to these types. This difference in **int** size can create some problems for users who  
8458 migrate from one system to another which assigns different sizes to integer types, because the  
8459 ISO C standard integer promotion rule can produce silent changes unexpectedly. The need for  
8460 defining an extended integer type increased with the introduction of 64-bit systems.

8461 The purpose of <inttypes.h> is to provide a set of integer types whose definitions are consistent  
8462 across machines and independent of operating systems and other implementation  
8463 idiosyncrasies. It defines, via **typedef**, integer types of various sizes. Implementations are free to  
8464 **typedef** them as ISO C standard integer types or extensions that they support. Consistent use of  
8465 this header will greatly increase the portability of a users program across platforms.

**8466 FUTURE DIRECTIONS**

8467 Macro names beginning with PRI or SCN followed by any lowercase letter or 'x' may be added  
8468 to the macros defined in the <inttypes.h> header.

**8469 SEE ALSO**

8470 The System Interfaces volume of IEEE Std 1003.1-200x, *imaxdiv()*

**8471 CHANGE HISTORY**

8472 First released in Issue 5.

**8473 Issue 6**

8474 The Open Group Base Resolution bwg97-006 is applied.

8475 This reference page is updated to align with the ISO/IEC 9899:1999 standard.

8476 **NAME**

8477        iso646.h — alternative spellings

8478 **SYNOPSIS**

8479        #include &lt;iso646.h&gt;

8480 **DESCRIPTION**

8481 *cx*       The functionality described on this reference page is aligned with the ISO C standard. Any  
8482 conflict between the requirements described here and the ISO C standard is unintentional. This  
8483 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

8484        The **<iso646.h>** header shall define the following eleven macros (on the left) that expand to the  
8485 corresponding tokens (on the right):

8486        *and*            &&8487        *and\_eq*         &=8488        *bitand*         &8489        *bitor*           |8490        *compl*         ~8491        *not*             !8492        *not\_eq*        !=8493        *or*             | |8494        *or\_eq*         |=8495        *xor*             ^8496        *xor\_eq*        ^=8497 **APPLICATION USAGE**

8498        None.

8499 **RATIONALE**

8500        None.

8501 **FUTURE DIRECTIONS**

8502        None.

8503 **SEE ALSO**

8504        None.

8505 **CHANGE HISTORY**

8506        First released in Issue 5. Derived from ISO/IEC 9899:1990/Amendment 1:1995 (E).

8507 **NAME**

8508 langinfo.h — language information constants

8509 **SYNOPSIS**

8510 xSI #include <langinfo.h>

8511

8512 **DESCRIPTION**

8513 The <langinfo.h> header contains the constants used to identify items of *langinfo* data (see  
8514 *nl\_langinfo()*). The type of the constant, **nl\_item**, shall be defined as described in <nl\_types.h>.

8515 The following constants shall be defined. The entries under **Category** indicate in which  
8516 *setlocale()* category each item is defined.

8517

Constant	Category	Meaning
8519 CODESET	<i>LC_CTYPE</i>	Codeset name.
8520 D_T_FMT	<i>LC_TIME</i>	String for formatting date and time.
8521 D_FMT	<i>LC_TIME</i>	Date format string.
8522 T_FMT	<i>LC_TIME</i>	Time format string.
8523 T_FMT_AMPM	<i>LC_TIME</i>	a.m. or p.m. time format string.
8524 AM_STR	<i>LC_TIME</i>	Ante Meridian affix.
8525 PM_STR	<i>LC_TIME</i>	Post Meridian affix.
8526 DAY_1	<i>LC_TIME</i>	Name of the first day of the week (for example, Sunday).
8527 DAY_2	<i>LC_TIME</i>	Name of the second day of the week (for example, Monday).
8528 DAY_3	<i>LC_TIME</i>	Name of the third day of the week (for example, Tuesday).
8529 DAY_4	<i>LC_TIME</i>	Name of the fourth day of the week (for example, Wednesday).
8530		
8531 DAY_5	<i>LC_TIME</i>	Name of the fifth day of the week (for example, Thursday).
8532 DAY_6	<i>LC_TIME</i>	Name of the sixth day of the week (for example, Friday).
8533 DAY_7	<i>LC_TIME</i>	Name of the seventh day of the week (for example, Saturday).
8534		
8535 ABDAY_1	<i>LC_TIME</i>	Abbreviated name of the first day of the week.
8536 ABDAY_2	<i>LC_TIME</i>	Abbreviated name of the second day of the week.
8537 ABDAY_3	<i>LC_TIME</i>	Abbreviated name of the third day of the week.
8538 ABDAY_4	<i>LC_TIME</i>	Abbreviated name of the fourth day of the week.
8539 ABDAY_5	<i>LC_TIME</i>	Abbreviated name of the fifth day of the week.
8540 ABDAY_6	<i>LC_TIME</i>	Abbreviated name of the sixth day of the week.
8541 ABDAY_7	<i>LC_TIME</i>	Abbreviated name of the seventh day of the week.
8542 MON_1	<i>LC_TIME</i>	Name of the first month of the year.
8543 MON_2	<i>LC_TIME</i>	Name of the second month.
8544 MON_3	<i>LC_TIME</i>	Name of the third month.
8545 MON_4	<i>LC_TIME</i>	Name of the fourth month.
8546 MON_5	<i>LC_TIME</i>	Name of the fifth month.
8547 MON_6	<i>LC_TIME</i>	Name of the sixth month.
8548 MON_7	<i>LC_TIME</i>	Name of the seventh month.
8549 MON_8	<i>LC_TIME</i>	Name of the eighth month.
8550 MON_9	<i>LC_TIME</i>	Name of the ninth month.
8551 MON_10	<i>LC_TIME</i>	Name of the tenth month.
8552 MON_11	<i>LC_TIME</i>	Name of the eleventh month.
8553 MON_12	<i>LC_TIME</i>	Name of the twelfth month.

8554  
8555  
8556  
8557  
8558  
8559  
8560  
8561  
8562  
8563  
8564  
8565  
8566  
8567  
8568  
8569  
8570  
8571  
8572  
8573  
8574  
8575  
8576  
8577  
8578  
8579  
8580

Constant	Category	Meaning
ABMON_1	LC_TIME	Abbreviated name of the first month.
ABMON_2	LC_TIME	Abbreviated name of the second month.
ABMON_3	LC_TIME	Abbreviated name of the third month.
ABMON_4	LC_TIME	Abbreviated name of the fourth month.
ABMON_5	LC_TIME	Abbreviated name of the fifth month.
ABMON_6	LC_TIME	Abbreviated name of the sixth month.
ABMON_7	LC_TIME	Abbreviated name of the seventh month.
ABMON_8	LC_TIME	Abbreviated name of the eighth month.
ABMON_9	LC_TIME	Abbreviated name of the ninth month.
ABMON_10	LC_TIME	Abbreviated name of the tenth month.
ABMON_11	LC_TIME	Abbreviated name of the eleventh month.
ABMON_12	LC_TIME	Abbreviated name of the twelfth month.
ERA	LC_TIME	Era description segments.
ERA_D_FMT	LC_TIME	Era date format string.
ERA_D_T_FMT	LC_TIME	Era date and time format string.
ERA_T_FMT	LC_TIME	Era time format string.
ALT_DIGITS	LC_TIME	Alternative symbols for digits.
RADIXCHAR	LC_NUMERIC	Radix character.
THOUSEP	LC_NUMERIC	Separator for thousands.
YESEXPR	LC_MESSAGES	Affirmative response expression.
NOEXPR	LC_MESSAGES	Negative response expression.
CRNCYSTR	LC_MONETARY	Currency symbol, preceded by '-' if the symbol should appear before the value, '+' if the symbol should appear after the value, or '.' if the symbol should replace the radix character.

8581 If the locale's values for **p\_cs\_precedes** and **n\_cs\_precedes** do not match, the value of  
8582 *nl\_langinfo*(CRNCYSTR) is unspecified.

8583 The following shall be declared as a function and may also be defined as a macro. A function  
8584 prototype shall be provided.

```
8585 char *nl_langinfo(nl_item);
```

8586 Inclusion of the <langinfo.h> header may also make visible all symbols from <nl\_types.h>.

8587 **APPLICATION USAGE**

8588 Wherever possible, users are advised to use functions compatible with those in the ISO C  
8589 standard to access items of *langinfo* data. In particular, the *strptime*() function should be used to  
8590 access date and time information defined in category *LC\_TIME*. The *localeconv*() function  
8591 should be used to access information corresponding to *RADIXCHAR*, *THOUSEP*, and  
8592 *CRNCYSTR*.

8593 **RATIONALE**

8594 None.

8595 **FUTURE DIRECTIONS**

8596 None.

8597 **SEE ALSO**

8598 The System Interfaces volume of IEEE Std 1003.1-200x, *nl\_langinfo*(), *localeconv*(), *strfmon*(),  
8599 *strptime*(), Chapter 7 (on page 119)

8600 **CHANGE HISTORY**

8601 First released in Issue 2.

8602 **Issue 5**

8603 The constants YESSTR and NOSTR are marked LEGACY.

8604 **Issue 6**

8605 The constants YESSTR and NOSTR are removed.

8606 **NAME**

8607 libgen.h — definitions for pattern matching functions

8608 **SYNOPSIS**

8609 XSI #include &lt;libgen.h&gt;

8610

8611 **DESCRIPTION**8612 The following shall be declared as functions and may also be defined as macros. Function |  
8613 prototypes shall be provided. |

8614 char \*basename(char \*);

8615 char \*dirname(char \*);

8616 **APPLICATION USAGE**

8617 None.

8618 **RATIONALE**

8619 None.

8620 **FUTURE DIRECTIONS**

8621 None.

8622 **SEE ALSO**8623 The System Interfaces volume of IEEE Std 1003.1-200x, *basename()*, *dirname()*8624 **CHANGE HISTORY**

8625 First released in Issue 4, Version 2.

8626 **Issue 5**8627 The function prototypes for *basename()* and *dirname()* are changed to indicate that the first  
8628 argument is of type **char \*** rather than **const char \***.8629 **Issue 6**8630 The **\_\_loc1** symbol and the *regcmp()* and *regex()* functions are removed.

8631 **NAME**

8632 limits.h — implementation-defined constants

8633 **SYNOPSIS**

8634 #include <limits.h>

8635 **DESCRIPTION**

8636 CX Some of the functionality described on this reference page extends the ISO C standard.  
 8637 Applications shall define the appropriate feature test macro (see the System Interfaces volume of  
 8638 IEEE Std 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of these  
 8639 symbols in this header.

8640 CX Many of the symbols listed here are not defined by the ISO/IEC 9899:1999 standard. Such  
 8641 symbols are not shown as CX shaded.

8642 The <limits.h> header shall define various symbolic names. Different categories of names are  
 8643 described below.

8644 The names represent various limits on resources that the implementation imposes on  
 8645 applications.

8646 Implementations may choose any appropriate value for each limit, provided it is not more  
 8647 restrictive than the Minimum Acceptable Values listed below. Symbolic constant names  
 8648 beginning with \_POSIX may be found in <unistd.h>.

8649 Applications should not assume any particular value for a limit. To achieve maximum  
 8650 portability, an application should not require more resource than the Minimum Acceptable  
 8651 Value quantity. However, an application wishing to avail itself of the full amount of a resource  
 8652 available on an implementation may make use of the value given in <limits.h> on that  
 8653 particular implementation, by using the symbolic names listed below. It should be noted,  
 8654 however, that many of the listed limits are not invariant, and at runtime, the value of the limit  
 8655 may differ from those given in this header, for the following reasons:

- 8656 • The limit is pathname-dependent.
- 8657 • The limit differs between the compile and runtime machines.

8658 For these reasons, an application may use the *fpathconf()*, *pathconf()*, and *sysconf()* functions to  
 8659 determine the actual value of a limit at runtime.

8660 The items in the list ending in \_MIN give the most negative values that the mathematical types  
 8661 are guaranteed to be capable of representing. Numbers of a more negative value may be  
 8662 supported on some implementations, as indicated by the <limits.h> header on the  
 8663 implementation, but applications requiring such numbers are not guaranteed to be portable to  
 8664 all implementations. For positive constants ending in \_MIN, this indicates the minimum  
 8665 acceptable value.

8666 The Minimum Acceptable Value symbol ' \* ' indicates that there is no guaranteed value across  
 8667 all conforming implementations.

8668 **Runtime Invariant Values (Possibly Indeterminate)**

8669 A definition of one of the symbolic names in the following list shall be omitted from **<limits.h>**  
8670 on specific implementations where the corresponding value is equal to or greater than the stated  
8671 minimum, but is unspecified.

8672 This indetermination might depend on the amount of available memory space on a specific  
8673 instance of a specific implementation. The actual value supported by a specific instance shall be  
8674 provided by the *sysconf()* function.

8675 AIO {AIO\_LISTIO\_MAX}  
8676 Maximum number of I/O operations in a single list I/O call supported by the  
8677 implementation.  
8678 Minimum Acceptable Value: {\_POSIX\_AIO\_LISTIO\_MAX}

8679 AIO {AIO\_MAX}  
8680 Maximum number of outstanding asynchronous I/O operations supported by the  
8681 implementation.  
8682 Minimum Acceptable Value: {\_POSIX\_AIO\_MAX}

8683 AIO {AIO\_PRIO\_DELTA\_MAX}  
8684 The maximum amount by which a process can decrease its asynchronous I/O priority level  
8685 from its own scheduling priority.  
8686 Minimum Acceptable Value: 0

8687 {ARG\_MAX}  
8688 Maximum length of argument to the *exec* functions including environment data.  
8689 Minimum Acceptable Value: {\_POSIX\_ARG\_MAX}

8690 XSI {ATEXIT\_MAX}  
8691 Maximum number of functions that may be registered with *atexit()*.  
8692 Minimum Acceptable Value: 32

8693 {CHILD\_MAX}  
8694 Maximum number of simultaneous processes per real user ID.  
8695 Minimum Acceptable Value: {\_POSIX\_CHILD\_MAX}

8696 TMR {DELAYTIMER\_MAX}  
8697 Maximum number of timer expiration overruns.  
8698 Minimum Acceptable Value: {\_POSIX\_DELAYTIMER\_MAX}

8699 {HOST\_NAME\_MAX}  
8700 Maximum length of a host name (not including the terminating null) as returned from the  
8701 *gethostname()* function.  
8702 Minimum Acceptable Value: {\_POSIX\_HOST\_NAME\_MAX}

8703 XSI {IOV\_MAX}  
8704 Maximum number of *iovec* structures that one process has available for use with *readv()* or  
8705 *writev()*.  
8706 Minimum Acceptable Value: {\_XOPEN\_IOV\_MAX}

8707 {LOGIN\_NAME\_MAX}  
8708 Maximum length of a login name.  
8709 Minimum Acceptable Value: {\_POSIX\_LOGIN\_NAME\_MAX}

8710 MSG {MQ\_OPEN\_MAX}  
8711 The maximum number of open message queue descriptors a process may hold.  
8712 Minimum Acceptable Value: {\_POSIX\_MQ\_OPEN\_MAX}

8713	MSG	{MQ_PRIO_MAX}
8714		The maximum number of message priorities supported by the implementation.
8715		Minimum Acceptable Value: {_POSIX_MQ_PRIO_MAX}
8716		{OPEN_MAX}
8717		Maximum number of files that one process can have open at any one time.
8718		Minimum Acceptable Value: {_POSIX_OPEN_MAX}
8719		{PAGESIZE}
8720		Size in bytes of a page.
8721		Minimum Acceptable Value: 1
8722	XSI	{PAGE_SIZE}
8723		Equivalent to {PAGESIZE}. If either {PAGESIZE} or {PAGE_SIZE} is defined, the other is
8724		defined with the same value.
8725	THR	{PTHREAD_DESTRUCTOR_ITERATIONS}
8726		Maximum number of attempts made to destroy a thread's thread-specific data values on
8727		thread exit.
8728		Minimum Acceptable Value: {_POSIX_THREAD_DESTRUCTOR_ITERATIONS}
8729	THR	{PTHREAD_KEYS_MAX}
8730		Maximum number of data keys that can be created by a process.
8731		Minimum Acceptable Value: {_POSIX_THREAD_KEYS_MAX}
8732	THR	{PTHREAD_STACK_MIN}
8733		Minimum size in bytes of thread stack storage.
8734		Minimum Acceptable Value: 0
8735	THR	{PTHREAD_THREADS_MAX}
8736		Maximum number of threads that can be created per process.
8737		Minimum Acceptable Value: {_POSIX_THREAD_THREADS_MAX}
8738		{RE_DUP_MAX}
8739		The number of repeated occurrences of a BRE permitted by the <i>regex()</i> and <i>regcomp()</i>
8740		functions when using the interval notation $\{m,n\}$ ; see Section 9.3.6 (on page 170).
8741		Minimum Acceptable Value: {_POSIX2_RE_DUP_MAX}
8742	RTS	{RTSIG_MAX}
8743		Maximum number of realtime signals reserved for application use in this implementation.
8744		Minimum Acceptable Value: {_POSIX_RTSIG_MAX}
8745	SEM	{SEM_NSEMS_MAX}
8746		Maximum number of semaphores that a process may have.
8747		Minimum Acceptable Value: {_POSIX_SEM_NSEMS_MAX}
8748	SEM	{SEM_VALUE_MAX}
8749		The maximum value a semaphore may have.
8750		Minimum Acceptable Value: {_POSIX_SEM_VALUE_MAX}
8751	RTS	{SIGQUEUE_MAX}
8752		Maximum number of queued signals that a process may send and have pending at the
8753		receiver(s) at any time.
8754		Minimum Acceptable Value: {_POSIX_SIGQUEUE_MAX}
8755	SS TSP	{SS_REPL_MAX}
8756		The maximum number of replenishment operations that may be simultaneously pending
8757		for a particular sporadic server scheduler.
8758		Minimum Acceptable Value: {_POSIX_SS_REPL_MAX}

8759 {STREAM\_MAX}  
8760 The number of streams that one process can have open at one time. If defined, it has the  
8761 same value as {FOPEN\_MAX} (see <stdio.h>).  
8762 Minimum Acceptable Value: {\_POSIX\_STREAM\_MAX}

8763 {SYMLOOP\_MAX}  
8764 Maximum number of symbolic links that can be reliably traversed in the resolution of a  
8765 pathname in the absence of a loop.  
8766 Minimum Acceptable Value: {\_POSIX\_SYMLOOP\_MAX}

8767 TMR {TIMER\_MAX}  
8768 Maximum number of timers per-process supported by the implementation.  
8769 Minimum Acceptable Value: {\_POSIX\_TIMER\_MAX}

8770 TRC {TRACE\_EVENT\_NAME\_MAX}  
8771 Maximum length of the trace event name.  
8772 Minimum Acceptable Value: {\_POSIX\_TRACE\_EVENT\_NAME\_MAX}

8773 TRC {TRACE\_NAME\_MAX}  
8774 Maximum length of the trace generation version string or of the trace stream name.  
8775 Minimum Acceptable Value: {\_POSIX\_TRACE\_NAME\_MAX}

8776 TRC {TRACE\_SYS\_MAX}  
8777 Maximum number of trace streams that may simultaneously exist in the system.  
8778 Minimum Acceptable Value: {\_POSIX\_TRACE\_SYS\_MAX}

8779 TRC {TRACE\_USER\_EVENT\_MAX}  
8780 Maximum number of user trace event type identifiers that may simultaneously exist in a  
8781 traced process, including the predefined user trace event  
8782 POSIX\_TRACE\_UNNAMED\_USER\_EVENT.  
8783 Minimum Acceptable Value: {\_POSIX\_TRACE\_USER\_EVENT\_MAX}

8784 {TTY\_NAME\_MAX}  
8785 Maximum length of terminal device name.  
8786 Minimum Acceptable Value: {\_POSIX\_TTY\_NAME\_MAX}

8787 {TZNAME\_MAX}  
8788 Maximum number of bytes supported for the name of a timezone (not of the TZ variable).  
8789 Minimum Acceptable Value: {\_POSIX\_TZNAME\_MAX}

8790 **Note:** The length given by {TZNAME\_MAX} does not include the quoting characters mentioned in  
8791 Section 8.3 (on page 161).

### 8792 Pathname Variable Values

8793 The values in the following list may be constants within an implementation or may vary from  
8794 one pathname to another. For example, file systems or directories may have different  
8795 characteristics.

8796 A definition of one of the values shall be omitted from the <limits.h> header on specific  
8797 implementations where the corresponding value is equal to or greater than the stated minimum,  
8798 but where the value can vary depending on the file to which it is applied. The actual value  
8799 supported for a specific pathname shall be provided by the *pathconf()* function.

8800 {FILESIZEBITS}  
8801 Minimum number of bits needed to represent, as a signed integer value, the maximum size  
8802 of a regular file allowed in the specified directory.  
8803 Minimum Acceptable Value: 32

8804 {LINK\_MAX}  
 8805 Maximum number of links to a single file.  
 8806 Minimum Acceptable Value: {\_POSIX\_LINK\_MAX}

8807 {MAX\_CANON}  
 8808 Maximum number of bytes in a terminal canonical input line.  
 8809 Minimum Acceptable Value: {\_POSIX\_MAX\_CANON}

8810 {MAX\_INPUT}  
 8811 Minimum number of bytes for which space is available in a terminal input queue; therefore, |  
 8812 the maximum number of bytes a conforming application may require to be typed as input |  
 8813 before reading them.  
 8814 Minimum Acceptable Value: {\_POSIX\_MAX\_INPUT}

8815 {NAME\_MAX}  
 8816 Maximum number of bytes in a filename (not including terminating null).  
 8817 Minimum Acceptable Value: {\_POSIX\_NAME\_MAX}  
 8818 XSI Minimum Acceptable Value: {\_XOPEN\_NAME\_MAX}

8819 {PATH\_MAX}  
 8820 Maximum number of bytes in a pathname, including the terminating null character. |  
 8821 Minimum Acceptable Value: {\_POSIX\_PATH\_MAX}  
 8822 XSI Minimum Acceptable Value: {\_XOPEN\_PATH\_MAX}

8823 {PIPE\_BUF}  
 8824 Maximum number of bytes that is guaranteed to be atomic when writing to a pipe.  
 8825 Minimum Acceptable Value: {\_POSIX\_PIPE\_BUF}

8826 ADV {POSIX\_ALLOC\_SIZE\_MIN}  
 8827 Minimum number of bytes of storage actually allocated for any portion of a file.  
 8828 Minimum Acceptable Value: Not specified.

8829 ADV {POSIX\_REC\_INCR\_XFER\_SIZE}  
 8830 Recommended increment for file transfer sizes between the  
 8831 {POSIX\_REC\_MIN\_XFER\_SIZE} and {POSIX\_REC\_MAX\_XFER\_SIZE} values.  
 8832 Minimum Acceptable Value: Not specified.

8833 ADV {POSIX\_REC\_MAX\_XFER\_SIZE}  
 8834 Maximum recommended file transfer size.  
 8835 Minimum Acceptable Value: Not specified.

8836 ADV {POSIX\_REC\_MIN\_XFER\_SIZE}  
 8837 Minimum recommended file transfer size.  
 8838 Minimum Acceptable Value: Not specified.

8839 ADV {POSIX\_REC\_XFER\_ALIGN}  
 8840 Recommended file transfer buffer alignment.  
 8841 Minimum Acceptable Value: Not specified.

8842 {SYMLINK\_MAX}  
 8843 Maximum number of bytes in a symbolic link.  
 8844 Minimum Acceptable Value: {\_POSIX\_SYMLINK\_MAX}

8845        **Runtime Inceasable Values**

8846        The magnitude limitations in the following list shall be fixed by specific implementations. An  
8847        application should assume that the value supplied by **<limits.h>** in a specific implementation is  
8848        the minimum that pertains whenever the application is run under that implementation. A  
8849        specific instance of a specific implementation may increase the value relative to that supplied by  
8850        **<limits.h>** for that implementation. The actual value supported by a specific instance shall be  
8851        provided by the *sysconf()* function.

8852        {BC\_BASE\_MAX}

8853               Maximum *obase* values allowed by the *bc* utility.  
8854               Minimum Acceptable Value: {\_POSIX2\_BC\_BASE\_MAX}

8855        {BC\_DIM\_MAX}

8856               Maximum number of elements permitted in an array by the *bc* utility.  
8857               Minimum Acceptable Value: {\_POSIX2\_BC\_DIM\_MAX}

8858        {BC\_SCALE\_MAX}

8859               Maximum *scale* value allowed by the *bc* utility.  
8860               Minimum Acceptable Value: {\_POSIX2\_BC\_SCALE\_MAX}

8861        {BC\_STRING\_MAX}

8862               Maximum length of a string constant accepted by the *bc* utility.  
8863               Minimum Acceptable Value: {\_POSIX2\_BC\_STRING\_MAX}

8864        {CHARCLASS\_NAME\_MAX}

8865               Maximum number of bytes in a character class name.  
8866               Minimum Acceptable Value: {\_POSIX2\_CHARCLASS\_NAME\_MAX}

8867        {COLL\_WEIGHTS\_MAX}

8868               Maximum number of weights that can be assigned to an entry of the *LC\_COLLATE* **order**  
8869               keyword in the locale definition file; see Chapter 7 (on page 119).  
8870               Minimum Acceptable Value: {\_POSIX2\_COLL\_WEIGHTS\_MAX}

8871        {EXPR\_NEST\_MAX}

8872               Maximum number of expressions that can be nested within parentheses by the *expr* utility.  
8873               Minimum Acceptable Value: {\_POSIX2\_EXPR\_NEST\_MAX}

8874        {LINE\_MAX}

8875               Unless otherwise noted, the maximum length, in bytes, of a utility's input line (either  
8876               standard input or another file), when the utility is described as processing text files. The  
8877               length includes room for the trailing newline.  
8878               Minimum Acceptable Value: {\_POSIX2\_LINE\_MAX}

8879        {NGROUPS\_MAX}

8880               Maximum number of simultaneous supplementary group IDs per process.  
8881               Minimum Acceptable Value: {\_POSIX2\_NGROUPS\_MAX}

8882        {RE\_DUP\_MAX}

8883               Maximum number of repeated occurrences of a regular expression permitted when using  
8884               the interval notation  $\{m,n\}$ ; see Chapter 9 (on page 165).  
8885               Minimum Acceptable Value: {\_POSIX2\_RE\_DUP\_MAX}

8886 **Maximum Values**

8887 TMR The symbolic constants in the following list shall be defined in <limits.h> with the values  
 8888 shown. These are symbolic names for the most restrictive value for certain features on an  
 8889 implementation supporting the Timers option. A conforming implementation shall provide  
 8890 values no larger than these values. A conforming application must not require a smaller value  
 8891 for correct operation.

8892 TMR `{_POSIX_CLOCKRES_MIN}`  
 8893 The resolution of the `CLOCK_REALTIME` clock, in nanoseconds.  
 8894 Value: 20 000 000

8895 MON If the Monotonic Clock option is supported, the resolution of the `CLOCK_MONOTONIC`  
 8896 clock, in nanoseconds, is represented by `{_POSIX_CLOCKRES_MIN}`.

8897 **Minimum Values**

8898 The symbolic constants in the following list shall be defined in <limits.h> with the values  
 8899 shown. These are symbolic names for the most restrictive value for certain features on an  
 8900 implementation conforming to this volume of IEEE Std 1003.1-200x. Related symbolic constants  
 8901 are defined elsewhere in this volume of IEEE Std 1003.1-200x which reflect the actual  
 8902 implementation and which need not be as restrictive. A conforming implementation shall  
 8903 provide values at least this large. A strictly conforming application must not require a larger  
 8904 value for correct operation.

8905 AIO `{_POSIX_AIO_LISTIO_MAX}`  
 8906 The number of I/O operations that can be specified in a list I/O call.  
 8907 Value: 2

8908 AIO `{_POSIX_AIO_MAX}`  
 8909 The number of outstanding asynchronous I/O operations.  
 8910 Value: 1

8911 `{_POSIX_ARG_MAX}`  
 8912 Maximum length of argument to the `exec` functions including environment data.  
 8913 Value: 4 096

8914 `{_POSIX_CHILD_MAX}`  
 8915 Maximum number of simultaneous processes per real user ID.  
 8916 Value: 6

8917 TMR `{_POSIX_DELAYTIMER_MAX}`  
 8918 The number of timer expiration overruns.  
 8919 Value: 32

8920 `{_POSIX_HOST_NAME_MAX}`  
 8921 Maximum length of a host name (not including the terminating null) as returned from the  
 8922 `gethostname()` function.  
 8923 Value: 255

8924 `{_POSIX_LINK_MAX}`  
 8925 Maximum number of links to a single file.  
 8926 Value: 8

8927 `{_POSIX_LOGIN_NAME_MAX}`  
 8928 The size of the storage required for a login name, in bytes, including the terminating null.  
 8929 Value: 9

8930            { \_POSIX\_MAX\_CANON}  
8931            Maximum number of bytes in a terminal canonical input queue.  
8932            Value: 255

8933            { \_POSIX\_MAX\_INPUT}  
8934            Maximum number of bytes allowed in a terminal input queue.  
8935            Value: 255

8936 MSG        { \_POSIX\_MQ\_OPEN\_MAX}  
8937            The number of message queues that can be open for a single process.  
8938            Value: 8

8939 MSG        { \_POSIX\_MQ\_PRIO\_MAX}  
8940            The maximum number of message priorities supported by the implementation.  
8941            Value: 32

8942            { \_POSIX\_NAME\_MAX}  
8943            Maximum number of bytes in a filename (not including terminating null).  
8944            Value: 14

8945            { \_POSIX\_NGROUPS\_MAX}  
8946            Maximum number of simultaneous supplementary group IDs per process.  
8947            Value: 8

8948            { \_POSIX\_OPEN\_MAX}  
8949            Maximum number of files that one process can have open at any one time.  
8950            Value: 20

8951            { \_POSIX\_PATH\_MAX}  
8952            Maximum number of bytes in a pathname. |  
8953            Value: 256

8954            { \_POSIX\_PIPE\_BUF}  
8955            Maximum number of bytes that is guaranteed to be atomic when writing to a pipe.  
8956            Value: 512

8957            { \_POSIX\_RE\_DUP\_MAX}  
8958            The number of repeated occurrences of a BRE permitted by the *regex*(*)* and *regcomp*(*)*  
8959            functions when using the interval notation *{\ (m,n)\}*; see Section 9.3.6 (on page 170).  
8960            Value: 255

8961 RTS        { \_POSIX\_RTSIG\_MAX}  
8962            The number of realtime signal numbers reserved for application use.  
8963            Value: 8

8964 SEM        { \_POSIX\_SEM\_NSEMS\_MAX}  
8965            The number of semaphores that a process may have.  
8966            Value: 256

8967 SEM        { \_POSIX\_SEM\_VALUE\_MAX}  
8968            The maximum value a semaphore may have.  
8969            Value: 32 767

8970 RTS        { \_POSIX\_SIGQUEUE\_MAX}  
8971            The number of queued signals that a process may send and have pending at the receiver(s)  
8972            at any time.  
8973            Value: 32

8974		{_POSIX_SSIZE_MAX}
8975		The value that can be stored in an object of type <code>ssize_t</code> .
8976		Value: 32 767
8977		{_POSIX_STREAM_MAX}
8978		The number of streams that one process can have open at one time.
8979		Value: 8
8980	SS TSP	{_POSIX_SS_REPL_MAX}
8981		The number of replenishment operations that may be simultaneously pending for a particular sporadic server scheduler.
8982		Value: 4
8983		
8984		{_POSIX_SYMLINK_MAX}
8985		The number of bytes in a symbolic link.
8986		Value: 255
8987		{_POSIX_SYMLOOP_MAX}
8988		The number of symbolic links that can be traversed in the resolution of a pathname in the absence of a loop.
8989		Value: 8
8990		
8991	THR	{_POSIX_THREAD_DESTRUCTOR_ITERATIONS}
8992		The number of attempts made to destroy a thread's thread-specific data values on thread exit.
8993		Value: 4
8994		
8995	THR	{_POSIX_THREAD_KEYS_MAX}
8996		The number of data keys per process.
8997		Value: 128
8998	THR	{_POSIX_THREAD_THREADS_MAX}
8999		The number of threads per process.
9000		Value: 64
9001	TMR	{_POSIX_TIMER_MAX}
9002		The per process number of timers.
9003		Value: 32
9004	TRC	{_POSIX_TRACE_EVENT_NAME_MAX}
9005		The length in bytes of a trace event name.
9006		Value: 30
9007	TRC	{_POSIX_TRACE_NAME_MAX}
9008		The length in bytes of a trace generation version string or a trace stream name.
9009		Value: 8
9010	TRC	{_POSIX_TRACE_SYS_MAX}
9011		The number of trace streams that may simultaneously exist in the system.
9012		Value: 8
9013	TRC	{_POSIX_TRACE_USER_EVENT_MAX}
9014		The number of user trace event type identifiers that may simultaneously exist in a traced process, including the predefined user trace event <code>POSIX_TRACE_UNNAMED_USER_EVENT</code> .
9015		Value: 32
9016		
9017		
9018		{_POSIX_TTY_NAME_MAX}
9019		The size of the storage required for a terminal device name, in bytes, including the

9020 terminating null.  
9021 Value: 9

9022 `{_POSIX_TZNAME_MAX}`  
9023 Maximum number of bytes supported for the name of a timezone (not of the TZ variable).  
9024 Value: 6

9025 **Note:** The length given by `{_POSIX_TZNAME_MAX}` does not include the quoting characters  
9026 mentioned in Section 8.3 (on page 161).

9027 `{_POSIX2_BC_BASE_MAX}`  
9028 Maximum *obase* values allowed by the *bc* utility.  
9029 Value: 99

9030 `{_POSIX2_BC_DIM_MAX}`  
9031 Maximum number of elements permitted in an array by the *bc* utility.  
9032 Value: 2 048

9033 `{_POSIX2_BC_SCALE_MAX}`  
9034 Maximum *scale* value allowed by the *bc* utility.  
9035 Value: 99

9036 `{_POSIX2_BC_STRING_MAX}`  
9037 Maximum length of a string constant accepted by the *bc* utility.  
9038 Value: 1 000

9039 `{_POSIX2_CHARCLASS_NAME_MAX}`  
9040 Maximum number of bytes in a character class name.  
9041 Value: 14

9042 `{_POSIX2_COLL_WEIGHTS_MAX}`  
9043 Maximum number of weights that can be assigned to an entry of the *LC\_COLLATE* **order**  
9044 keyword in the locale definition file; see Chapter 7 (on page 119).  
9045 Value: 2

9046 `{_POSIX2_EXPR_NEST_MAX}`  
9047 Maximum number of expressions that can be nested within parentheses by the *expr* utility.  
9048 Value: 32

9049 `{_POSIX2_LINE_MAX}`  
9050 Unless otherwise noted, the maximum length, in bytes, of a utility's input line (either  
9051 standard input or another file), when the utility is described as processing text files. The  
9052 length includes room for the trailing newline.  
9053 Value: 2 048

9054 `{_POSIX2_RE_DUP_MAX}`  
9055 Maximum number of repeated occurrences of a regular expression permitted when using  
9056 the interval notation `\{m,n\}`; see Chapter 9 (on page 165).  
9057 Value: 255

9058 XSI `{_XOPEN_IOV_MAX}`  
9059 Maximum number of **iovec** structures that one process has available for use with *readv()* or  
9060 *writev()*.  
9061 Value: 16

9062 XSI `{_XOPEN_NAME_MAX}`  
9063 Maximum number of bytes in a filename (not including terminating null).  
9064 Value: 255

9065	XSI	{_XOPEN_PATH_MAX}	
9066		Maximum number of bytes in a pathname.	
9067		Value: 1 024	
9068		<b>Numerical Limits</b>	
9069		The values in the following lists shall be defined in <limits.h> and are constant expressions	
9070	XSI	suitable for use in #if preprocessing directives. Moreover, except for {CHAR_BIT}, {DBL_DIG},	
9071		{DBL_MAX}, {FLT_DIG}, {FLT_MAX}, {LONG_BIT}, {WORD_BIT}, and {MB_LEN_MAX}, the	
9072		symbolic names are defined as expressions of the correct type.	
9073		If the value of an object of type <b>char</b> is treated as a signed integer when used in an expression,	
9074		the value of {CHAR_MIN} is the same as that of {SCHAR_MIN} and the value of {CHAR_MAX}	
9075		is the same as that of {SCHAR_MAX}. Otherwise, the value of {CHAR_MIN} is 0 and the value	
9076		of {CHAR_MAX} is the same as that of {UCHAR_MAX}.	
9077		{CHAR_BIT}	
9078		Number of bits in a type <b>char</b> .	
9079	CX	Value: 8	
9080		{CHAR_MAX}	
9081		Maximum value of type <b>char</b> .	
9082		Minimum Acceptable Value: {UCHAR_MAX} or {SCHAR_MAX}	
9083		{INT_MAX}	
9084		Maximum value of an <b>int</b> .	
9085		Minimum Acceptable Value: 2 147 483 647	
9086	XSI	{LONG_BIT}	
9087		Number of bits in a <b>long</b> .	
9088		Minimum Acceptable Value: 32	
9089		{LONG_MAX}	
9090		Maximum value of a <b>long</b> .	
9091		Minimum Acceptable Value: +2 147 483 647	
9092		{MB_LEN_MAX}	
9093		Maximum number of bytes in a character, for any supported locale.	
9094		Minimum Acceptable Value: 1	
9095		{SCHAR_MAX}	
9096		Maximum value of type <b>signed char</b> .	
9097	CX	Value: +127	
9098		{SHRT_MAX}	
9099		Maximum value of type <b>short</b> .	
9100		Minimum Acceptable Value: +32 767	
9101		{SSIZE_MAX}	
9102		Maximum value of an object of type <b>ssize_t</b> .	
9103		Minimum Acceptable Value: {_POSIX_SSIZE_MAX}	
9104		{UCHAR_MAX}	
9105		Maximum value of type <b>unsigned char</b> .	
9106	CX	Value: 255	
9107		{UINT_MAX}	
9108		Maximum value of type <b>unsigned</b> .	
9109		Minimum Acceptable Value: 4 294 967 295	

9110 {ULONG\_MAX}  
 9111 Maximum value of type **unsigned long**.  
 9112 Minimum Acceptable Value: 4 294 967 295

9113 {USHRT\_MAX}  
 9114 Maximum value for a type **unsigned short**.  
 9115 Minimum Acceptable Value: 65 535

9116 XSI {WORD\_BIT}  
 9117 Number of bits in a word or type **int**.  
 9118 Minimum Acceptable Value: 16

9119 {CHAR\_MIN}  
 9120 Minimum value of type **char**.  
 9121 Maximum Acceptable Value: {SCHAR\_MIN} or 0

9122 {INT\_MIN}  
 9123 Minimum value of type **int**.  
 9124 Maximum Acceptable Value: -2 147 483 647

9125 {LONG\_MIN}  
 9126 Minimum value of type **long**.  
 9127 Maximum Acceptable Value: -2 147 483 647

9128 {SCHAR\_MIN}  
 9129 Minimum value of type **signed char**.  
 9130 CX Value: -128

9131 {SHRT\_MIN}  
 9132 Minimum value of type **short**.  
 9133 Maximum Acceptable Value: -32 767

9134 {LLONG\_MIN}  
 9135 Minimum value of type **long long**.  
 9136 Maximum Acceptable Value: -9223372036854775807

9137 {LLONG\_MAX}  
 9138 Maximum value of type **long long**.  
 9139 Minimum Acceptable Value: +9223372036854775807

9140 {ULLONG\_MAX}  
 9141 Maximum value of type **unsigned long long**.  
 9142 Minimum Acceptable Value: 18446744073709551615

**9143 Other Invariant Values**

9144 XSI The following constants shall be defined on all implementations in **<limits.h>**:

9145 XSI {CHARCLASS\_NAME\_MAX}  
 9146 Maximum number of bytes in a character class name.  
 9147 Minimum Acceptable Value: 14

9148 XSI {NL\_ARGMAX}  
 9149 Maximum value of *digit* in calls to the *printf()* and *scanf()* functions.  
 9150 Minimum Acceptable Value: 9

9151 XSI {NL\_LANGMAX}  
 9152 Maximum number of bytes in a *LANG* name.  
 9153 Minimum Acceptable Value: 14

9154	XSI	{NL_MSGMAX}
9155		Maximum message number.
9156		Minimum Acceptable Value: 32 767
9157	XSI	{NL_NMAX}
9158		Maximum number of bytes in an N-to-1 collation mapping.
9159		Minimum Acceptable Value: ' * '
9160	XSI	{NL_SETMAX}
9161		Maximum set number.
9162		Minimum Acceptable Value: 255
9163	XSI	{NL_TEXTMAX}
9164		Maximum number of bytes in a message string.
9165		Minimum Acceptable Value: {_POSIX2_LINE_MAX}
9166	XSI	{NZERO}
9167		Default process priority.
9168		Minimum Acceptable Value: 20

9169 **APPLICATION USAGE**

9170 None.

9171 **RATIONALE**

9172 A request was made to reduce the value of {\_POSIX\_LINK\_MAX} from the value of 8 specified  
 9173 for it in the POSIX.1-1990 standard to 2. The standard developers decided to deny this request  
 9174 for several reasons.

9175 • They wanted to avoid making any changes to the standard that could break conforming  
 9176 applications, and the requested change could have that effect.

9177 • The use of multiple hard links to a file cannot always be replaced with use of symbolic links.  
 9178 Symbolic links are semantically different from hard links in that they associate a pathname  
 9179 with another pathname rather than a pathname with a file. This has implications for access  
 9180 control, file permanence, and transparency.

9181 • The original standard developers had considered the issue of allowing for implementations  
 9182 that did not in general support hard links, and decided that this would reduce consensus on  
 9183 the standard.

9184 Systems that support historical versions of the development option of the ISO POSIX-2 standard  
 9185 retain the name {\_POSIX2\_RE\_DUP\_MAX} as an alias for {\_POSIX\_RE\_DUP\_MAX}.

9186 {PATH\_MAX}

9187 IEEE PASC Interpretation 1003.1 #15 addressed the inconsistency in the standard with the  
 9188 definition of pathname and the description of {PATH\_MAX}, allowing application writers to  
 9189 allocate either {PATH\_MAX} or {PATH\_MAX}+1 bytes. The inconsistency has been  
 9190 removed by correction to the {PATH\_MAX} definition to include the null character. With  
 9191 this change, applications that previously allocated {PATH\_MAX} bytes will continue to  
 9192 succeed.

9193 {SYMLINK\_MAX}

9194 This symbol refers to space for data that is stored in the file system, as opposed to  
 9195 {PATH\_MAX} which is the length of a name that can be passed to a function. In some  
 9196 existing implementations, the filenames pointed to by symbolic links are stored in the  
 9197 inodes of the links, so it is important that {SYMLINK\_MAX} not be constrained to be as  
 9198 large as {PATH\_MAX}.

**9199 FUTURE DIRECTIONS**

9200 None.

**9201 SEE ALSO**

9202 The System Interfaces volume of IEEE Std 1003.1-200x, *fpathconf()*, *pathconf()*, *sysconf()*

**9203 CHANGE HISTORY**

9204 First released in Issue 1.

**9205 Issue 5**

9206 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX  
9207 Threads Extension.

9208 {FILESIZEBITS} added for the Large File Summit extensions.

9209 The minimum acceptable values for {INT\_MAX}, {INT\_MIN}, and {UINT\_MAX} are changed to  
9210 make 32-bit values the minimum requirement.

9211 The entry is restructured to improve readability.

**9212 Issue 6**

9213 The Open Group Corrigendum U033/4 is applied. The wording is made clear for {CHAR\_MIN},  
9214 {INT\_MIN}, {LONG\_MIN}, {SCHAR\_MIN}, and {SHRT\_MIN} that these are maximum  
9215 acceptable values.

9216 The following new requirements on POSIX implementations derive from alignment with the  
9217 Single UNIX Specification:

- 9218 • The minimum value for {CHILD\_MAX} is 25. This is a FIPS requirement.
- 9219 • The minimum value for {OPEN\_MAX} is 20. This is a FIPS requirement.
- 9220 • The minimum value for {NGROUPS\_MAX} is 8. This is also a FIPS requirement.

9221 Symbolic constants are added for {\_POSIX\_SYMLINK\_MAX}, {\_POSIX\_SYMLINK\_MAX},  
9222 {\_POSIX\_RE\_DUP\_MAX}, {RE\_DUP\_MAX}, {SYMLOOP\_MAX}, and {SYMLOOP\_MAX}.

9223 The following values are added for alignment with IEEE Std 1003.1d-1999:

9224 {\_POSIX\_SS\_REPL\_MAX}  
9225 {SS\_REPL\_MAX}  
9226 {POSIX\_ALLOC\_SIZE\_MIN}  
9227 {POSIX\_REC\_INCR\_XFER\_SIZE}  
9228 {POSIX\_REC\_MAX\_XFER\_SIZE}  
9229 {POSIX\_REC\_MIN\_XFER\_SIZE}  
9230 {POSIX\_REC\_XFER\_ALIGN}

9231 Reference to CLOCK\_MONOTONIC is added in the description of {\_POSIX\_CLOCKRES\_MIN}  
9232 for alignment with IEEE Std 1003.1j-2000.

9233 The constants {LLONG\_MIN}, {LLONG\_MAX}, and {ULLONG\_MAX} are added for alignment  
9234 with the ISO/IEC 9899:1999 standard.

9235 The following values are added for alignment with IEEE Std 1003.1q-2000:

9236	{_POSIX_TRACE_EVENT_NAME_MAX}	
9237	{_POSIX_TRACE_NAME_MAX}	
9238	{_POSIX_TRACE_SYS_MAX}	
9239	{_POSIX_TRACE_USER_EVENT_MAX}	
9240	{TRACE_EVENT_NAME_MAX}	
9241	{TRACE_NAME_MAX}	
9242	{TRACE_SYS_MAX}	
9243	{TRACE_USER_EVENT_MAX}	
9244	The new limits {_XOPEN_NAME_MAX} and {_XOPEN_PATH_MAX} are added as minimum	
9245	values for {PATH_MAX} and {NAME_MAX} limits on XSI-conformant systems.	
9246	The legacy symbols {PASS_MAX} and {TMP_MAX} are removed.	
9247	The values for the limits {CHAR_BIT}, {CHAR_MAX}, {SCHAR_MAX}, and {UCHAR_MAX} are	
9248	now required to be 8, +127, 255, and -128, respectively.	

9249 **NAME**

9250 locale.h — category macros

9251 **SYNOPSIS**

9252 #include &lt;locale.h&gt;

9253 **DESCRIPTION**

9254 **CX** Some of the functionality described on this reference page extends the ISO C standard. Any  
 9255 conflict between the requirements described here and the ISO C standard is unintentional. This  
 9256 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

9257 The **<locale.h>** header shall provide a definition for structure **lconv**, which shall include at least  
 9258 the following members. (See the definitions of *LC\_MONETARY* in the Section 7.3.3 (on page  
 9259 138), and Section 7.3.4 (on page 141).)

```

9260 char    *currency_symbol
9261 char    *decimal_point
9262 char    frac_digits
9263 char    *grouping
9264 char    *int_curr_symbol
9265 char    int_frac_digits
9266 char    int_n_cs_precedes
9267 char    int_n_sep_by_space
9268 char    int_n_sign_posn
9269 char    int_p_cs_precedes
9270 char    int_p_sep_by_space
9271 char    int_p_sign_posn
9272 char    *mon_decimal_point
9273 char    *mon_grouping
9274 char    *mon_thousands_sep
9275 char    *negative_sign
9276 char    n_cs_precedes
9277 char    n_sep_by_space
9278 char    n_sign_posn
9279 char    *positive_sign
9280 char    p_cs_precedes
9281 char    p_sep_by_space
9282 char    p_sign_posn
9283 char    *thousands_sep

```

9284 The **<locale.h>** header shall define NULL (as defined in **<stddef.h>**) and at least the following as  
 9285 macros:

```

9286 LC_ALL
9287 LC_COLLATE
9288 LC_CTYPE
9289 CX LC_MESSAGES
9290 LC_MONETARY
9291 LC_NUMERIC
9292 LC_TIME

```

9293 which shall expand to distinct integer constant expressions, for use as the first argument to the  
 9294 *setlocale()* function.

9295 Additional macro definitions, beginning with the characters *LC\_* and an uppercase letter, may  
 9296 also be given here.

9297           The following shall be declared as functions and may also be defined as macros. Function |  
9298           prototypes shall be provided. |

9299           struct lconv \*localeconv (void); |  
9300           char    \*setlocale(int, const char \*); |

9301 **APPLICATION USAGE** |  
9302           None. |

9303 **RATIONALE** |  
9304           None. |

9305 **FUTURE DIRECTIONS** |  
9306           None. |

9307 **SEE ALSO** |  
9308           The System Interfaces volume of IEEE Std 1003.1-200x, *localeconv()*, *setlocale()*, Chapter 8 (on |  
9309           page 157) |

9310 **CHANGE HISTORY** |  
9311           First released in Issue 3. |  
9312           Entry included for alignment with the ISO C standard. |

9313 **Issue 6** |  
9314           The **lconv** structure is expanded with new members (**int\_n\_cs\_precedes**, **int\_n\_sep\_by\_space**, |  
9315           **int\_n\_sign\_posn**, **int\_p\_cs\_precedes**, **int\_p\_sep\_by\_space**, and **int\_p\_sign\_posn**) for alignment |  
9316           with the ISO/IEC 9899:1999 standard. |

9317           Extensions beyond the ISO C standard are now marked. |

9318 **NAME**9319 `math.h` — mathematical declarations9320 **SYNOPSIS**9321 `#include <math.h>`9322 **DESCRIPTION**

9323 `CX` Some of the functionality described on this reference page extends the ISO C standard.  
 9324 Applications shall define the appropriate feature test macro (see the System Interfaces volume of  
 9325 IEEE Std 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of these  
 9326 symbols in this header.

9327 The **<math.h>** header shall include definitions for at least the following types: |9328 **float\_t** A real-floating type at least as wide as **float**. |9329 **double\_t** A real-floating type at least as wide as **double**, and at least as wide as **float\_t**. |

9330 If `FLT_EVAL_METHOD` equals 0, **float\_t** and **double\_t** shall be **float** and **double**, respectively; if  
 9331 `FLT_EVAL_METHOD` equals 1, they shall both be **double**; if `FLT_EVAL_METHOD` equals 2,  
 9332 they shall both be **long double**; for other values of `FLT_EVAL_METHOD`, they are otherwise  
 9333 implementation-defined.

9334 The **<math.h>** header shall define the following macros, where **real-floating** indicates that the  
 9335 argument shall be an expression of real-floating type:

```
9336 int fpclassify(real-floating x);
9337 int isfinite(real-floating x);
9338 int isinf(real-floating x);
9339 int isnan(real-floating x);
9340 int isnormal(real-floating x);
9341 int signbit(real-floating x);
9342 int isgreater(real-floating x, real-floating y);
9343 int isgreaterequal(real-floating x, real-floating y);
9344 int isless(real-floating x, real-floating y);
9345 int islessequal(real-floating x, real-floating y);
9346 int islessgreater(real-floating x, real-floating y);
9347 int isunordered(real-floating x, real-floating y);
```

9348 The **<math.h>** header shall provide for the following constants. The values are of type **double**  
 9349 and are accurate within the precision of the **double** type.

9350	<code>XSI</code>	<code>M_E</code>	Value of $e$
9351		<code>M_LOG2E</code>	Value of $\log_2 e$
9352		<code>M_LOG10E</code>	Value of $\log_{10} e$
9353		<code>M_LN2</code>	Value of $\log_e 2$
9354		<code>M_LN10</code>	Value of $\log_e 10$
9355		<code>M_PI</code>	Value of $\pi$
9356		<code>M_PI_2</code>	Value of $\pi/2$
9357		<code>M_PI_4</code>	Value of $\pi/4$
9358		<code>M_1_PI</code>	Value of $1/\pi$
9359		<code>M_2_PI</code>	Value of $2/\pi$

9360	<code>M_2_SQRTPI</code>	Value of $2\sqrt{\pi}$	
9361	<code>M_SQRT2</code>	Value of $\sqrt{2}$	
9362	<code>M_SQRT1_2</code>	Value of $1\sqrt{2}$	
9363	The header shall define the following symbolic constants:		
9364	<code>MAXFLOAT</code>	Value of maximum non-infinite single-precision floating-point number.	
9365	<code>HUGE_VAL</code>	A positive <b>double</b> expression, not necessarily representable as a <b>float</b> . Used as an error value returned by the mathematics library. <code>HUGE_VAL</code> evaluates to +infinity on systems supporting IEEE Std 754-1985.	
9366			
9367			
9368	<code>HUGE_VALF</code>	A positive <b>float</b> constant expression. Used as an error value returned by the mathematics library. <code>HUGE_VALF</code> evaluates to +infinity on systems supporting IEEE Std 754-1985.	
9369			
9370			
9371	<code>HUGE_VALL</code>	A positive <b>long double</b> constant expression. Used as an error value returned by the mathematics library. <code>HUGE_VALL</code> evaluates to +infinity on systems supporting IEEE Std 754-1985.	
9372			
9373			
9374	<code>INFINITY</code>	A constant expression of type <b>float</b> representing positive or unsigned infinity, if available; else a positive constant of type <b>float</b> that overflows at translation time.	
9375			
9376			
9377	<code>NAN</code>	A constant expression of type <b>float</b> representing a quiet NaN. This symbolic constant is only defined if the implementation supports quiet NaNs for the <b>float</b> type.	
9378			
9379			
9380	The following macros shall be defined for number classification. They represent the mutually-exclusive kinds of floating-point values. They expand to integer constant expressions with distinct values. Additional implementation-defined floating-point classifications, with macro definitions beginning with <code>FP_</code> and an uppercase letter, may also be specified by the implementation.		
9381			
9382			
9383			
9384			
9385	<code>FP_INFINITE</code>		
9386	<code>FP_NAN</code>		
9387	<code>FP_NORMAL</code>		
9388	<code>FP_SUBNORMAL</code>		
9389	<code>FP_ZERO</code>		
9390	The following optional macros indicate whether the <code>fma()</code> family of functions are fast compared with direct code:		
9391			
9392	<code>FP_FAST_FMA</code>		
9393	<code>FP_FAST_FMAF</code>		
9394	<code>FP_FAST_FMAL</code>		
9395	The <code>FP_FAST_FMA</code> macro shall be defined to indicate that the <code>fma()</code> function generally executes about as fast as, or faster than, a multiply and an add of <b>double</b> operands. The other macros have the equivalent meaning for the <b>float</b> and <b>long double</b> versions.		
9396			
9397			
9398	The following macros shall expand to integer constant expressions whose values are returned by <code>ilogb(x)</code> if <code>x</code> is zero or NaN, respectively. The value of <code>FP_ILOGB0</code> shall be either <code>{INT_MIN}</code> or <code>-{INT_MAX}</code> . The value of <code>FP_ILOGBNAN</code> shall be either <code>{INT_MAX}</code> or <code>{INT_MIN}</code> .		
9399			
9400			
9401	<code>FP_ILOGB0</code>		
9402	<code>FP_ILOGBNAN</code>		

9403           The following macros shall expand to the integer constants 1 and 2, respectively;

```
9404           MATH_ERRNO  
9405           MATH_ERREXCEPT
```

9406           The following macro shall expand to an expression that has type **int** and the value  
9407           MATH\_ERRNO, MATH\_ERREXCEPT, or the bitwise-inclusive OR of both:

```
9408           math_errhandling
```

9409           The value of `math_errhandling` is constant for the duration of the program. It is unspecified  
9410           whether `math_errhandling` is a macro or an identifier with external linkage. If a macro definition  
9411           is suppressed or a program defines an identifier with the name `math_errhandling`, the behavior  
9412           is undefined. If the expression `(math_errhandling & MATH_ERREXCEPT)` can be non-zero, the  
9413           implementation shall define the macros `FE_DIVBYZERO`, `FE_INVALID`, and `FE_OVERFLOW` in  
9414           **<fenv.h>**.

9415           The following shall be declared as functions and may also be defined as macros. Function  
9416           prototypes shall be provided.

```
9417           double       acos(double);  
9418           float        acosf(float);  
9419           double       acosh(double);  
9420           float        acoshf(float);  
9421           long double  acoshl(long double);  
9422           long double  acosl(long double);  
9423           double       asin(double);  
9424           float        asinf(float);  
9425           double       asinh(double);  
9426           float        asinhf(float);  
9427           long double  asinh1(long double);  
9428           long double  asinl(long double);  
9429           double       atan(double);  
9430           double       atan2(double, double);  
9431           float        atan2f(float, float);  
9432           long double  atan2l(long double, long double);  
9433           float        atanf(float);  
9434           double       atanh(double);  
9435           float        atanhf(float);  
9436           long double  atanh1(long double);  
9437           long double  atanl(long double);  
9438           double       cbrt(double);  
9439           float        cbrtf(float);  
9440           long double  cbrtl(long double);  
9441           double       ceil(double);  
9442           float        ceilf(float);  
9443           long double  ceill(long double);  
9444           double       copysign(double, double);  
9445           float        copysignf(float, float);  
9446           long double  copysignl(long double, long double);  
9447           double       cos(double);  
9448           float        cosf(float);  
9449           double       cosh(double);  
9450           float        coshf(float);  
9451           long double  cosh1(long double);
```

```

9452     long double cosl(long double);
9453     double      erf(double);
9454     double      erfc(double);
9455     float       erfcf(float);
9456     long double erfcl(long double);
9457     float       erff(float);
9458     long double erfl(long double);
9459     double      exp(double);
9460     double      exp2(double);
9461     float       exp2f(float);
9462     long double exp2l(long double);
9463     float       expf(float);
9464     long double expl(long double);
9465     double      expm1(double);
9466     float       expm1f(float);
9467     long double expm1l(long double);
9468     double      fabs(double);
9469     float       fabsf(float);
9470     long double fabsl(long double);
9471     double      fdim(double, double);
9472     float       fdimf(float, float);
9473     long double fdiml(long double, long double);
9474     double      floor(double);
9475     float       floorf(float);
9476     long double floorl(long double);
9477     double      fma(double, double, double);
9478     float       fmaf(float, float, float);
9479     long double fmal(long double, long double, long double);
9480     double      fmax(double, double);
9481     float       fmaxf(float, float);
9482     long double fmaxl(long double, long double);
9483     double      fmin(double, double);
9484     float       fminf(float, float);
9485     long double fminl(long double, long double);
9486     double      fmod(double, double);
9487     float       fmodf(float, float);
9488     long double fmodl(long double, long double);
9489     double      frexp(double, int *);
9490     float       frexpf(float value, int *);
9491     long double frexpl(long double value, int *);
9492     double      hypot(double, double);
9493     float       hypotf(float, float);
9494     long double hypotl(long double, long double);
9495     int         ilogb(double);
9496     int         ilogbf(float);
9497     int         ilogbl(long double);
9498 XSI     double      j0(double);
9499         double      j1(double);
9500         double      jn(int, double);
9501     double      ldexp(double, int);
9502     float       ldexpf(float, int);
9503     long double ldexpl(long double, int);

```

```
9504     double      lgamma(double);
9505     float        lgammaf(float);
9506     long double   lgammal(long double);
9507     long long     llrint(double);
9508     long long     llrintf(float);
9509     long long     llrintl(long double);
9510     long long     llround(double);
9511     long long     llroundf(float);
9512     long long     llroundl(long double);
9513     double        log(double);
9514     double        log10(double);
9515     float         log10f(float);
9516     long double   log10l(long double);
9517     double        log1p(double);
9518     float         log1pf(float);
9519     long double   log1pl(long double);
9520     double        log2(double);
9521     float         log2f(float);
9522     long double   log2l(long double);
9523     double        logb(double);
9524     float         logbf(float);
9525     long double   logbl(long double);
9526     float         logf(float);
9527     long double   logl(long double);
9528     long          lrint(double);
9529     long          lrintf(float);
9530     long          lrintl(long double);
9531     long          lround(double);
9532     long          lroundf(float);
9533     long          lroundl(long double);
9534     double        modf(double, double *);
9535     float         modff(float, float *);
9536     long double   modfl(long double, long double *);
9537     double        nan(const char *);
9538     float         nanf(const char *);
9539     long double   nanl(const char *);
9540     double        nearbyint(double);
9541     float         nearbyintf(float);
9542     long double   nearbyintl(long double);
9543     double        nextafter(double, double);
9544     float         nextafterf(float, float);
9545     long double   nextafterl(long double, long double);
9546     double        nexttoward(double, long double);
9547     float         nexttowardf(float, long double);
9548     long double   nexttowardl(long double, long double);
9549     double        pow(double, double);
9550     float         powf(float, float);
9551     long double   powl(long double, long double);
9552     double        remainder(double, double);
9553     float         remainderf(float, float);
9554     long double   remainderl(long double, long double);
9555     double        remquo(double, double, int *);
```

```

9556     float      remquof(float, float, int *);
9557     long double remquol(long double, long double, int *);
9558     double     rint(double);
9559     float      rintf(float);
9560     long double rintl(long double);
9561     double     round(double);
9562     float      roundf(float);
9563     long double roundl(long double);
9564 XSI    double     scalb(double, double);
9565     double     scalbln(double, long);
9566     float      scalblnf(float, long);
9567     long double scalblnl(long double, long);
9568     double     scalbn(double, int);
9569     float      scalbnf(float, int);
9570     long double scalbnl(long double, int);
9571     double     sin(double);
9572     float      sinf(float);
9573     double     sinh(double);
9574     float      sinhf(float);
9575     long double sinhl(long double);
9576     long double sinl(long double);
9577     double     sqrt(double);
9578     float      sqrtf(float);
9579     long double sqrtl(long double);
9580     double     tan(double);
9581     float      tanf(float);
9582     double     tanh(double);
9583     float      tanhf(float);
9584     long double tanhl(long double);
9585     long double tanl(long double);
9586     double     tgamma(double);
9587     float      tgammaf(float);
9588     long double tgammaL(long double);
9589     double     trunc(double);
9590     float      truncf(float);
9591     long double truncL(long double);
9592 XSI    double     y0(double);
9593     double     y1(double);
9594     double     yn(int, double);
9595

```

9596 The following external variable shall be defined:

```

9597 XSI    extern int signgam;
9598

```

9599 The behavior of each of the functions defined in <math.h> is specified in the System Interfaces  
9600 volume of IEEE Std 1003.1-200x for all representable values of its input arguments, except where  
9601 stated otherwise. Each function shall execute as if it were a single operation without generating  
9602 any externally visible exceptional conditions.

**9603 APPLICATION USAGE**

9604 The FP\_CONTRACT pragma can be used to allow (if the state is on) or disallow (if the state is  
9605 off) the implementation to contract expressions. Each pragma can occur either outside external  
9606 declarations or preceding all explicit declarations and statements inside a compound statement.  
9607 When outside external declarations, the pragma takes effect from its occurrence until another  
9608 FP\_CONTRACT pragma is encountered, or until the end of the translation unit. When inside a  
9609 compound statement, the pragma takes effect from its occurrence until another FP\_CONTRACT  
9610 pragma is encountered (including within a nested compound statement), or until the end of the  
9611 compound statement; at the end of a compound statement the state for the pragma is restored to  
9612 its condition just before the compound statement. If this pragma is used in any other context, the  
9613 behavior is undefined. The default state (on or off) for the pragma is implementation-defined.

**9614 RATIONALE**

9615 Before the ISO/IEC 9899:1999 standard, the math library was defined only for the floating type  
9616 **double**. All the names formed by appending 'f' or 'l' to a name in **<math.h>** were reserved  
9617 to allow for the definition of **float** and **long double** libraries; and the ISO/IEC 9899:1999  
9618 standard provides for all three versions of math functions.

9619 The functions *ecvt()*, *fcvt()*, and *gcvt()* have been dropped from the ISO C standard since their  
9620 capability is available through *sprintf()*. These are provided on XSI-conformant systems  
9621 supporting the Legacy Option Group.

**9622 FUTURE DIRECTIONS**

9623 None.

**9624 SEE ALSO**

9625 The System Interfaces volume of IEEE Std 1003.1-200x, *acos()*, *acosh()*, *asin()*, *atan()*, *atan2()*,  
9626 *cbrt()*, *ceil()*, *cos()*, *cosh()*, *erf()*, *exp()*, *expm1()*, *fabs()*, *floor()*, *fmod()*, *frexp()*, *hypot()*, *ilogb()*,  
9627 *isnan()*, *j0()*, *ldexp()*, *lgamma()*, *log()*, *log10()*, *log1p()*, *logb()*, *modf()*, *nextafter()*, *pow()*,  
9628 *remainder()*, *rint()*, *scalb()*, *sin()*, *sinh()*, *sqrt()*, *tan()*, *tanh()*, *y0()*

**9629 CHANGE HISTORY**

9630 First released in Issue 1.

**9631 Issue 6**

9632 This reference page is updated to align with the ISO/IEC 9899:1999 standard.

9633 **NAME**

9634       monetary.h — monetary types

9635 **SYNOPSIS**

9636 XSI     #include &lt;monetary.h&gt;

9637

9638 **DESCRIPTION**

9639       The &lt;monetary.h&gt; header shall define the following types:

9640       **size\_t**            As described in <stddef.h>.9641       **ssize\_t**          As described in <sys/types.h>.9642       The following shall be declared as a function and may also be defined as a macro. A function |  
9643       prototype shall be provided. |

9644       ssize\_t   strfmon(char \*restrict, size\_t, const char \*restrict, ...);

9645 **APPLICATION USAGE**

9646       None.

9647 **RATIONALE**

9648       None.

9649 **FUTURE DIRECTIONS**

9650       None.

9651 **SEE ALSO**9652       The System Interfaces volume of IEEE Std 1003.1-200x, *strfmon()*9653 **CHANGE HISTORY**

9654       First released in Issue 4.

9655 **Issue 6**9656       The **restrict** keyword is added to the prototype for *strfmon()*.

9657 **NAME**9658 mqueue.h — message queues (**REALTIME**)9659 **SYNOPSIS**

9660 MSG #include &lt;mqueue.h&gt;

9661

9662 **DESCRIPTION**9663 The <mqueue.h> header shall define the **mqd\_t** type, which is used for message queue  
9664 descriptors. This is not an array type.9665 The <mqueue.h> header shall define the **sigevent** structure (as described in <signal.h>) and the  
9666 **mq\_attr** structure, which is used in getting and setting the attributes of a message queue.  
9667 Attributes are initially set when the message queue is created. An **mq\_attr** structure shall have at  
9668 least the following fields:

9669	long	mq_flags	Message queue flags.
9670	long	mq_maxmsg	Maximum number of messages.
9671	long	mq_msgsize	Maximum message size.
9672	long	mq_curmsgs	Number of messages currently queued.

9673 The following shall be declared as functions and may also be defined as macros. Function  
9674 prototypes shall be provided.

```

9675 int      mq_close(mqd_t);
9676 int      mq_getattr(mqd_t, struct mq_attr *);
9677 int      mq_notify(mqd_t, const struct sigevent *);
9678 mqd_t    mq_open(const char *, int, ...);
9679 ssize_t  mq_receive(mqd_t, char *, size_t, unsigned *);
9680 int      mq_send(mqd_t, const char *, size_t, unsigned);
9681 int      mq_setattr(mqd_t, const struct mq_attr *restrict,
9682                  struct mq_attr *restrict);
9683 TMO     ssize_t mq_timedreceive(mqd_t, char *restrict, size_t,
9684                               unsigned *restrict, const struct timespec *restrict);
9685 int      mq_timedsend(mqd_t, const char *, size_t, unsigned,
9686                     const struct timespec *);
9687 int      mq_unlink(const char *);

```

9688 Inclusion of the <mqueue.h> header may make visible symbols defined in the headers <fcntl.h>,  
9689 <signal.h>, <sys/types.h>, and <time.h>.9690 **APPLICATION USAGE**

9691 None.

9692 **RATIONALE**

9693 None.

9694 **FUTURE DIRECTIONS**

9695 None.

9696 **SEE ALSO**9697 <fcntl.h>, <signal.h>, <sys/types.h>, <time.h>, the System Interfaces volume of  
9698 IEEE Std 1003.1-200x, *mq\_close()*, *mq\_getattr()*, *mq\_notify()*, *mq\_open()*, *mq\_receive()*, *mq\_send()*,  
9699 *mq\_setattr()*, *mq\_timedreceive()*, *mq\_timedsend()*, *mq\_unlink()*

9700 **CHANGE HISTORY**

9701 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

9702 **Issue 6**

9703 The <mqqueue.h> header is marked as part of the Message Passing option.

9704 The *mq\_timedreceive()* and *mq\_timedsend()* functions are added for alignment with  
9705 IEEE Std 1003.1d-1999.

9706 The **restrict** keyword is added to the prototypes for *mq\_setattr()* and *mq\_timedreceive()*.

9707 **NAME**

9708 ndbm.h — definitions for ndbm database operations

9709 **SYNOPSIS**

9710 xSI #include &lt;ndbm.h&gt;

9711

9712 **DESCRIPTION**9713 The <ndbm.h> header shall define the **datum** type as a structure that includes at least the  
9714 following members:

9715 void \*dptr A pointer to the application's data.

9716 size\_t dsize The size of the object pointed to by *dptr*.9717 The **size\_t** type shall be defined as described in <stddef.h>.9718 The <ndbm.h> header shall define the **DBM** type.9719 The following constants shall be defined as possible values for the *store\_mode* argument to  
9720 *dbm\_store()*:

9721 DBM\_INSERT Insertion of new entries only.

9722 DBM\_REPLACE Allow replacing existing entries.

9723 The following shall be declared as functions and may also be defined as macros. Function |  
9724 prototypes shall be provided. |

9725 int dbm\_clearerr(DBM \*);

9726 void dbm\_close(DBM \*);

9727 int dbm\_delete(DBM \*, datum);

9728 int dbm\_error(DBM \*);

9729 datum dbm\_fetch(DBM \*, datum);

9730 datum dbm\_firstkey(DBM \*);

9731 datum dbm\_nextkey(DBM \*);

9732 DBM \*dbm\_open(const char \*, int, mode\_t);

9733 int dbm\_store(DBM \*, datum, datum, int);

9734 The **mode\_t** type shall be defined through **typedef** as described in <sys/types.h>.9735 **APPLICATION USAGE**

9736 None.

9737 **RATIONALE**

9738 None.

9739 **FUTURE DIRECTIONS**

9740 None.

9741 **SEE ALSO**9742 The System Interfaces volume of IEEE Std 1003.1-200x, *dbm\_clearerr()*9743 **CHANGE HISTORY**

9744 First released in Issue 4, Version 2.

9745 **Issue 5**9746 References to the definitions of **size\_t** and **mode\_t** are added to the DESCRIPTION.

9747 **NAME**

9748 net/if.h — sockets local interfaces

9749 **SYNOPSIS**

9750 #include <net/if.h>

9751 **DESCRIPTION**

9752 The <net/if.h> header shall define the **if\_nameindex** structure that includes at least the  
 9753 following members:

9754 unsigned if\_index Numeric index of the interface.  
 9755 char \*if\_name Null-terminated name of the interface.

9756 The <net/if.h> header shall define the following macro for the length of a buffer containing an  
 9757 interface name (including the terminating NULL character):

9758 **IF\_NAMESIZE** Interface name length.

9759 The following shall be declared as functions and may also be defined as macros. Function |  
 9760 prototypes shall be provided. |

9761 unsigned if\_nametoindex(const char \*); |  
 9762 char \*if\_indextoname(unsigned, char \*); |  
 9763 struct if\_nameindex \*if\_nameindex(void); |  
 9764 void if\_freenameindex(struct if\_nameindex \*); |

9765 **APPLICATION USAGE** |

9766 None. |

9767 **RATIONALE**

9768 None.

9769 **FUTURE DIRECTIONS**

9770 None.

9771 **SEE ALSO**

9772 The System Interfaces volume of IEEE Std 1003.1-200x, *if\_freenameindex()*, *if\_indextoname()*,  
 9773 *if\_nameindex()*, *if\_nametoindex()*

9774 **CHANGE HISTORY**

9775 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

9776 **NAME**

9777 netdb.h — definitions for network database operations

9778 **SYNOPSIS**

9779 #include &lt;netdb.h&gt;

9780 **DESCRIPTION**9781 The <netdb.h> header may define the **in\_port\_t** type and the **in\_addr\_t** type as described in  
9782 <netinet/in.h>.9783 The <netdb.h> header shall define the **hostent** structure that includes at least the following  
9784 members:

9785	char	*h_name	Official name of the host.
9786	char	**h_aliases	A pointer to an array of pointers to 9787 alternative host names, terminated by a 9788 null pointer.
9789	int	h_addrtype	Address type.
9790	int	h_length	The length, in bytes, of the address.
9791	char	**h_addr_list	A pointer to an array of pointers to network 9792 addresses (in network byte order) for the host, 9793 terminated by a null pointer.

9794 The <netdb.h> header shall define the **netent** structure that includes at least the following  
9795 members:

9796	char	*n_name	Official, fully-qualified (including the 9797 domain) name of the host.
9798	char	**n_aliases	A pointer to an array of pointers to 9799 alternative network names, terminated by a 9800 null pointer.
9801	int	n_addrtype	The address type of the network.
9802	uint32_t	n_net	The network number, in host byte order.

9803 The **uint32\_t** type shall be defined as described in <inttypes.h>.9804 The <netdb.h> header shall define the **protoent** structure that includes at least the following  
9805 members:

9806	char	*p_name	Official name of the protocol.
9807	char	**p_aliases	A pointer to an array of pointers to 9808 alternative protocol names, terminated by 9809 a null pointer.
9810	int	p_proto	The protocol number.

9811 The <netdb.h> header shall define the **servent** structure that includes at least the following  
9812 members:

9813	char	*s_name	Official name of the service.
9814	char	**s_aliases	A pointer to an array of pointers to 9815 alternative service names, terminated by 9816 a null pointer.
9817	int	s_port	The port number at which the service 9818 resides, in network byte order.
9819	char	*s_proto	The name of the protocol to use when 9820 contacting the service.

9821 The <netdb.h> header shall define the IPPORT\_RESERVED macro with the value of the highest  
 9822 reserved Internet port number.

9823 OB When the <netdb.h> header is included, *h\_errno* shall be available as a modifiable l-value of type  
 9824 **int**. It is unspecified whether *h\_errno* is a macro or an identifier declared with external linkage.

9825 The <netdb.h> header shall define the following macros for use as error values for  
 9826 *gethostbyaddr()* and *gethostbyname()*:

9827 HOST\_NOT\_FOUND  
 9828 NO\_DATA  
 9829 NO\_RECOVERY  
 9830 TRY\_AGAIN

9831 **Address Information Structure**

9832 The <netdb.h> header shall define the **addrinfo** structure that includes at least the following  
 9833 members:

9834	int	ai_flags	Input flags.
9835	int	ai_family	Address family of socket.
9836	int	ai_socktype	Socket type.
9837	int	ai_protocol	Protocol of socket.
9838	socklen_t	ai_addrlen	Length of socket address.
9839	struct sockaddr	*ai_addr	Socket address of socket.
9840	char	*ai_canonname	Canonical name of service location.
9841	struct addrinfo	*ai_next	Pointer to next in list.

9842 The <netdb.h> header shall define the following macros that evaluate to bitwise-distinct integer  
 9843 constants for use in the *flags* field of the **addrinfo** structure:

9844	AI_PASSIVE	Socket address is intended for <i>bind()</i> .
9845	AI_CANONNAME	Request for canonical name.
9847	AI_NUMERICHOST	Return numeric host address as name.
9849	AI_NUMERICSERV	Inhibit service name resolution.
9851	AI_V4MAPPED	If no IPv6 addresses are found, query for IPv4 addresses and return them to the caller as IPv4-mapped IPv6 addresses.
9854	AI_ALL	Query for both IPv4 and IPv6 addresses.
9855	AI_ADDRCONFIG	Query for IPv4 addresses only when an IPv4 address is configured; query for IPv6 addresses only when an IPv6 address is configured.

9858 The <netdb.h> header shall define the following macros that evaluate to bitwise-distinct integer  
 9859 constants for use in the *flags* argument to *getnameinfo()*:

9860	NI_NOFQDN	Only the nodename portion of the FQDN is returned for local hosts.
9861	NI_NUMERICHOST	The numeric form of the node's address is returned instead of its name.

9863 NI\_NAMEREQD Return an error if the node's name cannot be located in the database.  
 9864 NI\_NUMERICSERV  
 9865 The numeric form of the service address is returned instead of its name.  
 9866 NI\_DGRAM Indicates that the service is a datagram service (SOCK\_DGRAM).

9867 **Address Information Errors**

9868 The <netdb.h> header shall define the following macros for use as error values for *getaddrinfo()*  
 9869 and *getnameinfo()*:

9870 EAI\_AGAIN The name could not be resolved at this time. Future attempts may succeed.

9871 EAI\_BADFLAGS The flags had an invalid value.

9872 EAI\_FAIL A non-recoverable error occurred.

9873 EAI\_FAMILY The address family was not recognized or the address length was invalid for  
 9874 the specified family.

9875 EAI\_MEMORY There was a memory allocation failure.

9876 EAI\_NONAME The name does not resolve for the supplied parameters.

9877 NI\_NAMEREQD is set and the host's name cannot be located, or both  
 9878 *nodename* and *servname* were null.

9879 EAI\_SERVICE The service passed was not recognized for the specified socket type.

9880 EAI\_SOCKTYPE The intended socket type was not recognized.

9881 EAI\_SYSTEM A system error occurred. The error code can be found in *errno*. |

9882 EAI\_OVERFLOW An argument buffer overflowed. |

9883 The following shall be declared as functions and may also be defined as macros. Function |  
 9884 prototypes shall be provided. |

```

9885 void          endhostent(void);
9886 void          endnetent(void);
9887 void          endprotoent(void);
9888 void          endservent(void);
9889 void          freeaddrinfo(struct addrinfo *);
9890 const char    *gai_strerror(int);
9891 int           getaddrinfo(const char *restrict, const char *restrict, |
9892               const struct addrinfo *restrict, |
9893               struct addrinfo **restrict);
9894 struct hostent *gethostbyaddr(const void *, socklen_t, int);
9895 struct hostent *gethostbyname(const char *);
9896 struct hostent *gethostent(void);
9897 int           getnameinfo(const struct sockaddr *restrict, socklen_t, |
9898               char *restrict, socklen_t, char *restrict, |
9899               socklen_t, unsigned);
9900 struct netent *getnetbyaddr(uint32_t, int);
9901 struct netent *getnetbyname(const char *);
9902 struct netent *getnetent(void);
9903 struct protoent *getprotobyname(const char *);
9904 struct protoent *getprotobynumber(int);
9905 struct protoent *getprotoent(void);

```

```

9906     struct servent    *getservbyname(const char *, const char *);
9907     struct servent    *getservbyport(int, const char *);
9908     struct servent    *getservent(void);
9909     void              sethostent(int);
9910     void              setnetent(int);
9911     void              setprotoent(int);
9912     void              setservent(int);

```

9913 The type **socklen\_t** shall be defined through **typedef** as described in <sys/socket.h>.

9914 Inclusion of the <netdb.h> header may also make visible all symbols from <netinet/in.h>, |  
9915 <sys/socket.h>, and <inttypes.h>. |

#### 9916 APPLICATION USAGE

9917 None.

#### 9918 RATIONALE

9919 None.

#### 9920 FUTURE DIRECTIONS

9921 None.

#### 9922 SEE ALSO

9923 <netinet/in.h>, <inttypes.h>, <sys/socket.h>, the System Interfaces volume of  
9924 IEEE Std 1003.1-200x, *bind()*, *endhostent()*, *endnetent()*, *endprotoent()*, *endservent()*, *getaddrinfo()*,  
9925 *getnameinfo()*

#### 9926 CHANGE HISTORY

9927 First released in Issue 6. Derived from the XNS, Issue 5.2 specification. |

9928 The Open Group Base Resolution bwg2001-009 is applied, which changes the return type for |  
9929 *gai\_strerror()* from **char \*** to **const char \***. This is for coordination with the IPnG Working Group. |

9930 NAME

9931       netinet/in.h — Internet address family

9932 SYNOPSIS

9933       #include <netinet/in.h>

9934 DESCRIPTION

9935       The <netinet/in.h> header shall define the following types:

9936       **in\_port\_t**   Equivalent to the type **uint16\_t** as defined in <inttypes.h>.

9937       **in\_addr\_t**   Equivalent to the type **uint32\_t** as defined in <inttypes.h>.

9938       The **sa\_family\_t** type shall be defined as described in <sys/socket.h>.

9939       The **uint8\_t** and **uint32\_t** type shall be defined as described in <inttypes.h>. Inclusion of the  
9940 <netinet/in.h> header may also make visible all symbols from <inttypes.h> and <sys/socket.h>.

9941       The <netinet/in.h> header shall define the **in\_addr** structure that includes at least the following  
9942 member:

9943       in\_addr\_t   s\_addr

9944       The <netinet/in.h> header shall define the **sockaddr\_in** structure that includes at least the  
9945 following members (all in network byte order):

9946       sa\_family\_t       sin\_family    AF\_INET.  
9947       in\_port\_t         sin\_port     Port number.  
9948       struct in\_addr    sin\_addr     IP address.

9949       The **sockaddr\_in** structure is used to store addresses for the Internet address family. Values of  
9950 this type shall be cast by applications to **struct sockaddr** for use with socket functions.

9951 IP6       The <netinet/in.h> header shall define the **in6\_addr** structure that contains at least the following  
9952 member:

9953       uint8\_t   s6\_addr[16]

9954       This array is used to contain a 128-bit IPv6 address, stored in network byte order.

9955       The <netinet/in.h> header shall define the **sockaddr\_in6** structure that includes at least the  
9956 following members (all in network byte order):

9957       sa\_family\_t       sin6\_family   AF\_INET6.  
9958       in\_port\_t         sin6\_port    Port number.  
9959       uint32\_t         sin6\_flowinfo IPv6 traffic class and flow information.  
9960       struct in6\_addr   sin6\_addr     IPv6 address.  
9961       uint32\_t         sin6\_scope\_id  Set of interfaces for a scope.

9962       The **sockaddr\_in6** structure shall be set to zero by an application prior to using it, since  
9963 implementations are free to have additional, implementation-defined fields in **sockaddr\_in6**.

9964       The *sin6\_scope\_id* field is a 32-bit integer that identifies a set of interfaces as appropriate for the  
9965 scope of the address carried in the *sin6\_addr* field. For a link scope *sin6\_addr*, *sin6\_scope\_id* would  
9966 be an interface index. For a site scope *sin6\_addr*, *sin6\_scope\_id* would be a site identifier. The  
9967 mapping of *sin6\_scope\_id* to an interface or set of interfaces is implementation-defined.

9968       The <netinet/in.h> header shall declare the following external variable:

9969       struct in6\_addr in6addr\_any

9970       This variable is initialized by the system to contain the wildcard IPv6 address. The  
9971 <netinet/in.h> header also defines the IN6ADDR\_ANY\_INIT macro. This macro must be

9972 constant at compile time and can be used to initialize a variable of type **struct in6\_addr** to the  
 9973 IPv6 wildcard address.

9974 The <netinet/in.h> header shall declare the following external variable:

```
9975 struct in6_addr in6addr_loopback
```

9976 This variable is initialized by the system to contain the loopback IPv6 address. The  
 9977 <netinet/in.h> header also defines the IN6ADDR\_LOOPBACK\_INIT macro. This macro must be  
 9978 constant at compile time and can be used to initialize a variable of type **struct in6\_addr** to the  
 9979 IPv6 loopback address.

9980 The <netinet/in.h> header shall define the **ipv6\_mreq** structure that includes at least the  
 9981 following members:

```
9982 struct in6_addr  ipv6mr_multiaddr  IPv6 multicast address.  

9983 unsigned        ipv6mr_interface  Interface index.
```

9984

9985 The <netinet/in.h> header shall define the following macros for use as values of the *level*  
 9986 argument of *getsockopt()* and *setsockopt()*:

9987	IPPROTO_IP	Internet protocol.
9988	IPPROTO_IPV6	Internet Protocol Version 6.
9989	IPPROTO_ICMP	Control message protocol.
9990	IPPROTO_RAW	Raw IP Packets Protocol.
9991	IPPROTO_TCP	Transmission control protocol.
9992	IPPROTO_UDP	User datagram protocol.

9993 The <netinet/in.h> header shall define the following macros for use as destination addresses for  
 9994 *connect()*, *sendmsg()*, and *sendto()*:

9995	INADDR_ANY	IPv4 local host address.
9996	INADDR_BROADCAST	IPv4 broadcast address.

9997 The <netinet/in.h> header shall define the following macro to help applications declare buffers  
 9998 of the proper size to store IPv4 addresses in string form:

9999	INET_ADDRSTRLEN	16. Length of the string form for IP.
------	-----------------	---------------------------------------

10000 The *htonl()*, *htons()*, *ntohl()*, and *ntohs()* functions shall be available as defined in <arpa/inet.h>.  
 10001 Inclusion of the <netinet/in.h> header may also make visible all symbols from <arpa/inet.h>.

10002 IP6 The <netinet/in.h> header shall define the following macro to help applications declare buffers  
 10003 of the proper size to store IPv6 addresses in string form:

10004	INET6_ADDRSTRLEN	46. Length of the string form for IPv6.
-------	------------------	---

10005 The <netinet/in.h> header shall define the following macros, with distinct integer values, for use  
 10006 in the *option\_name* argument in the *getsockopt()* or *setsockopt()* functions at protocol level  
 10007 IPPROTO\_IPV6:

10008	IPV6_JOIN_GROUP	Join a multicast group.
10009	IPV6_LEAVE_GROUP	Quit a multicast group.
10010	IPV6_MULTICAST_HOPS	
10011		Multicast hop limit.

10012	IPV6_MULTICAST_IF	Interface to use for outgoing multicast packets.
10013	IPV6_MULTICAST_LOOP	
10014		Multicast packets are delivered back to the local application.
10015	IPV6_UNICAST_HOPS	Unicast hop limit.
10016	IPV6_V6ONLY	Restrict AF_INET6 socket to IPv6 communications only.
10017	The <netinet/in.h> header shall define the following macros that test for special IPv6 addresses.	
10018	Each macro is of type <b>int</b> and takes a single argument of type <b>const struct in6_addr*</b> :	
10019	IN6_IS_ADDR_UNSPECIFIED	
10020		Unspecified address.
10021	IN6_IS_ADDR_LOOPBACK	
10022		Loopback address.
10023	IN6_IS_ADDR_MULTICAST	
10024		Multicast address.
10025	IN6_IS_ADDR_LINKLOCAL	
10026		Unicast link-local address.
10027	IN6_IS_ADDR_SITELOCAL	
10028		Unicast site-local address.
10029	IN6_IS_ADDR_V4MAPPED	
10030		IPv4 mapped address.
10031	IN6_IS_ADDR_V4COMPAT	
10032		IPv4-compatible address.
10033	IN6_IS_ADDR_MC_NODELOCAL	
10034		Multicast node-local address.
10035	IN6_IS_ADDR_MC_LINKLOCAL	
10036		Multicast link-local address.
10037	IN6_IS_ADDR_MC_SITELOCAL	
10038		Multicast site-local address.
10039	IN6_IS_ADDR_MC_ORGLOCAL	
10040		Multicast organization-local address.
10041	IN6_IS_ADDR_MC_GLOBAL	
10042		Multicast global address.
10043	IN6_IS_ADDR_LINKLOCAL and IN6_IS_ADDR_SITELOCAL return true only for the two	
10044	local-use IPv6 unicast addresses. They do not return true for multicast addresses of either link-	
10045	local or site-local scope.	

10046 **APPLICATION USAGE**

10047       None.

10048 **RATIONALE**

10049       None.

10050 **FUTURE DIRECTIONS**

10051       None.

10052 **SEE ALSO**

10053       Section 4.8 (on page 97), <arpa/inet.h>, <inttypes.h>, <sys/socket.h>, the System Interfaces |  
10054       volume of IEEE Std 1003.1-200x, *connect()*, *getsockopt()*, *htonl()*, *htons()*, *ntohl()*, *ntohs()*, |  
10055       *sendmsg()*, *sendto()*, *setsockopt()*

10056 **CHANGE HISTORY**

10057       First released in Issue 6. Derived from the XNS, Issue 5.2 specification. |

10058       The *sin\_zero* member was removed from the **sockaddr\_in** structure as per The Open Group Base |  
10059       Resolution bwg2001-004. |

10060 **NAME**

10061       netinet/tcp.h — definitions for the Internet Transmission Control Protocol (TCP)

10062 **SYNOPSIS**

10063       #include &lt;netinet/tcp.h&gt;

10064 **DESCRIPTION**10065       The **<netinet/tcp.h>** header shall define the following macro for use as a socket option at the  
10066       IPPROTO\_TCP level:

10067       TCP\_NODELAY   Avoid coalescing of small segments.

10068       The macro shall be defined in the header. The implementation need not allow the value of the  
10069       option to be set via *setsockopt()* or retrieved via *getsockopt()*.10070 **APPLICATION USAGE**

10071       None.

10072 **RATIONALE**

10073       None.

10074 **FUTURE DIRECTIONS**

10075       None.

10076 **SEE ALSO**10077       **<sys/socket.h>**, the System Interfaces volume of IEEE Std 1003.1-200x, *getsockopt()*, *setsockopt()*10078 **CHANGE HISTORY**

10079       First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

10080 **NAME**

10081 nl\_types.h — data types

10082 **SYNOPSIS**

10083 XSI #include <nl\_types.h>

10084

10085 **DESCRIPTION**

10086 The <nl\_types.h> header shall contain definitions of at least the following types:

10087 **nl\_catd** Used by the message catalog functions *catopen()*, *catgets()*, and *catclose()*  
 10088 to identify a catalog descriptor.

10089 **nl\_item** Used by *nl\_langinfo()* to identify items of *langinfo* data. Values of objects  
 10090 of type **nl\_item** are defined in <langinfo.h>.

10091 The <nl\_types.h> header shall contain definitions of at least the following constants:

10092 **NL\_SETD** Used by *genocat* when no *\$set* directive is specified in a message text source  
 10093 file; see the Internationalization Guide. This constant can be passed as the  
 10094 value of *set\_id* on subsequent calls to *catgets()* (that is, to retrieve  
 10095 messages from the default message set). The value of **NL\_SETD** is  
 10096 implementation-defined.

10097 **NL\_CAT\_LOCALE** Value that must be passed as the *offlag* argument to *catopen()* to ensure  
 10098 that message catalog selection depends on the *LC\_MESSAGES* locale  
 10099 category, rather than directly on the *LANG* environment variable.

10100 The following shall be declared as functions and may also be defined as macros. Function |  
 10101 prototypes shall be provided. |

```
10102 int      catclose(nl_catd);
10103 char     *catgets(nl_catd, int, int, const char *);
10104 nl_catd  catopen(const char *, int);
```

10105 **APPLICATION USAGE**

10106 None.

10107 **RATIONALE**

10108 None.

10109 **FUTURE DIRECTIONS**

10110 None.

10111 **SEE ALSO**

10112 <langinfo.h>, the System Interfaces volume of IEEE Std 1003.1-200x, *catclose()*, *catgets()*,  
 10113 *catopen()*, *nl\_langinfo()*, the Shell and Utilities volume of IEEE Std 1003.1-200x, *genocat*

10114 **CHANGE HISTORY**

10115 First released in Issue 2.

10116 **NAME**

10117 poll.h — definitions for the poll() function

10118 **SYNOPSIS**

10119 XSI #include &lt;poll.h&gt;

10120

10121 **DESCRIPTION**10122 The <poll.h> header shall define the **pollfd** structure that includes at least the following  
10123 members:

10124 int fd The following descriptor being polled.

10125 short events The input event flags (see below).

10126 short revents The output event flags (see below).

10127 The <poll.h> header shall define the following type through **typedef**:10128 **nfds\_t** An unsigned integer type used for the number of file descriptors.10129 The implementation shall support one or more programming environments in which the width |  
10130 of **nfds\_t** is no greater than the width of type **long**. The names of these programming |  
10131 environments can be obtained using the *confstr()* function or the *getconf* utility. |10132 The following symbolic constants shall be defined, zero or more of which may be OR'ed together |  
10133 to form the *events* or *revents* members in the **pollfd** structure: |

10134 POLLIN Data other than high-priority data may be read without blocking.

10135 POLLRDNORM Normal data may be read without blocking.

10136 POLLRDBAND Priority data may be read without blocking.

10137 POLLPRI High priority data may be read without blocking.

10138 POLLOUT Normal data may be written without blocking.

10139 POLLWRNORM Equivalent to POLLOUT. |

10140 POLLWRBAND Priority data may be written.

10141 POLLERR An error has occurred (*revents* only).10142 POLLHUP Device has been disconnected (*revents* only).10143 POLLNVAL Invalid *fd* member (*revents* only).10144 The significance and semantics of normal, priority, and high-priority data are file and device-  
10145 specific.10146 The following shall be declared as a function and may also be defined as a macro. A function |  
10147 prototype shall be provided. |

10148 int poll(struct pollfd[], nfds\_t, int);

10149 **APPLICATION USAGE**

10150 None.

10151 **RATIONALE**

10152 None.

10153 **FUTURE DIRECTIONS**

10154 None.

10155 **SEE ALSO**

10156 The System Interfaces volume of IEEE Std 1003.1-200x, *confstr()*, *poll()*, the Shell and Utilities |  
10157 volume of IEEE Std 1003.1-200x, *getconf* |

10158 **CHANGE HISTORY**

10159 First released in Issue 4, Version 2.

10160 **Issue 6**10161 The description of the symbolic constants is updated to match the *poll()* function.10162 Text related to STREAMS has been moved to the *poll()* reference page.

10163 A note is added to the DESCRIPTION regarding the significance and semantics of normal,  
10164 priority, and high-priority data.

10165 **NAME**

10166 pthread.h — threads

10167 **SYNOPSIS**

10168 THR #include &lt;pthread.h&gt;

10169

10170 **DESCRIPTION**

10171 The &lt;pthread.h&gt; header shall define the following symbols:

10172 BAR PTHREAD\_BARRIER\_SERIAL\_THREAD

10173 PTHREAD\_CANCEL\_ASYNCHRONOUS

10174 PTHREAD\_CANCEL\_ENABLE

10175 PTHREAD\_CANCEL\_DEFERRED

10176 PTHREAD\_CANCEL\_DISABLE

10177 PTHREAD\_CANCELED

10178 PTHREAD\_COND\_INITIALIZER

10179 PTHREAD\_CREATE\_DETACHED

10180 PTHREAD\_CREATE\_JOINABLE

10181 PTHREAD\_EXPLICIT\_SCHED

10182 PTHREAD\_INHERIT\_SCHED

10183 XSI PTHREAD\_MUTEX\_DEFAULT

10184 PTHREAD\_MUTEX\_ERRORCHECK

10185 PTHREAD\_MUTEX\_INITIALIZER

10186 XSI PTHREAD\_MUTEX\_NORMAL

10187 PTHREAD\_MUTEX\_RECURSIVE

10188 PTHREAD\_ONCE\_INIT

10189 TPP|TPI PTHREAD\_PRIO\_INHERIT

10190 PTHREAD\_PRIO\_NONE

10191 PTHREAD\_PRIO\_PROTECT

10192 PTHREAD\_PROCESS\_SHARED

10193 PTHREAD\_PROCESS\_PRIVATE

10194 TPS PTHREAD\_SCOPE\_PROCESS

10195 PTHREAD\_SCOPE\_SYSTEM

10196

10197 The following types shall be defined as described in &lt;sys/types.h&gt;:

10198 pthread\_attr\_t

10199 BAR pthread\_barrier\_t

10200 pthread\_barrierattr\_t

10201 pthread\_cond\_t

10202 pthread\_condattr\_t

10203 pthread\_key\_t

10204 pthread\_mutex\_t

10205 pthread\_mutexattr\_t

10206 pthread\_once\_t

10207 pthread\_rwlock\_t

10208 pthread\_rwlockattr\_t

10209 SPI pthread\_spinlock\_t

10210 pthread\_t

10211 The following shall be declared as functions and may also be defined as macros. Function |  
10212 prototypes shall be provided. |

```

10213     int    pthread_atfork(void (*)(void), void (*)(void),
10214                          void (*)(void));
10215     int    pthread_attr_destroy(pthread_attr_t *);
10216     int    pthread_attr_getdetachstate(const pthread_attr_t *, int *);
10217 XSI    int    pthread_attr_getguardsize(const pthread_attr_t *restrict,
10218                          size_t *restrict);
10219 TPS    int    pthread_attr_getinheritsched(const pthread_attr_t *restrict,
10220                          int *restrict);
10221     int    pthread_attr_getschedparam(const pthread_attr_t *restrict,
10222                          struct sched_param *restrict);
10223 TPS    int    pthread_attr_getschedpolicy(const pthread_attr_t *restrict,
10224                          int *restrict);
10225 TPS    int    pthread_attr_getscope(const pthread_attr_t *restrict,
10226                          int *restrict);
10227 XSI    int    pthread_attr_getstack(const pthread_attr_t *restrict,
10228                          void **restrict, size_t *restrict);
10229 TSA    int    pthread_attr_getstackaddr(const pthread_attr_t *restrict,
10230                          void **restrict);
10231     int    pthread_attr_getstacksize(const pthread_attr_t *restrict,
10232                          size_t *restrict);
10233     int    pthread_attr_init(pthread_attr_t *);
10234     int    pthread_attr_setdetachstate(pthread_attr_t *, int);
10235 XSI    int    pthread_attr_setguardsize(pthread_attr_t *, size_t);
10236 TPS    int    pthread_attr_setinheritsched(pthread_attr_t *, int);
10237     int    pthread_attr_setschedparam(pthread_attr_t *restrict,
10238                          const struct sched_param *restrict);
10239 TPS    int    pthread_attr_setschedpolicy(pthread_attr_t *, int);
10240     int    pthread_attr_setscope(pthread_attr_t *, int);
10241 XSI    int    pthread_attr_setstack(pthread_attr_t *, void *, size_t);
10242 TSA    int    pthread_attr_setstackaddr(pthread_attr_t *, void *);
10243     int    pthread_attr_setstacksize(pthread_attr_t *, size_t);
10244 BAR    int    pthread_barrier_destroy(pthread_barrier_t *);
10245     int    pthread_barrier_init(pthread_barrier_t *restrict,
10246                          const pthread_barrierattr_t *restrict, unsigned);
10247     int    pthread_barrier_wait(pthread_barrier_t *);
10248     int    pthread_barrierattr_destroy(pthread_barrierattr_t *);
10249     int    pthread_barrierattr_getpshared( \
10250                          const pthread_barrierattr_t *restrict, int *restrict);
10251     int    pthread_barrierattr_init(pthread_barrierattr_t *);
10252     int    pthread_barrierattr_setpshared(pthread_barrierattr_t *, int);
10253     int    pthread_cancel(pthread_t);
10254     void   pthread_cleanup_push(void (*)(void *), void *);
10255     void   pthread_cleanup_pop(int);
10256     int    pthread_cond_broadcast(pthread_cond_t *);
10257     int    pthread_cond_destroy(pthread_cond_t *);
10258     int    pthread_cond_init(pthread_cond_t *restrict,
10259                          const pthread_condattr_t *restrict);
10260     int    pthread_cond_signal(pthread_cond_t *);
10261     int    pthread_cond_timedwait(pthread_cond_t *restrict,
10262                          pthread_mutex_t *restrict, const struct timespec *restrict);
10263     int    pthread_cond_wait(pthread_cond_t *restrict,
10264                          pthread_mutex_t *restrict);

```

```

10265     int  pthread_condattr_destroy(pthread_condattr_t *);
10266 CS   int  pthread_condattr_getclock(const pthread_condattr_t *restrict,
10267     clockid_t *restrict);
10268     int  pthread_condattr_getpshared(const pthread_condattr_t *restrict,
10269     int *restrict);
10270     int  pthread_condattr_init(pthread_condattr_t *);
10271 CS   int  pthread_condattr_setclock(pthread_condattr_t *, clockid_t);
10272     int  pthread_condattr_setpshared(pthread_condattr_t *, int);
10273     int  pthread_create(pthread_t *restrict, const pthread_attr_t *restrict,
10274     void *(*)(void *), void *restrict);
10275     int  pthread_detach(pthread_t);
10276     int  pthread_equal(pthread_t, pthread_t);
10277     void pthread_exit(void *);
10278 XSI   int  pthread_getconcurrency(void);
10279 TCT   int  pthread_getcpuclockid(pthread_t, clockid_t *);
10280 TPS   int  pthread_getschedparam(pthread_t, int *restrict,
10281     struct sched_param *restrict);
10282     void *pthread_getspecific(pthread_key_t);
10283     int  pthread_join(pthread_t, void **);
10284     int  pthread_key_create(pthread_key_t *, void (*)(void *));
10285     int  pthread_key_delete(pthread_key_t);
10286     int  pthread_mutex_destroy(pthread_mutex_t *);
10287 TPP   int  pthread_mutex_getprioceiling(const pthread_mutex_t *restrict,
10288     int *restrict);
10289     int  pthread_mutex_init(pthread_mutex_t *restrict,
10290     const pthread_mutexattr_t *restrict);
10291     int  pthread_mutex_lock(pthread_mutex_t *);
10292 TPP   int  pthread_mutex_setprioceiling(pthread_mutex_t *restrict, int,
10293     int *restrict);
10294 TMO   int  pthread_mutex_timedlock(pthread_mutex_t *,
10295     const struct timespec *);
10296     int  pthread_mutex_trylock(pthread_mutex_t *);
10297     int  pthread_mutex_unlock(pthread_mutex_t *);
10298     int  pthread_mutexattr_destroy(pthread_mutexattr_t *);
10299 TPP|TPI int  pthread_mutexattr_getprioceiling( \
10300     const pthread_mutexattr_t *restrict, int *restrict); |
10301     int  pthread_mutexattr_getprotocol(const pthread_mutexattr_t *restrict, |
10302     int *restrict); |
10303     int  pthread_mutexattr_getpshared(const pthread_mutexattr_t *restrict,
10304     int *restrict);
10305 XSI   int  pthread_mutexattr_gettype(const pthread_mutexattr_t *restrict,
10306     int *restrict);
10307     int  pthread_mutexattr_init(pthread_mutexattr_t *);
10308 TPP|TPI int  pthread_mutexattr_setprioceiling(pthread_mutexattr_t *, int);
10309     int  pthread_mutexattr_setprotocol(pthread_mutexattr_t *, int);
10310     int  pthread_mutexattr_setpshared(pthread_mutexattr_t *, int);
10311 XSI   int  pthread_mutexattr_settype(pthread_mutexattr_t *, int);
10312     int  pthread_once(pthread_once_t *, void (*)(void));
10313     int  pthread_rwlock_destroy(pthread_rwlock_t *);
10314     int  pthread_rwlock_init(pthread_rwlock_t *restrict,
10315     const pthread_rwlockattr_t *restrict);
10316     int  pthread_rwlock_rdlock(pthread_rwlock_t *);

```

```

10317     int    pthread_rwlock_timedrdlock(pthread_rwlock_t *restrict,
10318         const struct timespec *restrict);
10319     int    pthread_rwlock_timedwrlock(pthread_rwlock_t *restrict,
10320         const struct timespec *restrict);
10321     int    pthread_rwlock_tryrdlock(pthread_rwlock_t *);
10322     int    pthread_rwlock_trywrlock(pthread_rwlock_t *);
10323     int    pthread_rwlock_unlock(pthread_rwlock_t *);
10324     int    pthread_rwlock_wrlock(pthread_rwlock_t *);
10325     int    pthread_rwlockattr_destroy(pthread_rwlockattr_t *);
10326     int    pthread_rwlockattr_getpshared(const pthread_rwlockattr_t *restrict,
10327         int *restrict);
10328     int    pthread_rwlockattr_init(pthread_rwlockattr_t *);
10329     int    pthread_rwlockattr_setpshared(pthread_rwlockattr_t *, int);
10330     pthread_t
10331         pthread_self(void);
10332     int    pthread_setcancelstate(int, int *);
10333     int    pthread_setcanceltype(int, int *);
10334 XSI     int    pthread_setconcurrency(int);
10335 TPS     int    pthread_setschedparam(pthread_t, int,
10336         const struct sched_param *);
10337 THR TPS int    pthread_setschedprio(pthread_t, int);
10338     int    pthread_setspecific(pthread_key_t, const void *);
10339 SPI     int    pthread_spin_destroy(pthread_spinlock_t *);
10340     int    pthread_spin_init(pthread_spinlock_t *, int);
10341     int    pthread_spin_lock(pthread_spinlock_t *);
10342     int    pthread_spin_trylock(pthread_spinlock_t *);
10343     int    pthread_spin_unlock(pthread_spinlock_t *);
10344     void   pthread_testcancel(void);

```

10345 Inclusion of the <pthread.h> header shall make symbols defined in the headers <sched.h> and  
10346 <time.h> visible.

10347 **APPLICATION USAGE**

10348 None.

10349 **RATIONALE**

10350 None.

10351 **FUTURE DIRECTIONS**

10352 None.

10353 **SEE ALSO**

10354 <sched.h>, <time.h>, the System Interfaces volume of IEEE Std 1003.1-200x,  
10355 *pthread\_attr\_getguardsize()*, *pthread\_attr\_init()*, *pthread\_attr\_setscope()*, *pthread\_barrier\_destroy()*,  
10356 *pthread\_barrier\_init()*, *pthread\_barrier\_wait()*, *pthread\_barrierattr\_destroy()*,  
10357 *pthread\_barrierattr\_getpshared()*, *pthread\_barrierattr\_init()*, *pthread\_barrierattr\_setpshared()*,  
10358 *pthread\_cancel()*, *pthread\_cleanup\_pop()*, *pthread\_cond\_init()*, *pthread\_cond\_signal()*,  
10359 *pthread\_cond\_wait()*, *pthread\_condattr\_getclock()*, *pthread\_condattr\_init()*,  
10360 *pthread\_condattr\_setclock()*, *pthread\_create()*, *pthread\_detach()*, *pthread\_equal()*, *pthread\_exit()*,  
10361 *pthread\_getconcurrency()*, *pthread\_getcpuclid()*, *pthread\_getschedparam()*, *pthread\_join()*,  
10362 *pthread\_key\_create()*, *pthread\_key\_delete()*, *pthread\_mutex\_init()*, *pthread\_mutex\_lock()*,  
10363 *pthread\_mutex\_setprioceiling()*, *pthread\_mutex\_timedlock()*, *pthread\_mutexattr\_init()*,  
10364 *pthread\_mutexattr\_gettype()*, *pthread\_mutexattr\_setprotocol()*, *pthread\_once()*,  
10365 *pthread\_rwlock\_destroy()*, *pthread\_rwlock\_init()*, *pthread\_rwlock\_rdlock()*,  
10366 *pthread\_rwlock\_timedrdlock()*, *pthread\_rwlock\_timedwrlock()*, *pthread\_rwlock\_tryrdlock()*,

10367 *pthread\_rwlock\_trywrlock()*, *pthread\_rwlock\_unlock()*, *pthread\_rwlock\_wrlock()*,  
 10368 *pthread\_rwlockattr\_destroy()*, *pthread\_rwlockattr\_getpshared()*, *pthread\_rwlockattr\_init()*,  
 10369 *pthread\_rwlockattr\_setpshared()*, *pthread\_self()*, *pthread\_setcancelstate()*, *pthread\_setspecific()*,  
 10370 *pthread\_spin\_destroy()*, *pthread\_spin\_init()*, *pthread\_spin\_lock()*, *pthread\_spin\_trylock()*,  
 10371 *pthread\_spin\_unlock()*

10372 **CHANGE HISTORY**

10373 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

10374 **Issue 6**

10375 The RTT margin markers are now broken out into their POSIX options.

10376 The Open Group Corrigendum U021/9 is applied, correcting the prototype for the  
 10377 *pthread\_cond\_wait()* function.

10378 The Open Group Corrigendum U026/2 is applied correcting the prototype for the  
 10379 *pthread\_setschedparam()* function so that its second argument is of type **int**.

10380 The *pthread\_getcpuclockid()* and *pthread\_mutex\_timedlock()* functions are added for alignment  
 10381 with IEEE Std 1003.1d-1999.

10382 The following functions are added for alignment with IEEE Std 1003.1j-2000:

10383 *pthread\_barrier\_destroy()*, *pthread\_barrier\_init()*, *pthread\_barrier\_wait()*,  
 10384 *pthread\_barrierattr\_destroy()*, *pthread\_barrierattr\_getpshared()*, *pthread\_barrierattr\_init()*,  
 10385 *pthread\_barrierattr\_setpshared()*, *pthread\_condattr\_getclock()*, *pthread\_condattr\_setclock()*,  
 10386 *pthread\_rwlock\_timedrdlock()*, *pthread\_rwlock\_timedwrlock()*, *pthread\_spin\_destroy()*,  
 10387 *pthread\_spin\_init()*, *pthread\_spin\_lock()*, *pthread\_spin\_trylock()*, and *pthread\_spin\_unlock()*.

10388 PTHREAD\_RWLOCK\_INITIALIZER is deleted for alignment with IEEE Std 1003.1j-2000.

10389 Functions previously marked as part of the Read-Write Locks option are now moved to the  
 10390 Threads option.

10391 The **restrict** keyword is added to the prototypes for *pthread\_attr\_getguardsize()*,  
 10392 *pthread\_attr\_getinheritsched()*, *pthread\_attr\_getschedparam()*, *pthread\_attr\_getschedpolicy()*,  
 10393 *pthread\_attr\_getscope()*, *pthread\_attr\_getstackaddr()*, *pthread\_attr\_getstacksize()*,  
 10394 *pthread\_attr\_setschedparam()*, *pthread\_barrier\_init()*, *pthread\_barrierattr\_getpshared()*,  
 10395 *pthread\_cond\_init()*, *pthread\_cond\_signal()*, *pthread\_cond\_timedwait()*, *pthread\_cond\_wait()*,  
 10396 *pthread\_condattr\_getclock()*, *pthread\_condattr\_getpshared()*, *pthread\_create()*,  
 10397 *pthread\_getschedparam()*, *pthread\_mutex\_getprioceiling()*, *pthread\_mutex\_init()*,  
 10398 *pthread\_mutex\_setprioceiling()*, *pthread\_mutexattr\_getprioceiling()*, *pthread\_mutexattr\_getprotocol()*,  
 10399 *pthread\_mutexattr\_getpshared()*, *pthread\_mutexattr\_gettype()*, *pthread\_rwlock\_init()*,  
 10400 *pthread\_rwlock\_timedrdlock()*, *pthread\_rwlock\_timedwrlock()*, *pthread\_rwlockattr\_getpshared()*, and  
 10401 *pthread\_sigmask()*.

10402 IEEE PASC Interpretation 1003.1 #86 is applied, allowing the symbols from <sched.h> and  
 10403 <time.h> to be made visible when <pthread.h> is included. Previously this was an XSI  
 10404 extension.

10405 IEEE PASC Interpretation 1003.1c #42 is applied, removing the requirement for prototypes for  
 10406 the *pthread\_kill()* and *pthread\_sigmask()* functions. These are required to be in the <signal.h>  
 10407 header. They are allowed here through the name space rules. |

10408 IEEE PASC Interpretation 1003.1 #96 is applied, adding the *pthread\_setschedprio()* function. |

10409 **NAME**

10410 pwd.h — password structure

10411 **SYNOPSIS**

10412 #include <pwd.h>

10413 **DESCRIPTION**

10414 The <pwd.h> header shall provide a definition for **struct passwd**, which shall include at least the  
 10415 following members:

10416	char	*pw_name	User's login name.
10417	uid_t	pw_uid	Numerical user ID.
10418	gid_t	pw_gid	Numerical group ID.
10419	char	*pw_dir	Initial working directory.
10420	char	*pw_shell	Program to use as shell.

10421 The **gid\_t** and **uid\_t** types shall be defined as described in <sys/types.h>.

10422 The following shall be declared as functions and may also be defined as macros. Function  
 10423 prototypes shall be provided.

```

10424 struct passwd *getpwnam(const char *);
10425 struct passwd *getpwuid(uid_t);
10426 TSF int getpwnam_r(const char *, struct passwd *, char *,
10427 size_t, struct passwd **);
10428 int getpwuid_r(uid_t, struct passwd *, char *,
10429 size_t, struct passwd **);
10430 XSI void endpwent(void);
10431 struct passwd *getpwent(void);
10432 void setpwent(void);
10433
    
```

10434 **APPLICATION USAGE**

10435 None.

10436 **RATIONALE**

10437 None.

10438 **FUTURE DIRECTIONS**

10439 None.

10440 **SEE ALSO**

10441 <sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-200x, *endpwent()*, *getpwnam()*,  
 10442 *getpwuid()*

10443 **CHANGE HISTORY**

10444 First released in Issue 1.

10445 **Issue 5**

10446 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

10447 **Issue 6**

10448 The following new requirements on POSIX implementations derive from alignment with the  
 10449 Single UNIX Specification:

- 10450 • The **gid\_t** and **uid\_t** types are mandated.
- 10451 • The *getpwnam\_r()* and *getpwuid\_r()* functions are marked as part of the  
 10452 `_POSIX_THREAD_SAFE_FUNCTIONS` option.

10453 **NAME**10454 `regex.h` — regular expression matching types10455 **SYNOPSIS**10456 `#include <regex.h>`10457 **DESCRIPTION**10458 The **<regex.h>** header shall define the structures and symbolic constants used by the *regcomp()*,  
10459 *regexec()*, *regerror()*, and *regfree()* functions.10460 The structure type **regex\_t** shall contain at least the following member:10461 `size_t re_nsub` Number of parenthesized subexpressions.10462 The type **size\_t** shall be defined as described in **<sys/types.h>**.10463 The type **regoff\_t** shall be defined as a signed integer type that can hold the largest value that  
10464 can be stored in either a type **off\_t** or type **ssize\_t**. The structure type **regmatch\_t** shall contain  
10465 at least the following members:10466 `regoff_t rm_so` Byte offset from start of string  
10467 to start of substring.  
10468 `regoff_t rm_eo` Byte offset from start of string of the  
10469 first character after the end of substring.10470 Values for the *cflags* parameter to the *regcomp()* function:10471 **REG\_EXTENDED** Use Extended Regular Expressions.10472 **REG\_ICASE** Ignore case in match.10473 **REG\_NOSUB** Report only success or fail in *regexec()*.10474 **REG\_NEWLINE** Change the handling of newline.10475 Values for the *eflags* parameter to the *regexec()* function:10476 **REG\_NOTBOL** The circumflex character ('^'), when taken as a special character, does  
10477 not match the beginning of *string*.10478 **REG\_NOTEOL** The dollar sign ('\$'), when taken as a special character, does not match  
10479 the end of *string*.

10480 The following constants shall be defined as error return values:

10481 **REG\_NOMATCH** *regexec()* failed to match.10482 **REG\_BADPAT** Invalid regular expression.10483 **REG\_ECOLLATE** Invalid collating element referenced.10484 **REG\_ECTYPE** Invalid character class type referenced.10485 **REG\_EESCAPE** Trailing '\\' in pattern.10486 **REG\_ESUBREG** Number in *\digit* invalid or in error.10487 **REG\_EBRACK** "[ ]" imbalance.10488 **REG\_EPAREN** "\(\)" or "()" imbalance.10489 **REG\_EBRACE** "\{\}" imbalance.10490 **REG\_BADBR** Content of "\{\}" invalid: not a number, number too large, more than  
10491 two numbers, first larger than second.

10492        REG\_ERANGE        Invalid endpoint in range expression.

10493        REG\_ESPACE        Out of memory.

10494        REG\_BADRPT        '?', '\*', or '+' not preceded by valid regular expression.

10495 OB     REG\_ENOSYS        Reserved.

10496        The following shall be declared as functions and may also be defined as macros. Function |  
 10497        prototypes shall be provided. |

```

10498        int     regcomp(regex_t *restrict, const char *restrict, int);
10499        size_t regerror(int, const regex_t *restrict, char *restrict, size_t);
10500        int     regexec(const regex_t *restrict, const char *restrict, size_t,
10501                        regmatch_t[restrict], int);
10502        void    regfree(regex_t *);
    
```

10503        The implementation may define additional macros or constants using names beginning with  
 10504        REG\_.

10505 **APPLICATION USAGE**

10506        None.

10507 **RATIONALE**

10508        None.

10509 **FUTURE DIRECTIONS**

10510        None.

10511 **SEE ALSO**

10512        The System Interfaces volume of IEEE Std 1003.1-200x, *regcomp()*, the Shell and Utilities volume  
 10513        of IEEE Std 1003.1-200x

10514 **CHANGE HISTORY**

10515        First released in Issue 4.

10516        Originally derived from the ISO POSIX-2 standard.

10517 **Issue 6**

10518        The REG\_ENOSYS constant is marked obsolescent.

10519        The **restrict** keyword is added to the prototypes for *regcomp()*, *regerror()*, and *regexec()*.

10520        A statement is added that the **size\_t** type is defined as described in <sys/types.h>.

10521 NAME

10522 sched.h — execution scheduling (REALTIME)

10523 SYNOPSIS

10524 PS #include <sched.h>

10525

10526 DESCRIPTION

10527 The <sched.h> header shall define the **sched\_param** structure, which contains the scheduling  
10528 parameters required for implementation of each supported scheduling policy. This structure  
10529 shall contain at least the following member:

10530 int sched\_priority Process execution scheduling priority.

10531 SS|TSP In addition, if **\_POSIX\_SPORADIC\_SERVER** or **\_POSIX\_THREAD\_SPORADIC\_SERVER** is  
10532 defined, the **sched\_param** structure defined in <sched.h> shall contain the following members  
10533 in addition to those specified above:

10534 int sched\_ss\_low\_priority Low scheduling priority for  
10535 sporadic server.  
10536 struct timespec sched\_ss\_repl\_period Replenishment period for  
10537 sporadic server.  
10538 struct timespec sched\_ss\_init\_budget Initial budget for sporadic server.  
10539 int sched\_ss\_max\_repl Maximum pending replenishments for  
10540 sporadic server.

10541

10542 Each process is controlled by an associated scheduling policy and priority. Associated with each  
10543 policy is a priority range. Each policy definition specifies the minimum priority range for that  
10544 policy. The priority ranges for each policy may overlap the priority ranges of other policies.

10545 Four scheduling policies are defined; others may be defined by the implementation. The four  
10546 standard policies are indicated by the values of the following symbolic constants:

10547 SCHED\_FIFO First in-first out (FIFO) scheduling policy.

10548 SCHED\_RR Round robin scheduling policy.

10549 SS|TSP SCHED\_SPORADIC Sporadic server scheduling policy.

10550 SCHED\_OTHER Another scheduling policy.

10551 The values of these constants are distinct.

10552 The following shall be declared as functions and may also be defined as macros. Function |  
10553 prototypes shall be provided. |

10554 int sched\_get\_priority\_max(int);  
10555 int sched\_get\_priority\_min(int);  
10556 int sched\_getparam(pid\_t, struct sched\_param \*);  
10557 int sched\_getscheduler(pid\_t);  
10558 int sched\_rr\_get\_interval(pid\_t, struct timespec \*);  
10559 int sched\_setparam(pid\_t, const struct sched\_param \*);  
10560 int sched\_setscheduler(pid\_t, int, const struct sched\_param \*);  
10561 int sched\_yield(void);

10562 Inclusion of the <sched.h> header makes symbols defined in the header <time.h> visible.

10563 **APPLICATION USAGE**

10564           None.

10565 **RATIONALE**

10566           None.

10567 **FUTURE DIRECTIONS**

10568           None.

10569 **SEE ALSO**

10570           &lt;time.h&gt;

10571 **CHANGE HISTORY**

10572           First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

10573 **Issue 6**

10574           The &lt;sched.h&gt; header is marked as part of the Process Scheduling option.

10575           Sporadic server members are added to the **sched\_param** structure, and the SCHED\_SPORADIC  
10576           scheduling policy is added for alignment with IEEE Std 1003.1d-1999.10577           IEEE PASC Interpretation 1003.1 #108 is applied, correcting the **sched\_param** structure whose  
10578           members *sched\_ss\_repl\_period* and *sched\_ss\_init\_budget* members should be type **struct timespec**  
10579           and not **timespec**.

10580 **NAME**

10581 search.h — search tables

10582 **SYNOPSIS**

10583 XSI #include &lt;search.h&gt;

10584

10585 **DESCRIPTION**

10586 The <search.h> header shall define the **ENTRY** type for structure **entry** which shall include the  
 10587 following members:

```
10588 char    *key
10589 void    *data
```

10590 and shall define **ACTION** and **VISIT** as enumeration data types through type definitions as  
 10591 follows:

```
10592 enum { FIND, ENTER } ACTION;
10593 enum { preorder, postorder, endorder, leaf } VISIT;
```

10594 The **size\_t** type shall be defined as described in <sys/types.h>.

10595 The following shall be declared as functions and may also be defined as macros. Function |  
 10596 prototypes shall be provided. |

```
10597 int    hcreate(size_t);
10598 void   hdestroy(void);
10599 ENTRY *hsearch(ENTRY, ACTION);
10600 void   insque(void *, void *);
10601 void   *lfind(const void *, const void *, size_t *,
10602             size_t, int (*)(const void *, const void *));
10603 void   *lsearch(const void *, void *, size_t *,
10604               size_t, int (*)(const void *, const void *));
10605 void   remque(void *);
10606 void   *tdelete(const void *restrict, void **restrict,
10607                int (*)(const void *, const void *));
10608 void   *tfind(const void *, void *const *,
10609              int (*)(const void *, const void *));
10610 void   *tsearch(const void *, void **,
10611                int (*)(const void *, const void *));
10612 void   twalk(const void *,
10613             void (*)(const void *, VISIT, int ));
```

10614 **APPLICATION USAGE**

10615 None.

10616 **RATIONALE**

10617 None.

10618 **FUTURE DIRECTIONS**

10619 None.

10620 **SEE ALSO**

10621 <sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-200x, *hcreate()*, *insque()*,  
 10622 *lsearch()*, *remque()*, *tsearch()*

10623 **CHANGE HISTORY**

10624 First released in Issue 1. Derived from Issue 1 of the SVID.

10625 **Issue 6**

10626 The Open Group Corrigendum U021/6 is applied updating the prototypes for *tdelete()* and  
10627 *tsearch()*.

10628 The **restrict** keyword is added to the prototype for *tdelete()*.

10629 **NAME**10630 semaphore.h — semaphores (**REALTIME**)10631 **SYNOPSIS**10632 SEM `#include <semaphore.h>`

10633

10634 **DESCRIPTION**

10635 The **<semaphore.h>** header shall define the **sem\_t** type, used in performing semaphore  
10636 operations. The semaphore may be implemented using a file descriptor, in which case  
10637 applications are able to open up at least a total of {OPEN\_MAX} files and semaphores. The  
10638 symbol SEM\_FAILED shall be defined (see *sem\_open()*).

10639 The following shall be declared as functions and may also be defined as macros. Function  
10640 prototypes shall be provided.

```
10641 int    sem_close(sem_t *);  
10642 int    sem_destroy(sem_t *);  
10643 int    sem_getvalue(sem_t *restrict, int *restrict);  
10644 int    sem_init(sem_t *, int, unsigned);  
10645 sem_t *sem_open(const char *, int, ...);  
10646 int    sem_post(sem_t *);  
10647 TMO   int    sem_timedwait(sem_t *restrict, const struct timespec *restrict);  
10648 int    sem_trywait(sem_t *);  
10649 int    sem_unlink(const char *);  
10650 int    sem_wait(sem_t *);
```

10651 Inclusion of the **<semaphore.h>** header may make visible symbols defined in the headers  
10652 **<fcntl.h>** and **<sys/types.h>**.

10653 **APPLICATION USAGE**

10654 None.

10655 **RATIONALE**

10656 None.

10657 **FUTURE DIRECTIONS**

10658 None.

10659 **SEE ALSO**

10660 **<fcntl.h>**, **<sys/types.h>**, the System Interfaces volume of IEEE Std 1003.1-200x, *sem\_destroy()*,  
10661 *sem\_getvalue()*, *sem\_init()*, *sem\_open()*, *sem\_post()*, *sem\_timedwait()*, *sem\_trywait()*, *sem\_unlink()*,  
10662 *sem\_wait()*

10663 **CHANGE HISTORY**

10664 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

10665 **Issue 6**10666 The **<semaphore.h>** header is marked as part of the Semaphores option.

10667 The Open Group Corrigendum U021/3 is applied, adding a description of SEM\_FAILED.

10668 The *sem\_timedwait()* function is added for alignment with IEEE Std 1003.1d-1999.10669 The **restrict** keyword is added to the prototypes for *sem\_getvalue()* and *sem\_timedwait()*.

10670 **NAME**

10671 setjmp.h — stack environment declarations

10672 **SYNOPSIS**

10673 #include <setjmp.h>

10674 **DESCRIPTION**

10675 CX Some of the functionality described on this reference page extends the ISO C standard.  
 10676 Applications shall define the appropriate feature test macro (see the System Interfaces volume of  
 10677 IEEE Std 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of these  
 10678 symbols in this header.

10679 CX The <setjmp.h> header shall define the array types **jmp\_buf** and **sigjmp\_buf**.

10680 The following shall be declared as functions and may also be defined as macros. Function |  
 10681 prototypes shall be provided. |

```
10682 void longjmp(jmp_buf, int);
10683 CX void siglongjmp(sigjmp_buf, int);
10684 XSI void _longjmp(jmp_buf, int);
10685
```

10686 The following may be declared as a function, or defined as a macro, or both. Function prototypes |  
 10687 shall be provided. |

```
10688 int setjmp(jmp_buf);
10689 CX int sigsetjmp(sigjmp_buf, int);
10690 XSI int _setjmp(jmp_buf);
10691
```

10692 **APPLICATION USAGE**

10693 None.

10694 **RATIONALE**

10695 None.

10696 **FUTURE DIRECTIONS**

10697 None.

10698 **SEE ALSO**

10699 The System Interfaces volume of IEEE Std 1003.1-200x, *longjmp()*, *\_longjmp()*, *setjmp()*,  
 10700 *siglongjmp()*, *sigsetjmp()*

10701 **CHANGE HISTORY**

10702 First released in Issue 1.

10703 **Issue 6**

10704 Extensions beyond the ISO C standard are now marked.

10705 NAME

10706 signal.h — signals

10707 SYNOPSIS

10708 #include <signal.h>

10709 DESCRIPTION

10710 CX Some of the functionality described on this reference page extends the ISO C standard.  
10711 Applications shall define the appropriate feature test macro (see the System Interfaces volume of  
10712 IEEE Std 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of these  
10713 symbols in this header.

10714 The <signal.h> header shall define the following symbolic constants, each of which expands to a  
10715 distinct constant expression of the type:

10716 void (\*)(int)

10717 whose value matches no declarable function.

10718 SIG\_DFL Request for default signal handling.

10719 SIG\_ERR Return value from *signal()* in case of error.

10720 CX SIG\_HOLD Request that signal be held.

10721 SIG\_IGN Request that signal be ignored.

10722 The following data types shall be defined through **typedef**:

10723 **sig\_atomic\_t** Possibly volatile-qualified integer type of an object that can be accessed as  
10724 an atomic entity, even in the presence of asynchronous interrupts.

10725 CX **sigset\_t** Integer or structure type of an object used to represent sets of signals.

10726 CX **pid\_t** As described in <sys/types.h>.

10727 RTS The <signal.h> header shall define the **sigevent** structure, which has at least the following  
10728 members:

10729	int	sigev_notify	Notification type.
10730	int	sigev_signo	Signal number.
10731	union sigval	sigev_value	Signal value.
10732	void (*)(union sigval)	sigev_notify_function	Notification function.
10733	(pthread_attr_t *)	sigev_notify_attributes	Notification attributes.

10734 The following values of *sigev\_notify* shall be defined:

10735 SIGEV\_NONE No asynchronous notification is delivered when the event of interest  
10736 occurs.

10737 SIGEV\_SIGNAL A queued signal, with an application-defined value, is generated when  
10738 the event of interest occurs.

10739 SIGEV\_THREAD A notification function is called to perform notification.

10740 The **sigval** union shall be defined as:

10741	int	sival_int	Integer signal value.
10742	void *	sival_ptr	Pointer signal value.

10743 This header shall also declare the macros SIGRTMIN and SIGRTMAX, which evaluate to integer  
10744 expressions, and specify a range of signal numbers that are reserved for application use and for  
10745 which the realtime signal behavior specified in this volume of IEEE Std 1003.1-200x is supported.

10746 The signal numbers in this range do not overlap any of the signals specified in the following  
 10747 table.

10748 The range SIGRTMIN through SIGRTMAX inclusive shall include at least {RTSIG\_MAX} signal  
 10749 numbers.

10750 It is implementation-defined whether realtime signal behavior is supported for other signals.

10751 This header also declares the constants that are used to refer to the signals that occur in the  
 10752 system. Signals defined here begin with the letters SIG. Each of the signals have distinct positive  
 10753 integer values. The value 0 is reserved for use as the null signal (see *kill()*). Additional  
 10754 implementation-defined signals may occur in the system.

10755 cx The ISO C standard only requires the signal names SIGABRT, SIGFPE, SIGILL, SIGINT,  
 10756 SIGSEGV, and SIGTERM to be defined.

10757 The following signals shall be supported on all implementations (default actions are explained  
 10758 below the table):

10759

10760

Signal	Default Action	Description
SIGABRT	A	Process abort signal.
SIGALRM	T	Alarm clock.
SIGBUS	A	Access to an undefined portion of a memory object.
SIGCHLD	I	Child process terminated, stopped, or continued.
SIGCONT	C	Continue executing, if stopped.
SIGFPE	A	Erroneous arithmetic operation.
SIGHUP	T	Hangup.
SIGILL	A	Illegal instruction.
SIGINT	T	Terminal interrupt signal.
SIGKILL	T	Kill (cannot be caught or ignored).
SIGPIPE	T	Write on a pipe with no one to read it.
SIGQUIT	A	Terminal quit signal.
SIGSEGV	A	Invalid memory reference.
SIGSTOP	S	Stop executing (cannot be caught or ignored).
SIGTERM	T	Termination signal.
SIGTSTP	S	Terminal stop signal.
SIGTTIN	S	Background process attempting read.
SIGTTOU	S	Background process attempting write.
SIGUSR1	T	User-defined signal 1.
SIGUSR2	T	User-defined signal 2.
SIGPOLL	T	Pollable event.
SIGPROF	T	Profiling timer expired.
SIGSYS	A	Bad system call.
SIGTRAP	A	Trace/breakpoint trap.
SIGURG	I	High bandwidth data is available at a socket.
SIGVTALRM	T	Virtual timer expired.
SIGXCPU	A	CPU time limit exceeded.
SIGXFSZ	A	File size limit exceeded.

10790 The default actions are as follows:

10791 T Abnormal termination of the process. The process is terminated with all the consequences  
 10792 of *\_exit()* except that the status made available to *wait()* and *waitpid()* indicates abnormal  
 10793 termination by the specified signal.

10794 A Abnormal termination of the process. |  
10795 XSI Additionally, implementation-defined abnormal termination actions, such as creation of a |  
10796 core file, may occur. |  
10797 I Ignore the signal.  
10798 S Stop the process.  
10799 C Continue the process, if it is stopped; otherwise, ignore the signal.

10800 CX The header shall provide a declaration of **struct sigaction**, including at least the following  
10801 members:

```

10802 void (*sa_handler)(int) What to do on receipt of signal.
10803 sigset_t sa_mask Set of signals to be blocked during execution
10804 of the signal handling function.
10805 int sa_flags Special flags.
10806 void (*)(int, siginfo_t *, void *) sa_sigaction
10807 Pointer to signal handler function or one
10808 of the macros SIG_IGN or SIG_DFL.

```

10809

10810 XSI The storage occupied by *sa\_handler* and *sa\_sigaction* may overlap, and a portable program must  
10811 not use both simultaneously.

10812 The following shall be declared as constants:

10813 CX SA\_NOCLDSTOP Do not generate SIGCHLD when children stop |  
10814 XSI or stopped children continue. |

10815 CX SIG\_BLOCK The resulting set is the union of the current set and the signal set pointed  
10816 to by the argument *set*.

10817 CX SIG\_UNBLOCK The resulting set is the intersection of the current set and the complement  
10818 of the signal set pointed to by the argument *set*.

10819 CX SIG\_SETMASK The resulting set is the signal set pointed to by the argument *set*.

10820 XSI SA\_ONSTACK Causes signal delivery to occur on an alternate stack.

10821 XSI SA\_RESETHAND Causes signal dispositions to be set to SIG\_DFL on entry to signal  
10822 handlers.

10823 XSI SA\_RESTART Causes certain functions to become restartable.

10824 XSI SA\_SIGINFO Causes extra information to be passed to signal handlers at the time of  
10825 receipt of a signal.

10826 XSI SA\_NOCLDWAIT Causes implementations not to create zombie processes on child death.

10827 XSI SA\_NODEFER Causes signal not to be automatically blocked on entry to signal handler.

10828 XSI SS\_ONSTACK Process is executing on an alternate signal stack.

10829 XSI SS\_DISABLE Alternate signal stack is disabled.

10830 XSI MINSIGSTKSZ Minimum stack size for a signal handler.

10831 XSI SIGSTKSZ Default size in bytes for the alternate signal stack.

10832 XSI The **ucontext\_t** structure shall be defined through **typedef** as described in <ucontext.h>.

10833 The **mcontext\_t** type shall be defined through **typedef** as described in <ucontext.h>.

10834 The <signal.h> header shall define the **stack\_t** type as a structure that includes at least the  
 10835 following members:

10836	void	*ss_sp	Stack base or pointer.
10837	size_t	ss_size	Stack size.
10838	int	ss_flags	Flags.

10839 The <signal.h> header shall define the **sigstack** structure that includes at least the following  
 10840 members:

10841	int	ss_onstack	Non-zero when signal stack is in use.
10842	void	*ss_sp	Signal stack pointer.

10843

10844 CX The <signal.h> header shall define the **siginfo\_t** type as a structure that includes at least the  
 10845 following members:

10846 CX	int	si_signo	Signal number.
10847 XSI	int	si_errno	If non-zero, an <i>errno</i> value associated with 10848 this signal, as defined in <errno.h>.
10849 CX	int	si_code	Signal code.
10850 XSI	pid_t	si_pid	Sending process ID.
10851	uid_t	si_uid	Real user ID of sending process.
10852	void	*si_addr	Address of faulting instruction.
10853	int	si_status	Exit value or signal.
10854	long	si_band	Band event for SIGPOLL.
10855 RTS	union signal	si_value	Signal value.
10856			

10857 The macros specified in the **Code** column of the following table are defined for use as values of  
 10858 XSI *si\_code* that are signal-specific or non-signal-specific reasons why the signal was generated.

10859

10860

10861 XSI

10862

10863

10864

10865

10866

10867

10868

10869

10870

10871

10872

10873

10874

10875

10876

10877

10878

10879

10880

10881

10882

10883

10884

10885

10886

10887

10888

10889

10890

10891

10892

10893

10894

10895

10896 CX

10897

10898

10899

10900

10901

10902

Signal	Code	Reason
SIGILL	ILL_ILLOPC	Illegal opcode.
	ILL_ILLOPN	Illegal operand.
	ILL_ILLADR	Illegal addressing mode.
	ILL_ILLTRP	Illegal trap.
	ILL_PRVOPC	Privileged opcode.
	ILL_PRVREG	Privileged register.
	ILL_COPROC	Coprocessor error.
	ILL_BADSTK	Internal stack error.
SIGFPE	FPE_INTDIV	Integer divide by zero.
	FPE_INTOVF	Integer overflow.
	FPE_FLTDIV	Floating-point divide by zero.
	FPE_FLTOVF	Floating-point overflow.
	FPE_FLTUND	Floating-point underflow.
	FPE_FLTRES	Floating-point inexact result.
	FPE_FLTINV	Invalid floating-point operation.
	FPE_FLTSUB	Subscript out of range.
SIGSEGV	SEGV_MAPERR	Address not mapped to object.
	SEGV_ACCERR	Invalid permissions for mapped object.
SIGBUS	BUS_ADRALN	Invalid address alignment.
	BUS_ADRERR	Non-existent physical address.
	BUS_OBJERR	Object specific hardware error.
SIGTRAP	TRAP_BRKPT	Process breakpoint.
	TRAP_TRACE	Process trace trap.
SIGCHLD	CLD_EXITED	Child has exited.
	CLD_KILLED	Child has terminated abnormally and did not create a core file.
	CLD_DUMPED	Child has terminated abnormally and created a core file.
	CLD_TRAPPED	Traced child has trapped.
	CLD_STOPPED	Child has stopped.
	CLD_CONTINUED	Stopped child has continued.
SIGPOLL	POLL_IN	Data input available.
	POLL_OUT	Output buffers available.
	POLL_MSG	Input message available.
	POLL_ERR	I/O error.
	POLL_PRI	High priority input available.
	POLL_HUP	Device disconnected.
Any	SI_USER	Signal sent by <i>kill()</i> .
	SI_QUEUE	Signal sent by the <i>sigqueue()</i> .
	SI_TIMER	Signal generated by expiration of a timer set by <i>timer_settime()</i> .
	SI_ASYNCIO	Signal generated by completion of an asynchronous I/O request.
	SI_MESGQ	Signal generated by arrival of a message on an empty message queue.

10903 XSI

10904

10905

10906

10907

Implementations may support additional *si\_code* values not included in this list, may generate values included in this list under circumstances other than those described in this list, and may contain extensions or limitations that prevent some values from being generated. Implementations do not generate a different value from the ones described in this list for circumstances described in this list.

10908 In addition, the following signal-specific information shall be available:

10909

10910

10911

10912

10913

10914

10915

10916

10917

10918

Signal	Member	Value
SIGILL SIGFPE	<b>void * si_addr</b>	Address of faulting instruction.
SIGSEGV SIGBUS	<b>void * si_addr</b>	Address of faulting memory reference.
SIGCHLD	<b>pid_t si_pid</b> <b>int si_status</b> <b>uid_t si_uid</b>	Child process ID. Exit value or signal. Real user ID of the process that sent the signal.
SIGPOLL	<b>long si_band</b>	Band event for POLL_IN, POLL_OUT, or POLL_MSG.

10919

For some implementations, the value of *si\_addr* may be inaccurate.

10920

The following shall be declared as functions and may also be defined as macros:

10921 XSI

10922 CX

10923 XSI

10924 THR

10925

10926

10927 CX

10928

10929

10930 XSI

10931 CX

10932

10933

10934 XSI

10935

10936

10937 CX

10938

10939 XSI

10940 CX

10941

10942 RTS

10943 XSI

10944

10945 CX

10946 RTS

10947

10948 CX

10949 RTS

10950

```
void (*bsd_signal(int, void (*)(int)))(int);
int kill(pid_t, int);
int killpg(pid_t, int);
int pthread_kill(pthread_t, int);
int pthread_sigmask(int, const sigset_t *, sigset_t *);
int raise(int);
int sigaction(int, const struct sigaction *restrict,
              struct sigaction *restrict);
int sigaddset(sigset_t *, int);
int sigaltstack(const stack_t *restrict, stack_t *restrict);
int sigdelset(sigset_t *, int);
int sigemptyset(sigset_t *);
int sigfillset(sigset_t *);
int sighold(int);
int sigignore(int);
int siginterrupt(int, int);
int sigismember(const sigset_t *, int);
void (*signal(int, void (*)(int)))(int);
int sigpause(int);
int sigpending(sigset_t *);
int sigprocmask(int, const sigset_t *restrict, sigset_t *restrict);
int sigqueue(pid_t, int, const union sigval);
int sigrelse(int);
void (*sigset(int, void (*)(int)))(int);
int sigsuspend(const sigset_t *);
int sigtimedwait(const sigset_t *restrict, siginfo_t *restrict,
                const struct timespec *restrict);
int sigwait(const sigset_t *restrict, int *restrict);
int sigwaitinfo(const sigset_t *restrict, siginfo_t *restrict);
```

- 10951 CX Inclusion of the **<signal.h>** header may make visible all symbols from the **<time.h>** header.
- 10952 **APPLICATION USAGE**
- 10953 None.
- 10954 **RATIONALE**
- 10955 None.
- 10956 **FUTURE DIRECTIONS**
- 10957 None.
- 10958 **SEE ALSO**
- 10959 **<errno.h>**, **<stropts.h>**, **<sys/types.h>**, **<time.h>**, **<ucontext.h>**, the System Interfaces volume of  
10960 IEEE Std 1003.1-200x, *alarm()*, *bsd\_signal()*, *ioctl()*, *kill()*, *killpg()*, *raise()*, *sigaction()*, *sigaddset()*,  
10961 *sigaltstack()*, *sigdelset()*, *sigemptyset()*, *sigfillset()*, *siginterrupt()*, *sigismember()*, *signal()*,  
10962 *sigpending()*, *sigprocmask()*, *sigqueue()*, *sigsuspend()*, *sigwaitinfo()*, *wait()*, *waitid()*
- 10963 **CHANGE HISTORY**
- 10964 First released in Issue 1.
- 10965 **Issue 5**
- 10966 The DESCRIPTION is updated for alignment with POSIX Realtime Extension and the POSIX  
10967 Threads Extension.
- 10968 The default action for SIGURG is changed for i to iii. The function prototype for *sigmask()* is  
10969 removed.
- 10970 **Issue 6**
- 10971 The Open Group Corrigendum U035/2 is applied. In the DESCRIPTION, the wording for  
10972 abnormal termination is clarified.
- 10973 The Open Group Corrigendum U028/8 is applied, correcting the prototype for the *sigset()*  
10974 function.
- 10975 The Open Group Corrigendum U026/3 is applied, correcting the type of the *sigev\_notify\_function*  
10976 function member of the **sigevent** structure.
- 10977 The following new requirements on POSIX implementations derive from alignment with the  
10978 Single UNIX Specification:
- 10979 • The SIGCHLD, SIGCONT, SIGSTOP, SIGTSTP, SIGTTIN, and SIGTTOU signals are now  
10980 mandated. This is also a FIPS requirement.
  - 10981 • The **pid\_t** definition is mandated.
- 10982 The RT markings are now changed to RTS to denote that the semantics are part of the Realtime  
10983 Signals Extension option.
- 10984 The **restrict** keyword is added to the prototypes for *sigaction()*, *sigaltstack()*, *sigprocmask()*,  
10985 *sigtimedwait()*, *sigwait()*, and *sigwaitinfo()*.
- 10986 IEEE PASC Interpretation 1003.1 #85 is applied, adding the statement that symbols from  
10987 **<time.h>** may be made visible when **<signal.h>** is included. Extensions beyond the ISO C  
10988 standard are now marked.

10989 **NAME**

10990 spawn.h — spawn (ADVANCED REALTIME)

10991 **SYNOPSIS**

10992 SPN #include <spawn.h>

10993

10994 **DESCRIPTION**

10995 The <spawn.h> header shall define the **posix\_spawnattr\_t** and **posix\_spawn\_file\_actions\_t**  
 10996 types used in performing spawn operations.

10997 The <spawn.h> header shall define the flags that may be set in a **posix\_spawnattr\_t** object using  
 10998 the *posix\_spawnattr\_setflags()* function:

- 10999 POSIX\_SPAWN\_RESETEIDS
- 11000 POSIX\_SPAWN\_SETPGROUP
- 11001 PS POSIX\_SPAWN\_SETSCHEDPARAM
- 11002 POSIX\_SPAWN\_SETSCHEDULER
- 11003 POSIX\_SPAWN\_SETSIGDEF
- 11004 POSIX\_SPAWN\_SETSIGMASK

11005 The following shall be declared as functions and may also be defined as macros. Function |  
 11006 prototypes shall be provided. |

```

11007 int    posix_spawn(pid_t *restrict, const char *restrict,
11008                const posix_spawn_file_actions_t *,
11009                const posix_spawnattr_t *restrict, char *const [restrict],
11010                char *const [restrict]);
11011 int    posix_spawn_file_actions_addclose(posix_spawn_file_actions_t *,
11012                int);
11013 int    posix_spawn_file_actions_adddup2(posix_spawn_file_actions_t *,
11014                int, int);
11015 int    posix_spawn_file_actions_addopen(posix_spawn_file_actions_t *restrict,
11016                int, const char *restrict, int, mode_t);
11017 int    posix_spawn_file_actions_destroy(posix_spawn_file_actions_t *);
11018 int    posix_spawn_file_actions_init(posix_spawn_file_actions_t *);
11019 int    posix_spawnattr_destroy(posix_spawnattr_t *);
11020 int    posix_spawnattr_getsigdefault(const posix_spawnattr_t *restrict,
11021                sigset_t *restrict);
11022 int    posix_spawnattr_getflags(const posix_spawnattr_t *restrict,
11023                short *restrict);
11024 int    posix_spawnattr_getpgroup(const posix_spawnattr_t *restrict,
11025                pid_t *restrict);
11026 PS int    posix_spawnattr_getschedparam(const posix_spawnattr_t *restrict,
11027                struct sched_param *restrict);
11028 int    posix_spawnattr_getschedpolicy(const posix_spawnattr_t *restrict,
11029                int *restrict);
11030 int    posix_spawnattr_getsigmask(const posix_spawnattr_t *restrict,
11031                sigset_t *restrict);
11032 int    posix_spawnattr_init(posix_spawnattr_t *);
11033 int    posix_spawnattr_setsigdefault(posix_spawnattr_t *restrict,
11034                const sigset_t *restrict);
11035 int    posix_spawnattr_setflags(posix_spawnattr_t *, short);
11036 int    posix_spawnattr_setpgroup(posix_spawnattr_t *, pid_t);
    
```

11037 PS

```
11038 int    posix_spawnattr_setschedparam(posix_spawnattr_t *restrict,  
11039         const struct sched_param *restrict);  
11040 int    posix_spawnattr_setschedpolicy(posix_spawnattr_t *, int);  
11041 int    posix_spawnattr_setsigmask(posix_spawnattr_t *restrict,  
11042         const sigset_t *restrict);  
11043 int    posix_spawnnp(pid_t *restrict, const char *restrict,  
11044         const posix_spawn_file_actions_t *,  
11045         const posix_spawnattr_t *restrict,  
11046         char *const [restrict], char *const [restrict]);
```

11047 Inclusion of the **<spawn.h>** header may make visible symbols defined in the **<sched.h>**,  
11048 **<signal.h>**, and **<sys/types.h>** headers.

#### 11049 APPLICATION USAGE

11050 None.

#### 11051 RATIONALE

11052 None.

#### 11053 FUTURE DIRECTIONS

11054 None.

#### 11055 SEE ALSO

11056 **<sched.h>**, **<semaphore.h>**, **<signal.h>**, **<sys/types.h>**, the System Interfaces volume of  
11057 IEEE Std 1003.1-200x, *posix\_spawnattr\_destroy()*, *posix\_spawnattr\_getsigdefault()*,  
11058 *posix\_spawnattr\_getflags()*, *posix\_spawnattr\_getpgroup()*, *posix\_spawnattr\_getschedparam()*,  
11059 *posix\_spawnattr\_getschedpolicy()*, *posix\_spawnattr\_getsigmask()*, *posix\_spawnattr\_init()*,  
11060 *posix\_spawnattr\_setsigdefault()*, *posix\_spawnattr\_setflags()*, *posix\_spawnattr\_setpgroup()*,  
11061 *posix\_spawnattr\_setschedparam()*, *posix\_spawnattr\_setschedpolicy()*, *posix\_spawnattr\_setsigmask()*,  
11062 *posix\_spawn()*, *posix\_spawn\_file\_actions\_addclose()*, *posix\_spawn\_file\_actions\_adddup2()*,  
11063 *posix\_spawn\_file\_actions\_addopen()*, *posix\_spawn\_file\_actions\_destroy()*,  
11064 *posix\_spawn\_file\_actions\_init()*, *posix\_spawnnp()*

#### 11065 CHANGE HISTORY

11066 First released in Issue 6. Included for alignment with IEEE Std 1003.1d-1999.

11067 The **restrict** keyword is added to the prototypes for *posix\_spawn()*,  
11068 *posix\_spawn\_file\_actions\_addopen()*, *posix\_spawnattr\_getsigdefault()*, *posix\_spawnattr\_getflags()*,  
11069 *posix\_spawnattr\_getpgroup()*, *posix\_spawnattr\_getschedparam()*, *posix\_spawnattr\_getschedpolicy()*,  
11070 *posix\_spawnattr\_getsigmask()*, *posix\_spawnattr\_setsigdefault()*, *posix\_spawnattr\_setschedparam()*,  
11071 *posix\_spawnattr\_setsigmask()*, and *posix\_spawnnp()*.

11072 **NAME**11073        **stdarg.h** — handle variable argument list11074 **SYNOPSIS**

11075        #include &lt;stdarg.h&gt;

11076        void va\_start(va\_list *ap*, *argN*);11077        void va\_copy(va\_list *dest*, va\_list *src*);11078        type va\_arg(va\_list *ap*, *type*);11079        void va\_end(va\_list *ap*);11080 **DESCRIPTION**

11081 **CX**        The functionality described on this reference page is aligned with the ISO C standard. Any  
 11082        conflict between the requirements described here and the ISO C standard is unintentional. This  
 11083        volume of IEEE Std 1003.1-200x defers to the ISO C standard.

11084        The <stdarg.h> header shall contain a set of macros which allows portable functions that accept  
 11085        variable argument lists to be written. Functions that have variable argument lists (such as  
 11086        *printf()*) but do not use these macros, are inherently non-portable, as different systems use  
 11087        different argument-passing conventions.

11088        The type **va\_list** shall be defined for variables used to traverse the list.

11089        The *va\_start()* macro is invoked to initialize *ap* to the beginning of the list before any calls to  
 11090        *va\_arg()*.

11091        The *va\_copy()* macro initializes as a copy of *src*, as if the *va\_start()* macro had been applied to  
 11092        *dest* followed by the same sequence of uses of the *va\_arg()* macro as had previously been used to  
 11093        reach the present state of *src*. Neither the *va\_copy()* nor *va\_start()* macro shall be invoked to  
 11094        reinitialize *dest* without an intervening invocation of the *va\_end()* macro for the same *dest*.

11095        The object *ap* may be passed as an argument to another function; if that function invokes the  
 11096        *va\_arg()* macro with parameter *ap*, the value of *ap* in the calling function is unspecified and shall  
 11097        be passed to the *va\_end()* macro prior to any further reference to *ap*. The parameter *argN* is the  
 11098        identifier of the rightmost parameter in the variable parameter list in the function definition (the  
 11099        one just before the ...). If the parameter *argN* is declared with the **register** storage class, with a  
 11100        function type or array type, or with a type that is not compatible with the type that results after  
 11101        application of the default argument promotions, the behavior is undefined.

11102        The *va\_arg()* macro shall return the next argument in the list pointed to by *ap*. Each invocation  
 11103        of *va\_arg()* modifies *ap* so that the values of successive arguments are returned in turn. The *type*  
 11104        parameter is the type the argument is expected to be. This is the type name specified such that  
 11105        the type of a pointer to an object that has the specified type can be obtained simply by suffixing  
 11106        a '\*' to *type*. Different types can be mixed, but it is up to the routine to know what type of  
 11107        argument is expected.

11108        The *va\_end()* macro is used to clean up; it invalidates *ap* for use (unless *va\_start()* or *va\_copy()* is  
 11109        invoked again).

11110        Each invocation of the *va\_start()* and *va\_copy()* macros shall be matched by a corresponding  
 11111        invocation of the *va\_end()* macro in the same function.

11112        Multiple traversals, each bracketed by *va\_start()* ... *va\_end()*, are possible.

11113 **EXAMPLES**

11114        This example is a possible implementation of *execl()*:

11115        #include <stdarg.h>

```
11116     #define  MAXARGS      31
11117     /*
11118     * execl is called by
11119     * execl(file, arg1, arg2, ..., (char *) (0));
11120     */
11121     int execl(const char *file, const char *args, ...)
11122     {
11123         va_list ap;
11124         char *array[MAXARGS];
11125         int argno = 0;
11126         va_start(ap, args);
11127         while (args != 0) {
11128             array[argno++] = args;
11129             args = va_arg(ap, const char *);
11130         }
11131         va_end(ap);
11132         return execv(file, array);
11133     }
```

**11134 APPLICATION USAGE**

11135 It is up to the calling routine to communicate to the called routine how many arguments there  
11136 are, since it is not always possible for the called routine to determine this in any other way. For  
11137 example, *execl()* is passed a null pointer to signal the end of the list. The *printf()* function can tell  
11138 how many arguments are there by the *format* argument.

**11139 RATIONALE**

11140 None.

**11141 FUTURE DIRECTIONS**

11142 None.

**11143 SEE ALSO**

11144 The System Interfaces volume of IEEE Std 1003.1-200x, *exec()*, *printf()*

**11145 CHANGE HISTORY**

11146 First released in Issue 4. Derived from the ANSI C standard.

11147 **NAME**

11148           stdbool.h — boolean type and values

11149 **SYNOPSIS**

11150           #include &lt;stdbool.h&gt;

11151 **DESCRIPTION**

11152 cx       The functionality described on this reference page is aligned with the ISO C standard. Any  
11153           conflict between the requirements described here and the ISO C standard is unintentional. This  
11154           volume of IEEE Std 1003.1-200x defers to the ISO C standard.

11155           The &lt;stdbool.h&gt; header shall define the following macros:

11156           bool     Expands to **\_Bool**.

11157           true     Expands to the integer constant 1.

11158           false    Expands to the integer constant 0.

11159           \_\_bool\_true\_false\_are\_defined

11160                    Expands to the integer constant 1.

11161           An application may undefine and then possibly redefine the macros bool, true, and false.

11162 **APPLICATION USAGE**

11163           None.

11164 **RATIONALE**

11165           None.

11166 **FUTURE DIRECTIONS**

11167           The ability to undefine and redefine the macros bool, true, and false is an obsolescent feature  
11168           and may be withdrawn in the future.

11169 **SEE ALSO**

11170           None.

11171 **CHANGE HISTORY**

11172           First released in Issue 6. Included for alignment with the ISO/IEC 9899: 1999 standard.

11173 **NAME**11174       **stddef.h** — standard type definitions11175 **SYNOPSIS**

11176       #include &lt;stddef.h&gt;

11177 **DESCRIPTION**

11178 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
11179 conflict between the requirements described here and the ISO C standard is unintentional. This  
11180 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

11181       The **<stddef.h>** header shall define the following macros:11182       **NULL**       Null pointer constant.11183       **offsetof(*type*, *member-designator*)**

11184               Integer constant expression of type **size\_t**, the value of which is the offset in bytes  
11185 to the structure member (*member-designator*), from the beginning of its structure  
11186 (*type*).

11187       The **<stddef.h>** header shall define the following types:11188       **ptrdiff\_t**   Signed integer type of the result of subtracting two pointers.

11189       **wchar\_t**    Integer type whose range of values can represent distinct wide-character codes for  
11190 all members of the largest character set specified among the locales supported by  
11191 the compilation environment: the null character has the code value 0 and each  
11192 member of the portable character set has a code value equal to its value when used  
11193 as the lone character in an integer character constant.

11194       **size\_t**       Unsigned integer type of the result of the *sizeof* operator.

11195       The implementation shall support one or more programming environments in which the widths  
11196 of **ptrdiff\_t**, **size\_t**, and **wchar\_t** are no greater than the width of type **long**. The names of these  
11197 programming environments can be obtained using the *confstr()* function or the *getconf* utility.

11198 **APPLICATION USAGE**

11199       None.

11200 **RATIONALE**

11201       None.

11202 **FUTURE DIRECTIONS**

11203       None.

11204 **SEE ALSO**

11205       **<wchar.h>**, **<sys/types.h>**, the System Interfaces volume of IEEE Std 1003.1-200x, *confstr()*, the  
11206 Shell and Utilities volume of IEEE Std 1003.1-200x, *getconf*

11207 **CHANGE HISTORY**

11208       First released in Issue 4. Derived from the ANSI C standard.

11209 **NAME**

11210           stdint.h — integer types

11211 **SYNOPSIS**

11212           #include <stdint.h>

11213 **DESCRIPTION**

11214 **CX**       Some of the functionality described on this reference page extends the ISO C standard.  
 11215 Applications shall define the appropriate feature test macro (see the System Interfaces volume of  
 11216 IEEE Std 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of these  
 11217 symbols in this header.

11218           The <stdint.h> header shall declare sets of integer types having specified widths, and shall  
 11219 define corresponding sets of macros. It shall also define macros that specify limits of integer  
 11220 types corresponding to types defined in other standard headers.

11221           **Note:**     The “width” of an integer type is the number of bits used to store its value in a pure binary  
 11222 system; the actual type may use more bits than that (for example, a 28-bit type could be stored  
 11223 in 32 bits of actual storage). An  $N$ -bit signed type has values in the range  $-2^{N-1}$  or  $1-2^{N-1}$  to  
 11224  $2^{N-1}-1$ , while an  $N$ -bit unsigned type has values in the range 0 to  $2^N-1$ .

11225           Types are defined in the following categories:

- 11226           • Integer types having certain exact widths
- 11227           • Integer types having at least certain specified widths
- 11228           • Fastest integer types having at least certain specified widths
- 11229           • Integer types wide enough to hold pointers to objects
- 11230           • Integer types having greatest width

11231           (Some of these types may denote the same type.)

11232           Corresponding macros specify limits of the declared types and construct suitable constants.

11233           For each type described herein that the implementation provides, the <stdint.h> header shall  
 11234 declare that **typedef** name and define the associated macros. Conversely, for each type described  
 11235 herein that the implementation does not provide, the <stdint.h> header shall not declare that  
 11236 **typedef** name, nor shall it define the associated macros. An implementation shall provide those  
 11237 types described as required, but need not provide any of the others (described as optional).

11238           **Integer Types**

11239           When **typedef** names differing only in the absence or presence of the initial  $u$  are defined, they  
 11240 shall denote corresponding signed and unsigned types as described in the ISO/IEC 9899:1999  
 11241 standard, Section 6.2.5; an implementation providing one of these corresponding types shall also  
 11242 provide the other.

11243           In the following descriptions, the symbol  $N$  represents an unsigned decimal integer with no  
 11244 leading zeros (for example, 8 or 24, but not 04 or 048).

- 11245           • Exact-width integer types

11246           The **typedef** name **int $N$ \_t** designates a signed integer type with width  $N$ , no padding bits,  
 11247 and a two’s-complement representation. Thus, **int8\_t** denotes a signed integer type with a  
 11248 width of exactly 8 bits.

11249           The **typedef** name **uint $N$ \_t** designates an unsigned integer type with width  $N$ . Thus,  
 11250 **uint24\_t** denotes an unsigned integer type with a width of exactly 24 bits.

11251 cx      The following types are required:

11252      **int8\_t**

11253      **int16\_t**

11254      **int32\_t**

11255      **uint8\_t**

11256      **uint16\_t**

11257      **uint32\_t**

11258

11259      If an implementation provides integer types with width 64 that meet these requirements,  
11260      then the following types are required:

11261      **int64\_t**

11262      **uint64\_t**

11263 cx      In particular, this will be the case if any of the following are true:

11264      — The implementation supports the `_POSIX_V6_ILP32_OFFBIG` programming  
11265      environment and the application is being built in the `_POSIX_V6_ILP32_OFFBIG`  
11266      programming environment (see the Shell and Utilities volume of IEEE Std 1003.1-200x,  
11267      c99, Programming Environments).

11268      — The implementation supports the `_POSIX_V6_LP64_OFF64` programming environment  
11269      and the application is being built in the `_POSIX_V6_LP64_OFF64` programming  
11270      environment.

11271      — The implementation supports the `_POSIX_V6_LPBIG_OFFBIG` programming  
11272      environment and the application is being built in the `_POSIX_V6_LPBIG_OFFBIG`  
11273      programming environment.

11274      All other types are of this form optional.

11275      • Minimum-width integer types

11276      The **typedef** name **int\_leastN\_t** designates a signed integer type with a width of at least *N*,  
11277      such that no signed integer type with lesser size has at least the specified width. Thus,  
11278      **int\_least32\_t** denotes a signed integer type with a width of at least 32 bits.

11279      The **typedef** name **uint\_leastN\_t** designates an unsigned integer type with a width of at least  
11280      *N*, such that no unsigned integer type with lesser size has at least the specified width. Thus,  
11281      **uint\_least16\_t** denotes an unsigned integer type with a width of at least 16 bits.

11282      The following types are required:

11283      **int\_least8\_t**

11284      **int\_least16\_t**

11285      **int\_least32\_t**

11286      **int\_least64\_t**

11287      **uint\_least8\_t**

11288      **uint\_least16\_t**

11289      **uint\_least32\_t**

11290      **uint\_least64\_t**

11291      All other types of this form are optional.

11292      • Fastest minimum-width integer types

11293      Each of the following types designates an integer type that is usually fastest to operate with  
11294      among all integer types that have at least the specified width.

11295 The designated type is not guaranteed to be fastest for all purposes; if the implementation  
 11296 has no clear grounds for choosing one type over another, it will simply pick some integer  
 11297 type satisfying the signedness and width requirements.

11298 The **typedef** name **int\_fastN\_t** designates the fastest signed integer type with a width of at  
 11299 least *N*. The **typedef** name **uint\_fastN\_t** designates the fastest unsigned integer type with a  
 11300 width of at least *N*.

11301 The following types are required:

11302 **int\_fast8\_t**  
 11303 **int\_fast16\_t**  
 11304 **int\_fast32\_t**  
 11305 **int\_fast64\_t**  
 11306 **uint\_fast8\_t**  
 11307 **uint\_fast16\_t**  
 11308 **uint\_fast32\_t**  
 11309 **uint\_fast64\_t**

11310 All other types of this form are optional.

11311 • Integer types capable of holding object pointers

11312 The following type designates a signed integer type with the property that any valid pointer  
 11313 to **void** can be converted to this type, then converted back to a pointer to **void**, and the result  
 11314 will compare equal to the original pointer:

11315 **intptr\_t**

11316 The following type designates an unsigned integer type with the property that any valid  
 11317 pointer to **void** can be converted to this type, then converted back to a pointer to **void**, and  
 11318 the result will compare equal to the original pointer:

11319 **uintptr\_t**

11320 XSI On XSI-conformant systems, the **intptr\_t** and **uintptr\_t** types are required; otherwise, they are  
 11321 optional.

11322 • Greatest-width integer types

11323 The following type designates a signed integer type capable of representing any value of any  
 11324 signed integer type:

11325 **intmax\_t**

11326 The following type designates an unsigned integer type capable of representing any value of  
 11327 any unsigned integer type:

11328 **uintmax\_t**

11329 These types are required.

11330 **Note:** Applications can test for optional types by using the corresponding limit macro from **Limits of**  
 11331 **Specified-Width Integer Types** (on page 316).

11332       **Limits of Specified-Width Integer Types** |

11333       The following macros specify the minimum and maximum limits of the types declared in the  
 11334       <stdint.h> header. Each macro name corresponds to a similar type name in **Integer Types** (on  
 11335       page 313).

11336       Each instance of any defined macro shall be replaced by a constant expression suitable for use in  
 11337       #if preprocessing directives, and this expression shall have the same type as would an  
 11338       expression that is an object of the corresponding type converted according to the integer  
 11339       promotions. Its implementation-defined value shall be equal to or greater in magnitude  
 11340       (absolute value) than the corresponding value given below, with the same sign, except where  
 11341       stated to be exactly the given value.

11342       • Limits of exact-width integer types

11343       — Minimum values of exact-width signed integer types:

11344       {INTN\_MIN}               Exactly  $-(2^{N-1})$

11345       — Maximum values of exact-width signed integer types:

11346       {INTN\_MAX}               Exactly  $2^{N-1} - 1$

11347       — Maximum values of exact-width unsigned integer types:

11348       {UINTN\_MAX}              Exactly  $2^N - 1$

11349       • Limits of minimum-width integer types

11350       — Minimum values of minimum-width signed integer types:

11351       {INT\_LEASTN\_MIN}        $-(2^{N-1} - 1)$

11352       — Maximum values of minimum-width signed integer types:

11353       {INT\_LEASTN\_MAX}        $2^{N-1} - 1$  |

11354       — Maximum values of minimum-width unsigned integer types:

11355       {UINT\_LEASTN\_MAX}        $2^N - 1$

11356       • Limits of fastest minimum-width integer types

11357       — Minimum values of fastest minimum-width signed integer types:

11358       {INT\_FASTN\_MIN}        $-(2^{N-1} - 1)$

11359       — Maximum values of fastest minimum-width signed integer types:

11360       {INT\_FASTN\_MAX}        $2^{N-1} - 1$

11361       — Maximum values of fastest minimum-width unsigned integer types:

11362       {UINT\_FASTN\_MAX}        $2^N - 1$

11363       • Limits of integer types capable of holding object pointers

11364       — Minimum value of pointer-holding signed integer type:

11365       {INTPTR\_MIN}            $-(2^{15} - 1)$

11366       — Maximum value of pointer-holding signed integer type:

11367       {INTPTR\_MAX}            $2^{15} - 1$

11368       — Maximum value of pointer-holding unsigned integer type:

- 11369            {UINTPTR\_MAX}             $2^{16} - 1$
- 11370            • Limits of greatest-width integer types
- 11371            — Minimum value of greatest-width signed integer type:
- 11372            {INTMAX\_MIN}             $-(2^{63} - 1)$
- 11373            — Maximum value of greatest-width signed integer type:
- 11374            {INTMAX\_MAX}             $2^{63} - 1$
- 11375            — Maximum value of greatest-width unsigned integer type:
- 11376            {UINTMAX\_MAX}             $2^{64} - 1$

11377            **Limits of Other Integer Types**

11378            The following macros specify the minimum and maximum limits of integer types corresponding  
 11379            to types defined in other standard headers.

11380            Each instance of these macros shall be replaced by a constant expression suitable for use in #if  
 11381            preprocessing directives, and this expression shall have the same type as would an expression  
 11382            that is an object of the corresponding type converted according to the integer promotions. Its  
 11383            implementation-defined value shall be equal to or greater in magnitude (absolute value) than  
 11384            the corresponding value given below, with the same sign.

- 11385            • Limits of **ptrdiff\_t**:
- 11386            {PTRDIFF\_MIN}            -65535
- 11387            {PTRDIFF\_MAX}            +65535
- 11388            • Limits of **sig\_atomic\_t**:
- 11389            {SIG\_ATOMIC\_MIN}        See below.
- 11390            {SIG\_ATOMIC\_MAX}        See below.
- 11391            • Limit of **size\_t**:
- 11392            {SIZE\_MAX}                65535
- 11393            • Limits of **wchar\_t**:
- 11394            {WCHAR\_MIN}             See below.
- 11395            {WCHAR\_MAX}             See below.
- 11396            • Limits of **wint\_t**:
- 11397            {WINT\_MIN}                See below.
- 11398            {WINT\_MAX}                See below.

11399            If **sig\_atomic\_t** (see the <signal.h> header) is defined as a signed integer type, the value of  
 11400            {SIG\_ATOMIC\_MIN} shall be no greater than -127 and the value of {SIG\_ATOMIC\_MAX} shall  
 11401            be no less than 127; otherwise, **sig\_atomic\_t** shall be defined as an unsigned integer type, and the  
 11402            value of {SIG\_ATOMIC\_MIN} shall be 0 and the value of {SIG\_ATOMIC\_MAX} shall be no less  
 11403            than 255.

11404            If **wchar\_t** (see the <stddef.h> header) is defined as a signed integer type, the value of  
 11405            {WCHAR\_MIN} shall be no greater than -127 and the value of {WCHAR\_MAX} shall be no less  
 11406            than 127; otherwise, **wchar\_t** shall be defined as an unsigned integer type, and the value of  
 11407            {WCHAR\_MIN} shall be 0 and the value of {WCHAR\_MAX} shall be no less than 255.

11408 If **wint\_t** (see the **<wchar.h>** header) is defined as a signed integer type, the value of  
11409 {WINT\_MIN} shall be no greater than  $-32767$  and the value of {WINT\_MAX} shall be no less  
11410 than  $32767$ ; otherwise, **wint\_t** shall be defined as an unsigned integer type, and the value of  
11411 {WINT\_MIN} shall be 0 and the value of {WINT\_MAX} shall be no less than 65535.

#### 11412 **Macros for Integer Constant Expressions**

11413 The following macros expand to integer constant expressions suitable for initializing objects that  
11414 have integer types corresponding to types defined in the **<stdint.h>** header. Each macro name  
11415 corresponds to a similar type name listed under *Minimum-width integer types* and *Greatest-width*  
11416 *integer types*.

11417 Each invocation of one of these macros shall expand to an integer constant expression suitable  
11418 for use in **#if** preprocessing directives. The type of the expression shall have the same type as  
11419 would an expression that is an object of the corresponding type converted according to the  
11420 integer promotions. The value of the expression shall be that of the argument.

11421 The argument in any instance of these macros shall be a decimal, octal, or hexadecimal constant  
11422 with a value that does not exceed the limits for the corresponding type.

- 11423 • **Macros for minimum-width integer constant expressions**

11424 The macro *INTN\_C(value)* shall expand to an integer constant expression corresponding to  
11425 the type **int\_leastN\_t**. The macro *UINTN\_C(value)* shall expand to an integer constant  
11426 expression corresponding to the type **uint\_leastN\_t**. For example, if **uint\_least64\_t** is a name  
11427 for the type **unsigned long long**, then *UINT64\_C(0x123)* might expand to the integer  
11428 constant `0x123ULL`.

- 11429 • **Macros for greatest-width integer constant expressions**

11430 The following macro expands to an integer constant expression having the value specified by  
11431 its argument and the type **intmax\_t**:

11432 *INTMAX\_C(value)*

11433 The following macro expands to an integer constant expression having the value specified by  
11434 its argument and the type **uintmax\_t**:

11435 *UINTMAX\_C(value)*

#### 11436 **APPLICATION USAGE**

11437 None.

#### 11438 **RATIONALE**

11439 The **<stdint.h>** header is a subset of the **<inttypes.h>** header more suitable for use in  
11440 freestanding environments, which might not support the formatted I/O functions. In some  
11441 environments, if the formatted conversion support is not wanted, using this header instead of  
11442 the **<inttypes.h>** header avoids defining such a large number of macros.

11443 As a consequence of adding **int8\_t** the following are true:

- 11444 • A byte is exactly 8 bits.

- 11445 • {CHAR\_BIT} has the value 8, {SCHAR\_MAX} has the value 127, {SCHAR\_MIN} has the  
11446 value  $-127$  or  $-128$ , and {UCHAR\_MAX} has the value 255.

#### 11447 **FUTURE DIRECTIONS**

11448 **typedef** names beginning with **int** or **uint** and ending with **\_t** may be added to the types defined  
11449 in the **<stdint.h>** header. Macro names beginning with **INT** or **UINT** and ending with **\_MAX**,  
11450 **\_MIN**, or **\_C** may be added to the macros defined in the **<stdint.h>** header.

11451 **SEE ALSO**

11452           <signal.h>, <stddef.h>, <wchar.h>, <inttypes.h>

11453 **CHANGE HISTORY**

11454           First released in Issue 6. Included for alignment with the ISO/IEC 9899: 1999 standard. |

11455           ISO/IEC 9899: 1999 standard, Technical Corrigendum No. 1 is incorporated. |

11456 **NAME**11457         **stdio.h** — standard buffered input/output11458 **SYNOPSIS**

11459         #include &lt;stdio.h&gt;

11460 **DESCRIPTION**

11461 **CX**         Some of the functionality described on this reference page extends the ISO C standard.  
 11462         Applications shall define the appropriate feature test macro (see the System Interfaces volume of  
 11463         IEEE Std 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of these  
 11464         symbols in this header.

11465         The <**stdio.h**> header shall define the following macros as positive integer constant expressions:

11466         **BUFSIZ**                 Size of <**stdio.h**> buffers.

11467         **\_IOFBF**                 Input/output fully buffered.

11468         **\_IOLBF**                 Input/output line buffered.

11469         **\_IONBF**                 Input/output unbuffered.

11470 **CX**         **L\_ctermid**             Maximum size of character array to hold *ctermid()* output.

11471         **L\_tmpnam**             Maximum size of character array to hold *tmpnam()* output.

11472         **SEEK\_CUR**             Seek relative to current position.

11473         **SEEK\_END**             Seek relative to end-of-file.

11474         **SEEK\_SET**             Seek relative to start-of-file.

11475         The following macros shall be defined as positive integer constant expressions which denote  
 11476         implementation limits:

11477         {**FILENAME\_MAX**}         Maximum size in bytes of the longest filename string that the  
 11478         implementation guarantees can be opened.

11479         {**FOPEN\_MAX**}         Number of streams which the implementation guarantees can be open  
 11480         simultaneously. The value is at least eight.

11481         {**TMP\_MAX**}             Minimum number of unique filenames generated by *tmpnam()*.  
 11482         Maximum number of times an application can call *tmpnam()* reliably. The  
 11483 **XSI**         value of {**TMP\_MAX**} is at least 25. On XSI-conformant systems, the  
 11484         value of {**TMP\_MAX**} is at least 10,000.

11485         The following macro name shall be defined as a negative integer constant expression:

11486         **EOF**                     End-of-file return value.

11487         The following macro name shall be defined as a null pointer constant:

11488         **NULL**                    Null pointer.

11489         The following macro name shall be defined as a string constant:

11490 **XSI**         **P\_tmpdir**             Default directory prefix for *tmpnam()*.

11491         The following shall be defined as expressions of type “pointer to **FILE**” that point to the **FILE**  
 11492         objects associated, respectively, with the standard error, input, and output streams:

11493         **stderr**                 Standard error output stream.

11494         **stdin**                    Standard input stream.



```

11544 int      putc_unlocked(int, FILE *);
11545 int      putchar_unlocked(int);
11546 int      puts(const char *);
11547 int      remove(const char *);
11548 int      rename(const char *, const char *);
11549 void     rewind(FILE *);
11550 int      scanf(const char *restrict, ...);
11551 void     setbuf(FILE *restrict, char *restrict);
11552 int      setvbuf(FILE *restrict, char *restrict, int, size_t);
11553 int      snprintf(char *restrict, size_t, const char *restrict, ...);
11554 int      sprintf(char *restrict, const char *restrict, ...);
11555 int      sscanf(const char *restrict, const char *restrict, int ...);
11556 XSI     char *tempnam(const char *, const char *);
11557 FILE     *tmpfile(void);
11558 char     *tmpnam(char *);
11559 int      ungetc(int, FILE *);
11560 int      vfprintf(FILE *restrict, const char *restrict, va_list);
11561 int      vfscanf(FILE *restrict, const char *restrict, va_list);
11562 int      vprintf(const char *restrict, va_list);
11563 int      vscanf(const char *restrict, va_list);
11564 int      vsnprintf(char *restrict, size_t, const char *restrict, va_list);
11565 int      vsprintf(char *restrict, const char *restrict, va_list);
11566 int      vsscanf(const char *restrict, const char *restrict, va_list arg);

```

11567 XSI **Inclusion of the <stdio.h> header may also make visible all symbols from <stddef.h>.**

#### 11568 APPLICATION USAGE

11569 None.

#### 11570 RATIONALE

11571 None.

#### 11572 FUTURE DIRECTIONS

11573 None.

#### 11574 SEE ALSO

11575 <sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-200x, *clearerr()*, *ctermid()*,  
11576 *fclose()*, *fdopen()*, *fgetc()*, *fgetpos()*, *ferror()*, *feof()*, *fflush()*, *fgets()*, *fileno()*, *flockfile()*, *fopen()*,  
11577 *fputc()*, *fputs()*, *fread()*, *freopen()*, *fseek()*, *fsetpos()*, *ftell()*, *fwrite()*, *getc()*, *getc\_unlocked()*,  
11578 *getwchar()*, *getchar()*, *getopt()*, *gets()*, *pclose()*, *perror()*, *popen()*, *printf()*, *putc()*, *putchar()*, *puts()*,  
11579 *putwchar()*, *remove()*, *rename()*, *rewind()*, *scanf()*, *setbuf()*, *setvbuf()*, *sscanf()*, *stdin*, *system()*,  
11580 *tempnam()*, *tmpfile()*, *tmpnam()*, *ungetc()*, *vfscanf()*, *vscanf()*, *vprintf()*, *vsscanf()*

#### 11581 CHANGE HISTORY

11582 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 11583 Issue 5

11584 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

11585 Large File System extensions are added.

11586 The constant *L\_cuserid* and the external variables *optarg*, *opterr*, *optind*, and *optopt* are marked as  
11587 extensions and LEGACY.

11588 The *cuserid()* and *getopt()* functions are marked LEGACY.

11589 **Issue 6**

11590 The constant `L_cuserid` and the external variables `optarg`, `opterr`, `optind`, and `optopt` are removed  
11591 as they were previously marked LEGACY.

11592 The `cuserid()`, `getopt()`, and `getw()` functions are removed as they were previously marked |  
11593 LEGACY.

11594 Several functions are marked as part of the `_POSIX_THREAD_SAFE_FUNCTIONS` option.

11595 This reference page is updated to align with the ISO/IEC 9899:1999 standard. Note that the  
11596 description of the `fpos_t` type is now explicitly updated to exclude array types.

11597 Extensions beyond the ISO C standard are now marked. |

11598 NAME

11599         stdlib.h — standard library definitions

11600 SYNOPSIS

11601         #include <stdlib.h>

11602 DESCRIPTION

11603 cx       Some of the functionality described on this reference page extends the ISO C standard.  
11604       Applications shall define the appropriate feature test macro (see the System Interfaces volume of  
11605       IEEE Std 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of these  
11606       symbols in this header.

11607       The <stdlib.h> header shall define the following macros:

11608       EXIT\_FAILURE   Unsuccessful termination for *exit()*; evaluates to a non-zero value.

11609       EXIT\_SUCCESS   Successful termination for *exit()*; evaluates to 0.

11610       NULL           Null pointer.

11611       {RAND\_MAX}     Maximum value returned by *rand()*; at least 32,767.

11612       {MB\_CUR\_MAX}   Integer expression whose value is the maximum number of bytes in a  
11613       character specified by the current locale.

11614       The following data types shall be defined through **typedef**:

11615       **div\_t**           Structure type returned by the *div()* function.

11616       **ldiv\_t**          Structure type returned by the *ldiv()* function.

11617       **lldiv\_t**         Structure type returned by the *lldiv()* function.

11618       **size\_t**          As described in <stddef.h>.

11619       **wchar\_t**         As described in <stddef.h>.

11620       In addition, the following symbolic names and macros shall be defined as in <sys/wait.h>, for  
11621       use in decoding the return value from *system()*:

11622 xsi      WNOHANG

11623            WUNTRACED

11624            WEXITSTATUS

11625            WIFEXITED

11626            WIFSIGNALED

11627            WIFSTOPPED

11628            WSTOPSIG

11629            WTERMSIG

11630

11631       The following shall be declared as functions and may also be defined as macros. Function |  
11632       prototypes shall be provided. |

11633       void            \_Exit(int);

11634 xsi      long         a64l(const char \*);

11635       void            abort(void);

11636       int             abs(int);

11637       int             atexit(void (\*)(void));

11638       double          atof(const char \*);

11639       int             atoi(const char \*);

11640       long            atol(const char \*);

```

11641     long long    atoll(const char *);
11642     void        *bsearch(const void *, const void *, size_t, size_t,
11643                     int (*)(const void *, const void *));
11644     void        *calloc(size_t, size_t);
11645     div_t       div(int, int);
11646 XSI     double    drand48(void);
11647     char        *ecvt(double, int, int *restrict, int *restrict); (LEGACY)
11648     double     erand48(unsigned short[3]);
11649     void        exit(int);
11650 XSI     char        *fcvt(double, int, int *restrict, int *restrict); (LEGACY)
11651     void        free(void *);
11652 XSI     char        *gcvt(double, int, char *); (LEGACY)
11653     char        *getenv(const char *);
11654 XSI     int        getsubopt(char **, char *const *, char **);
11655     int        grantpt(int);
11656     char        *initstate(unsigned, char *, size_t);
11657     long        jrand48(unsigned short[3]);
11658     char        *l64a(long);
11659     long        labs(long);
11660 XSI     void        lcong48(unsigned short[7]);
11661     ldiv_t      ldiv(long, long);
11662     long long   llabs(long long);
11663     lldiv_t     lldiv(long long, long long);
11664 XSI     long        lrand48(void);
11665     void        *malloc(size_t);
11666     int        mblen(const char *, size_t);
11667     size_t     mbstowcs(wchar_t *restrict, const char *restrict, size_t);
11668     int        mbtowc(wchar_t *restrict, const char *restrict, size_t);
11669 XSI     char        *mktemp(char *); (LEGACY)
11670     int        mkstemp(char *);
11671     long        mrand48(void);
11672     long        nrand48(unsigned short[3]);
11673 ADV     int        posix_memalign(void **, size_t, size_t);
11674 XSI     int        posix_openpt(int);
11675     char        *ptsname(int);
11676     int        putenv(char *);
11677     void        qsort(void *, size_t, size_t, int (*)(const void *,
11678                     const void *));
11679     int        rand(void);
11680 TSF     int        rand_r(unsigned *);
11681 XSI     long        random(void);
11682     void        *realloc(void *, size_t);
11683 XSI     char        *realpath(const char *restrict, char *restrict);
11684     unsigned short seed48(unsigned short[3]);
11685 CX     int        setenv(const char *, const char *, int);
11686 XSI     void        setkey(const char *);
11687     char        *setstate(const char *);
11688     void        srand(unsigned);
11689 XSI     void        srand48(long);
11690     void        srandom(unsigned);
11691     double     strtod(const char *restrict, char **restrict);
11692     float      strtod(const char *restrict, char **restrict);

```

```

11693     long          strtol(const char *restrict, char **restrict, int);
11694     long double    strtold(const char *restrict, char **restrict);
11695     long long      strtoll(const char *restrict, char **restrict, int);
11696     unsigned long  strtoul(const char *restrict, char **restrict, int);
11697     long long      strtoull(const char *restrict, char **restrict, int);
11698     int            system(const char *);
11699 XSI     int          unlockpt(int);
11700 CX     int          unsetenv(const char *);
11701     size_t         wcstombs(char *restrict, const wchar_t *restrict, size_t);
11702     int            wctomb(char *, wchar_t);

11703 XSI     Inclusion of the <stdlib.h> header may also make visible all symbols from <stddef.h>,
11704         <limits.h>, <math.h>, and <sys/wait.h>.

```

**11705 APPLICATION USAGE**

11706 None.

**11707 RATIONALE**

11708 None.

**11709 FUTURE DIRECTIONS**

11710 None.

**11711 SEE ALSO**

11712 <sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-200x, *\_Exit()*, *a64l()*, *abort()*,  
11713 *abs()*, *atexit()*, *atof()*, *atoi()*, *atol()*, *atoll()*, *bsearch()*, *calloc()*, *div()*, *drand48()*, *erand48()*, *exit()*,  
11714 *free()*, *getenv()*, *getsubopt()*, *grantpt()*, *initstate()*, *jrand48()*, *l64a()*, *labs()*, *lcong48()*, *ldiv()*, *llabs()*,  
11715 *lldiv()*, *lrand48()*, *malloc()*, *mblen()*, *mbstowcs()*, *mbtowc()*, *mkstemp()*, *mrand48()*, *nrand48()*,  
11716 *posix\_memalign()*, *ptsname()*, *putenv()*, *qsort()*, *rand()*, *realloc()*, *realpath()*, *setstate()*, *srand()*,  
11717 *srand48()*, *srandom()*, *strtod()*, *strtof()*, *strtol()*, *strtold()*, *strtoll()*, *strtoul()*, *strtoull()*, *unlockpt()*,  
11718 *wcstombs()*, *wctomb()*

**11719 CHANGE HISTORY**

11720 First released in Issue 3.

**11721 Issue 5**

11722 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

11723 The *ttyslot()* and *valloc()* functions are marked LEGACY.

11724 The type of the third argument to *initstate()* is changed from **int** to **size\_t**. The type of the return  
11725 value from *setstate()* is changed from **char** to **char \***, and the type of the first argument is  
11726 changed from **char \*** to **const char \***.

**11727 Issue 6**

11728 The Open Group Corrigendum U021/1 is applied, correcting the prototype for *realpath()* to be  
11729 consistent with the reference page.

11730 The Open Group Corrigendum U028/13 is applied, correcting the prototype for *putenv()* to be  
11731 consistent with the reference page.

11732 The *rand\_r()* function is marked as part of the `_POSIX_THREAD_SAFE_FUNCTIONS` option.

11733 Function prototypes for *setenv()* and *unsetenv()* are added.

11734 The *posix\_memalign()* function is added for alignment with IEEE Std 1003.1d-1999.

11735 This reference page is updated to align with the ISO/IEC 9899:1999 standard.

- 11736 The *ecvt()*, *fcvt()*, *gcvt()*, and *mktemp()* functions are marked LEGACY.
- 11737 The *ttyslot()* and *valloc()* functions are removed as they were previously marked LEGACY. |
- 11738 Extensions beyond the ISO C standard are now marked. |

11739 **NAME**11740 `string.h` — string operations11741 **SYNOPSIS**11742 `#include <string.h>`11743 **DESCRIPTION**

11744 CX Some of the functionality described on this reference page extends the ISO C standard.  
 11745 Applications shall define the appropriate feature test macro (see the System Interfaces volume of  
 11746 IEEE Std 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of these  
 11747 symbols in this header.

11748 The **<string.h>** header shall define the following:11749 **NULL** Null pointer constant.11750 **size\_t** As described in **<stddef.h>**.

11751 The following shall be declared as functions and may also be defined as macros. Function |  
 11752 prototypes shall be provided. |

11753 XSI `void *memcpy(void *restrict, const void *restrict, int, size_t);`11754 `void *memchr(const void *, int, size_t);`11755 `int memcmp(const void *, const void *, size_t);`11756 `void *memcpy(void *restrict, const void *restrict, size_t);`11757 `void *memmove(void *, const void *, size_t);`11758 `void *memset(void *, int, size_t);`11759 `char *strcat(char *restrict, const char *restrict);`11760 `char *strchr(const char *, int);`11761 `int strcmp(const char *, const char *);`11762 `int strcoll(const char *, const char *);`11763 `char *strcpy(char *restrict, const char *restrict);`11764 `size_t strcspn(const char *, const char *);`11765 XSI `char *strdup(const char *);`11766 `char *strerror(int);`11767 `size_t strlen(const char *);`11768 `char *strncat(char *restrict, const char *restrict, size_t);`11769 `int strncmp(const char *, const char *, size_t);`11770 `char *strncpy(char *restrict, const char *restrict, size_t);`11771 `char *strpbrk(const char *, const char *);`11772 `char *strrchr(const char *, int);`11773 `size_t strspn(const char *, const char *);`11774 `char *strstr(const char *, const char *);`11775 `char *strtok(char *restrict, const char *restrict);`11776 TSF `char *strtok_r(char *, const char *, char **);`11777 `size_t strxfrm(char *restrict, const char *restrict, size_t);`11778 XSI Inclusion of the **<string.h>** header may also make visible all symbols from **<stddef.h>**.

11779 **APPLICATION USAGE**

11780 None.

11781 **RATIONALE**

11782 None.

11783 **FUTURE DIRECTIONS**

11784 None.

11785 **SEE ALSO**

11786 <sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-200x, *memcpy()*, *memchr()*,  
11787 *memcmp()*, *memcpy()*, *memmove()*, *memset()*, *strcat()*, *strchr()*, *strcmp()*, *strcoll()*, *strcpy()*,  
11788 *strcspn()*, *strdup()*, *strerror()*, *strlen()*, *strncat()*, *strncmp()*, *strncpy()*, *strpbrk()*, *strrchr()*, *strspn()*,  
11789 *strstr()*, *strtok()*, *strxfrm()*

11790 **CHANGE HISTORY**

11791 First released in Issue 1. Derived from Issue 1 of the SVID.

11792 **Issue 5**

11793 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

11794 **Issue 6**11795 The *strtok\_r()* function is marked as part of the `_POSIX_THREAD_SAFE_FUNCTIONS` option.

11796 This reference page is updated to align with the ISO/IEC 9899:1999 standard.

11797 **NAME**

11798 strings.h — string operations

11799 **SYNOPSIS**

11800 XSI #include &lt;strings.h&gt;

11801

11802 **DESCRIPTION**11803 The following shall be declared as functions and may also be defined as macros. Function |  
11804 prototypes shall be provided. |11805 int bcmp(const void \*, const void \*, size\_t); (**LEGACY**)11806 void bcopy(const void \*, void \*, size\_t); (**LEGACY**)11807 void bzero(void \*, size\_t); (**LEGACY**)

11808 int ffs(int);

11809 char \*index(const char \*, int); (**LEGACY**)11810 char \*rindex(const char \*, int); (**LEGACY**)

11811 int strcasecmp(const char \*, const char \*);

11812 int strncasecmp(const char \*, const char \*, size\_t);

11813 The `size_t` type shall be defined through `typedef` as described in `<stddef.h>`.11814 **APPLICATION USAGE**

11815 None.

11816 **RATIONALE**

11817 None.

11818 **FUTURE DIRECTIONS**

11819 None.

11820 **SEE ALSO**11821 `<stddef.h>`, the System Interfaces volume of IEEE Std 1003.1-200x, `ffs()`, `strcasecmp()`,  
11822 `strncasecmp()`11823 **CHANGE HISTORY**

11824 First released in Issue 4, Version 2.

11825 **Issue 6**11826 The Open Group Corrigendum U021/2 is applied, correcting the prototype for `index()` to be  
11827 consistent with the reference page.11828 The `bcmp()`, `bcopy()`, `bzero()`, `index()`, and `rindex()` functions are marked LEGACY.

11829 **NAME**

11830 `stropts.h` — STREAMS interface (**STREAMS**)

11831 **SYNOPSIS**

11832 XSR `#include <stropts.h>`

11833

11834 **DESCRIPTION**

11835 The `<stropts.h>` header shall define the **bandinfo** structure that includes at least the following  
 11836 members:

11837	<code>unsigned char</code>	<code>bi_pri</code>	Priority band.	
11838	<code>int</code>	<code>bi_flag</code>	Flushing type.	

11839 The `<stropts.h>` header shall define the **strpeek** structure that includes at least the following  
 11840 members:

11841	<code>struct strbuf</code>	<code>ctlbuf</code>	The control portion of the message.	
11842	<code>struct strbuf</code>	<code>databuf</code>	The data portion of the message.	
11843	<code>t_uscalar_t</code>	<code>flags</code>	<code>RS_HIPRI</code> or 0.	

11844 The `<stropts.h>` header shall define the **strbuf** structure that includes at least the following  
 11845 members:

11846	<code>int</code>	<code>maxlen</code>	Maximum buffer length.	
11847	<code>int</code>	<code>len</code>	Length of data.	
11848	<code>char</code>	<code>*buf</code>	Pointer to buffer.	

11849 The `<stropts.h>` header shall define the **strfdinsert** structure that includes at least the following  
 11850 members:

11851	<code>struct strbuf</code>	<code>ctlbuf</code>	The control portion of the message.	
11852	<code>struct strbuf</code>	<code>databuf</code>	The data portion of the message.	
11853	<code>t_uscalar_t</code>	<code>flags</code>	<code>RS_HIPRI</code> or 0.	
11854	<code>int</code>	<code>fildes</code>	File descriptor of the other STREAM.	
11855	<code>int</code>	<code>offset</code>	Relative location of the stored value.	

11856 The `<stropts.h>` header shall define the **striocctl** structure that includes at least the following  
 11857 members:

11858	<code>int</code>	<code>ic_cmd</code>	<code>ioctl()</code> command.	
11859	<code>int</code>	<code>ic_timeout</code>	Timeout for response.	
11860	<code>int</code>	<code>ic_len</code>	Length of data.	
11861	<code>char</code>	<code>*ic_dp</code>	Pointer to buffer.	

11862 The `<stropts.h>` header shall define the **strrecvfd** structure that includes at least the following  
 11863 members:

11864	<code>int</code>	<code>fda</code>	Received file descriptor.	
11865	<code>uid_t</code>	<code>uid</code>	UID of sender.	
11866	<code>gid_t</code>	<code>gid</code>	GID of sender.	

11867 The **uid\_t** and **gid\_t** types shall be defined through **typedef** as described in `<sys/types.h>`.

11868 The `<stropts.h>` header shall define the **t\_scalar\_t** and **t\_uscalar\_t** types respectively as signed  
 11869 and unsigned opaque types of equal length of at least 32 bits.

11870 The `<stropts.h>` header shall define the **str\_list** structure that includes at least the following  
 11871 members:

11872	int	sl_nmods	Number of STREAMS module names.		
11873	struct str_mlist	*sl_modlist	STREAMS module names.		
11874	The <b>&lt;stropts.h&gt;</b> header shall define the <b>str_mlist</b> structure that includes at least the following				
11875	member:				
11876	char	l_name[FMNAMESZ+1]	A STREAMS module name.		
11877	At least the following macros shall be defined for use as the <i>request</i> argument to <i>ioctl()</i> :				
11878	I_PUSH		Push a STREAMS module.		
11879	I_POP		Pop a STREAMS module.		
11880	I_LOOK		Get the top module name.		
11881	I_FLUSH		Flush a STREAM.		
11882	I_FLUSHBAND		Flush one band of a STREAM.		
11883	I_SETSIG		Ask for notification signals.		
11884	I_GETSIG		Retrieve current notification signals.		
11885	I_FIND		Look for a STREAMS module.		
11886	I_PEEK		Peek at the top message on a STREAM.		
11887	I_SRDOPT		Set the read mode.		
11888	I_GRDOPT		Get the read mode.		
11889	I_NREAD		Size the top message.		
11890	I_FDINSERT		Send implementation-defined information about another STREAM.		
11891	I_STR		Send a STREAMS <i>ioctl()</i> .		
11892	I_SWROPT		Set the write mode.		
11893	I_GWROPT		Get the write mode.		
11894	I_SENDFD		Pass a file descriptor through a STREAMS pipe.		
11895	I_RECVFD		Get a file descriptor sent via I_SENDFD.		
11896	I_LIST		Get all the module names on a STREAM.		
11897	I_ATMARK		Is the top message “marked”?		
11898	I_CKBAND		See if any messages exist in a band.		
11899	I_GETBAND		Get the band of the top message on a STREAM.		
11900	I_CANPUT		Is a band writable?		
11901	I_SETCLTIME		Set close time delay.		
11902	I_GETCLTIME		Get close time delay.		
11903	I_LINK		Connect two STREAMS.		
11904	I_UNLINK		Disconnect two STREAMS.		
11905	I_PLINK		Persistently connect two STREAMS.		
11906	I_PUNLINK		Dismantle a persistent STREAMS link.		

11907		At least the following macros shall be defined for use with I_LOOK:	
11908	FMNAMESZ	The minimum size in bytes of the buffer referred to by the <i>arg</i> argument.	
11909		At least the following macros shall be defined for use with I_FLUSH:	
11910	FLUSHR	Flush read queues.	
11911	FLUSHW	Flush write queues.	
11912	FLUSHRW	Flush read and write queues.	
11913		At least the following macros shall be defined for use with I_SETSIG:	
11914	S_RDNORM	A normal (priority band set to 0) message has arrived at the head of a	
11915		STREAM head read queue.	
11916	S_RDBAND	A message with a non-zero priority band has arrived at the head of a STREAM	
11917		head read queue.	
11918	S_INPUT	A message, other than a high-priority message, has arrived at the head of a	
11919		STREAM head read queue.	
11920	S_HIPRI	A high-priority message is present on a STREAM head read queue.	
11921	S_OUTPUT	The write queue for normal data (priority band 0) just below the STREAM	
11922		head is no longer full. This notifies the process that there is room on the queue	
11923		for sending (or writing) normal data downstream.	
11924	S_WRNORM	Equivalent to S_OUTPUT.	
11925	S_WRBAND	The write queue for a non-zero priority band just below the STREAM head is	
11926		no longer full.	
11927	S_MSG	A STREAMS signal message that contains the SIGPOLL signal reaches the	
11928		front of the STREAM head read queue.	
11929	S_ERROR	Notification of an error condition reaches the STREAM head.	
11930	S_HANGUP	Notification of a hangup reaches the STREAM head.	
11931	S_BANDURG	When used in conjunction with S_RDBAND, SIGURG is generated instead of	
11932		SIGPOLL when a priority message reaches the front of the STREAM head read	
11933		queue.	
11934		At least the following macros shall be defined for use with I_PEEK:	
11935	RS_HIPRI	Only look for high-priority messages.	
11936		At least the following macros shall be defined for use with I_SRDOPT:	
11937	RNORM	Byte-STREAM mode, the default.	
11938	RMSGD	Message-discard mode.	
11939	RMSGN	Message-nondiscard mode.	
11940	RPROTNORM	Fail <i>read()</i> with [EBADMSG] if a message containing a control part is at the	
11941		front of the STREAM head read queue.	
11942	RPROTDAT	Deliver the control part of a message as data when a process issues a <i>read()</i> .	
11943	RPROTDIS	Discard the control part of a message, delivering any data part, when a	
11944		process issues a <i>read()</i> .	

11945 At least the following macros shall be defined for use with I\_SWOPT: |

11946 SNDZERO Send a zero-length message downstream when a *write()* of 0 bytes occurs.

11947 At least the following macros shall be defined for use with I\_ATMARK: |

11948 ANYMARK Check if the message is marked.

11949 LASTMARK Check if the message is the last one marked on the queue.

11950 At least the following macros shall be defined for use with I\_UNLINK: |

11951 MUXID\_ALL Unlink all STREAMs linked to the STREAM associated with *fildev*.

11952 The following macros shall be defined for *getmsg()*, *getpmsg()*, *putmsg()*, and *putpmsg()*: |

11953 MSG\_ANY Receive any message.

11954 MSG\_BAND Receive message from specified band.

11955 MSG\_HIPRI Send/receive high-priority message.

11956 MORECTL More control information is left in message.

11957 MOREDATA More data is left in message.

11958 The **<stropts.h>** header may make visible all of the symbols from **<unistd.h>**.

11959 The following shall be declared as functions and may also be defined as macros. Function |

11960 prototypes shall be provided. |

```

11961 int isastream(int);
11962 int getmsg(int, struct strbuf *restrict, struct strbuf *restrict,
11963          int *restrict);
11964 int getpmsg(int, struct strbuf *restrict, struct strbuf *restrict,
11965           int *restrict, int *restrict);
11966 int ioctl(int, int, ... );
11967 int putmsg(int, const struct strbuf *, const struct strbuf *, int);
11968 int putpmsg(int, const struct strbuf *, const struct strbuf *, int,
11969           int);
11970 int fattach(int, const char *);
11971 int fdetach(const char *);

```

11972 **APPLICATION USAGE**

11973 None.

11974 **RATIONALE**

11975 None.

11976 **FUTURE DIRECTIONS**

11977 None.

11978 **SEE ALSO**

11979 **<sys/types.h>**, the System Interfaces volume of IEEE Std 1003.1-200x, *close()*, *fcntl()*, *getmsg()*,

11980 *ioctl()*, *open()*, *pipe()*, *read()*, *poll()*, *putmsg()*, *signal()*, *write()* |

11981 **CHANGE HISTORY**

11982 First released in Issue 4, Version 2.

11983 **Issue 5**

11984       The *flags* member of the **strpeek** and **strfdinsert** structures are changed from type **long** to  
11985       **t\_uscalar\_t**.

11986 **Issue 6**

11987       This header is marked as part of the XSI STREAMS Option Group.

11988       The **restrict** keyword is added to the prototypes for *getmsg()* and *getpmsg()*.

11989 **NAME**

11990 sys/ipc.h — XSI interprocess communication access structure

11991 **SYNOPSIS**

11992 xsi #include &lt;sys/ipc.h&gt;

11993

11994 **DESCRIPTION**

11995 The **<sys/ipc.h>** header is used by three mechanisms for XSI interprocess communication (IPC):  
 11996 messages, semaphores, and shared memory. All use a common structure type, **ipc\_perm** to pass  
 11997 information used in determining permission to perform an IPC operation.

11998 The **ipc\_perm** structure shall contain the following members:

11999	uid_t	uid	Owner's user ID.
12000	gid_t	gid	Owner's group ID.
12001	uid_t	cuid	Creator's user ID.
12002	gid_t	cgid	Creator's group ID.
12003	mode_t	mode	Read/write permission.

12004 The **uid\_t**, **gid\_t**, **mode\_t**, and **key\_t** types shall be defined as described in **<sys/types.h>**.

12005 Definitions shall be provided for the following constants:

12006 Mode bits:

12007	IPC_CREAT	Create entry if key does not exist.
12008	IPC_EXCL	Fail if key exists.
12009	IPC_NOWAIT	Error if request must wait.

12010 Keys:

12011	IPC_PRIVATE	Private key.
-------	-------------	--------------

12012 Control commands:

12013	IPC_RMID	Remove identifier.
12014	IPC_SET	Set options.
12015	IPC_STAT	Get options.

12016 The following shall be declared as a function and may also be defined as a macro. A function |  
 12017 prototype shall be provided. |

12018 key\_t ftok(const char \*, int);

12019 **APPLICATION USAGE**

12020 None.

12021 **RATIONALE**

12022 None.

12023 **FUTURE DIRECTIONS**

12024 None.

12025 **SEE ALSO**

12026 **<sys/types.h>**, the System Interfaces volume of IEEE Std 1003.1-200x, *ftok()*

12027 **CHANGE HISTORY**

12028 First released in Issue 2. Derived from System V Release 2.0.

12029 **NAME**12030 `sys/mman.h` — memory management declarations12031 **SYNOPSIS**12032 `#include <sys/mman.h>`12033 **DESCRIPTION**12034 The **<sys/mman.h>** header shall be supported if the implementation supports at least one of the  
12035 following options:

- 12036 MF • The Memory Mapped Files option
- 12037 SHM • The Shared Memory Objects option
- 12038 ML • The Process Memory Locking option
- 12039 MPR • The Memory Protection option
- 12040 TYM • The Typed Memory Objects option
- 12041 SIO • The Synchronized Input and Output option
- 12042 ADV • The Advisory Information option
- 12043 TYM • The Typed Memory Objects option

12044 MC2 If one or more of the Advisory Information, Memory Mapped Files, or Shared Memory Objects  
12045 options are supported, the following protection options shall be defined:

- 12046 MC2 `PROT_READ` Page can be read.
- 12047 MC2 `PROT_WRITE` Page can be written.
- 12048 MC2 `PROT_EXEC` Page can be executed.
- 12049 MC2 `PROT_NONE` Page cannot be accessed.

12050 The following *flag* options shall be defined:

- 12051 MF|SHM `MAP_SHARED` Share changes.
- 12052 MF|SHM `MAP_PRIVATE` Changes are private.
- 12053 MF|SHM `MAP_FIXED` Interpret *addr* exactly.

12054 The following flags shall be defined for *msync()*:

- 12055 MF|SIO `MS_ASYNC` Perform asynchronous writes.
- 12056 MF|SIO `MS_SYNC` Perform synchronous writes.
- 12057 MF|SIO `MS_INVALIDATE` Invalidate mappings.

12058 ML The following symbolic constants shall be defined for the *mlockall()* function:

- 12059 ML `MCL_CURRENT` Lock currently mapped pages.
- 12060 ML `MCL_FUTURE` Lock pages that become mapped.

12061 MF|SHM The symbolic constant `MAP_FAILED` shall be defined to indicate a failure from the *mmap()*  
12062 function.12063 MC1 If the Advisory Information and either the Memory Mapped Files or Shared Memory Objects  
12064 options are supported, values for *advice* used by *posix\_madvise()* shall be defined as follows:

- 12065 `POSIX_MADV_NORMAL`
- 12066 The application has no advice to give on its behavior with respect to the specified range. It

12067		is the default characteristic if no advice is given for a range of memory.
12068		<b>POSIX_MADV_SEQUENTIAL</b>
12069		The application expects to access the specified range sequentially from lower addresses to
12070		higher addresses.
12071		<b>POSIX_MADV_RANDOM</b>
12072		The application expects to access the specified range in a random order.
12073		<b>POSIX_MADV_WILLNEED</b>
12074		The application expects to access the specified range in the near future.
12075		<b>POSIX_MADV_DONTNEED</b>
12076		The application expects that it will not access the specified range in the near future.
12077		
12078	TYM	The following flags shall be defined for <i>posix_typed_mem_open()</i> :
12079		<b>POSIX_TYPED_MEM_ALLOCATE</b>
12080		Allocate on <i>mmap()</i> .
12081		<b>POSIX_TYPED_MEM_ALLOCATE_CONTIG</b>
12082		Allocate contiguously on <i>mmap()</i> .
12083		<b>POSIX_TYPED_MEM_MAP_ALLOCATABLE</b> Map on <i>mmap()</i> , without affecting allocatability.
12084		
12085		The <b>mode_t</b> , <b>off_t</b> , and <b>size_t</b> types shall be defined as described in <sys/types.h>.
12086	TYM	The <sys/mman.h> header shall define the structure <b>posix_typed_mem_info</b> , which includes at
12087		least the following member:
12088		<code>size_t posix_tmi_length</code> Maximum length which may be allocated
12089		from a typed memory object.
12090		
12091		The following shall be declared as functions and may also be defined as macros. Function
12092		prototypes shall be provided.
12093	ML	<code>int mlock(const void *, size_t);</code>
12094		<code>int mlockall(int);</code>
12095	MF SHM	<code>void *mmap(void *, size_t, int, int, int, off_t);</code>
12096	MPR	<code>int mprotect(void *, size_t, int);</code>
12097	MF SIO	<code>int msync(void *, size_t, int);</code>
12098	ML	<code>int munlock(const void *, size_t);</code>
12099		<code>int munlockall(void);</code>
12100	MF SHM	<code>int munmap(void *, size_t);</code>
12101	ADV	<code>int posix_madvise(void *, size_t, int);</code>
12102	TYM	<code>int posix_mem_offset(const void *restrict, size_t, off_t *restrict,</code>
12103		<code>size_t *restrict, int *restrict);</code>
12104		<code>int posix_typed_mem_get_info(int, struct posix_typed_mem_info *);</code>
12105		<code>int posix_typed_mem_open(const char *, int, int);</code>
12106	SHM	<code>int shm_open(const char *, int, mode_t);</code>
12107		<code>int shm_unlink(const char *);</code>
12108		

12109 **APPLICATION USAGE**

12110 None.

12111 **RATIONALE**

12112 None.

12113 **FUTURE DIRECTIONS**

12114 None.

12115 **SEE ALSO**

12116 <sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-200x, *mlock()*, *mlockall()*,  
12117 *mmap()*, *mprotect()*, *msync()*, *munlock()*, *munlockall()*, *munmap()*, *posix\_mem\_offset()*,  
12118 *posix\_typed\_mem\_get\_info()*, *posix\_typed\_mem\_open()*, *shm\_open()*, *shm\_unlink()*

12119 **CHANGE HISTORY**

12120 First released in Issue 4, Version 2.

12121 **Issue 5**

12122 Updated for alignment with the POSIX Realtime Extension.

12123 **Issue 6**

12124 The <sys/mman.h> header is marked as dependent on support for either the  
12125 `_POSIX_MAPPED_FILES`, `_POSIX_MEMLOCK`, or `_POSIX_SHARED_MEMORY` options.

12126 The following changes are made for alignment with IEEE Std 1003.1j-2000:

- 12127 • The TYM margin code is added to the list of margin codes for the <sys/mman.h> header line,  
12128 as well as for other lines.
- 12129 • The `POSIX_TYPED_MEM_ALLOCATE`, `POSIX_TYPED_MEM_ALLOCATE_CONTIG`, and  
12130 `POSIX_TYPED_MEM_MAP_ALLOCATABLE` flags are added.
- 12131 • The `posix_tmi_length` structure is added.
- 12132 • The *posix\_mem\_offset()*, *posix\_typed\_mem\_get\_info()*, and *posix\_typed\_mem\_open()* functions  
12133 are added.

12134 The `restrict` keyword is added to the prototype for *posix\_mem\_offset()*.

12135 IEEE PASC Interpretation 1003.1 #102 is applied adding the prototype for *posix\_madvise()*.

12136 **NAME**

12137 sys/msg.h — XSI message queue structures

12138 **SYNOPSIS**

12139 XSI `#include <sys/msg.h>`

12140

12141 **DESCRIPTION**

12142 The <sys/msg.h> header shall define the following constant and members of the structure  
12143 **msqid\_ds**.

12144 The following data types shall be defined through **typedef**:

12145 **msgqnum\_t**           Used for the number of messages in the message queue.

12146 **msglen\_t**            Used for the number of bytes allowed in a message queue.

12147 These types shall be unsigned integer types that are able to store values at least as large as a type  
12148 **unsigned short**.

12149 Message operation flag:

12150 **MSG\_NOERROR**       No error if big message.

12151 The **msqid\_ds** structure shall contain the following members:

12152	<code>struct ipc_perm</code>	<code>msg_perm</code>	Operation permission structure.
12153	<code>msgqnum_t</code>	<code>msg_qnum</code>	Number of messages currently on queue.
12154	<code>msglen_t</code>	<code>msg_qbytes</code>	Maximum number of bytes allowed on queue.
12155	<code>pid_t</code>	<code>msg_lspid</code>	Process ID of last <i>msgsnd()</i> .
12156	<code>pid_t</code>	<code>msg_lrpid</code>	Process ID of last <i>msgrcv()</i> .
12157	<code>time_t</code>	<code>msg_stime</code>	Time of last <i>msgsnd()</i> .
12158	<code>time_t</code>	<code>msg_rtime</code>	Time of last <i>msgrcv()</i> .
12159	<code>time_t</code>	<code>msg_ctime</code>	Time of last change.

12160 The **pid\_t**, **time\_t**, **key\_t**, **size\_t**, and **ssize\_t** types shall be defined as described in <sys/types.h>.

12161 The following shall be declared as functions and may also be defined as macros. Function  
12162 prototypes shall be provided. |

```
12163 int      msgctl(int, int, struct msqid_ds *);
12164 int      msgget(key_t, int);
12165 ssize_t  msgrcv(int, void *, size_t, long, int);
12166 int      msgsnd(int, const void *, size_t, int);
```

12167 In addition, all of the symbols from <sys/ipc.h> shall be defined when this header is included.

12168 **APPLICATION USAGE**

12169 None.

12170 **RATIONALE**

12171 None.

12172 **FUTURE DIRECTIONS**

12173 None.

12174 **SEE ALSO**

12175 <sys/types.h>, *msgctl()*, *msgget()*, *msgrcv()*, *msgsnd()*

12176 **CHANGE HISTORY**

12177 First released in Issue 2. Derived from System V Release 2.0.

12178 **NAME**

12179 sys/resource.h — definitions for XSI resource operations

12180 **SYNOPSIS**

12181 xsi #include <sys/resource.h>

12182

12183 **DESCRIPTION**

12184 The <sys/resource.h> header shall define the following symbolic constants as possible values of  
 12185 the *which* argument of *getpriority()* and *setpriority()*:

12186 PRIO\_PROCESS Identifies the *who* argument as a process ID.

12187 PRIO\_PGRP Identifies the *who* argument as a process group ID.

12188 PRIO\_USER Identifies the *who* argument as a user ID.

12189 The following type shall be defined through **typedef**:

12190 **rlim\_t** Unsigned integer type used for limit values.

12191 The following symbolic constants shall be defined:

12192 RLIM\_INFINITY A value of **rlim\_t** indicating no limit.

12193 RLIM\_SAVED\_MAX A value of type **rlim\_t** indicating an unrepresentable saved hard  
 12194 limit.

12195 RLIM\_SAVED\_CUR A value of type **rlim\_t** indicating an unrepresentable saved soft limit.

12196 On implementations where all resource limits are representable in an object of type **rlim\_t**,  
 12197 RLIM\_SAVED\_MAX and RLIM\_SAVED\_CUR need not be distinct from RLIM\_INFINITY.

12198 The following symbolic constants shall be defined as possible values of the *who* parameter of  
 12199 *getrusage()*:

12200 RUSAGE\_SELF Returns information about the current process.

12201 RUSAGE\_CHILDREN Returns information about children of the current process.

12202 The <sys/resource.h> header shall define the **rlimit** structure that includes at least the following  
 12203 members:

12204 rlim\_t rlim\_cur The current (soft) limit. |

12205 rlim\_t rlim\_max The hard limit. |

12206 The <sys/resource.h> header shall define the **rusage** structure that includes at least the following  
 12207 members: |

12208 struct timeval ru\_utime User time used. |

12209 struct timeval ru\_stime System time used. |

12210 The **timeval** structure shall be defined as described in <sys/time.h>. |

12211 The following symbolic constants shall be defined as possible values for the *resource* argument of  
 12212 *getrlimit()* and *setrlimit()*:

12213 RLIMIT\_CORE Limit on size of core dump file.

12214 RLIMIT\_CPU Limit on CPU time per process.

12215 RLIMIT\_DATA Limit on data segment size.

12216 RLIMIT\_FSIZE Limit on file size.

- 12217        RLIMIT\_NOFILE        Limit on number of open files.
- 12218        RLIMIT\_STACK        Limit on stack size.
- 12219        RLIMIT\_AS            Limit on address space size.
- 12220        The following shall be declared as functions and may also be defined as macros. Function |  
12221        prototypes shall be provided. |
- 12222        int    getpriority(int, id\_t);  
12223        int    getrlimit(int, struct rlimit \*);  
12224        int    getrusage(int, struct rusage \*);  
12225        int    setpriority(int, id\_t, int);  
12226        int    setrlimit(int, const struct rlimit \*);
- 12227        The **id\_t** type shall be defined through **typedef** as described in **<sys/types.h>**.
- 12228        Inclusion of the **<sys/resource.h>** header may also make visible all symbols from **<sys/time.h>**.
- 12229        **APPLICATION USAGE**
- 12230        None.
- 12231        **RATIONALE**
- 12232        None.
- 12233        **FUTURE DIRECTIONS**
- 12234        None.
- 12235        **SEE ALSO**
- 12236        **<sys/time.h>**, **<sys/types.h>**, the System Interfaces volume of IEEE Std 1003.1-200x, *getpriority()*,  
12237        *getrusage()*, *getrlimit()*
- 12238        **CHANGE HISTORY**
- 12239        First released in Issue 4, Version 2.
- 12240        **Issue 5**
- 12241        Large File System extensions are added.

12242 **NAME**

12243 sys/select.h — select types

12244 **SYNOPSIS**

12245 #include &lt;sys/select.h&gt;

12246 **DESCRIPTION**12247 The <sys/select.h> header shall define the **timeval** structure that includes at least the following  
12248 members:

12249 time\_t tv\_sec Seconds.

12250 suseconds\_t tv\_usec Microseconds.

12251 The **time\_t** and **suseconds\_t** types shall be defined as described in <sys/types.h>.12252 The **sigset\_t** type shall be defined as described in <signal.h>.12253 The **timespec** structure shall be defined as described in <time.h>.12254 The <sys/select.h> header shall define the **fd\_set** type as a structure. |

12255 Each of the following may be declared as a function, or defined as a macro, or both:

12256 void *FD\_CLR*(int *fd*, fd\_set \**fdset*)12257 Clears the bit for the file descriptor *fd* in the file descriptor set *fdset*.12258 int *FD\_ISSET*(int *fd*, fd\_set \**fdset*)12259 Returns a non-zero value if the bit for the file descriptor *fd* is set in the file descriptor set by  
12260 *fdset*, and 0 otherwise.12261 void *FD\_SET*(int *fd*, fd\_set \**fdset*)12262 Sets the bit for the file descriptor *fd* in the file descriptor set *fdset*.12263 void *FD\_ZERO*(fd\_set \**fdset*)12264 Initializes the file descriptor set *fdset* to have zero bits for all file descriptors.12265 If implemented as macros, these may evaluate their arguments more than once, so applications  
12266 should ensure that the arguments they supply are never expressions with side effects.

12267 The following shall be defined as a macro:

12268 **FD\_SETSIZE**12269 Maximum number of file descriptors in an **fd\_set** structure.12270 The following shall be declared as functions and may also be defined as macros. Function |  
12271 prototypes shall be provided. |12272 int pselect(int, fd\_set \*restrict, fd\_set \*restrict, fd\_set \*restrict,  
12273 const struct timespec \*restrict, const sigset\_t \*restrict);12274 int select(int, fd\_set \*restrict, fd\_set \*restrict, fd\_set \*restrict,  
12275 struct timeval \*restrict);12276 Inclusion of the <sys/select.h> header may make visible all symbols from the headers  
12277 <signal.h>, <sys/time.h>, and <time.h>.

12278 **APPLICATION USAGE**

12279       None.

12280 **RATIONALE**

12281       None.

12282 **FUTURE DIRECTIONS**

12283       None.

12284 **SEE ALSO**

12285       <signal.h>, <sys/time.h>, <sys/types.h>, <time.h>, the System Interfaces volume of

12286       IEEE Std 1003.1-200x, *pselect()*, *select()*

12287 **CHANGE HISTORY**

12288       First released in Issue 6. Derived from IEEE Std 1003.1g-2000. |

12289       The requirement for the **fd\_set** structure to have a member *fds\_bits* has been removed as per The |

12290       Open Group Base Resolution bwg2001-005. |

12291 **NAME**

12292 sys/sem.h — XSI semaphore facility

12293 **SYNOPSIS**

12294 XSI #include <sys/sem.h>

12295

12296 **DESCRIPTION**

12297 The <sys/sem.h> header shall define the following constants and structures.

12298 Semaphore operation flags:

12299 SEM\_UNDO Set up adjust on exit entry.

12300 Command definitions for the *semctl()* function shall be provided as follows:

12301 GETNCNT Get *semncnt*.

12302 GETPID Get *sempid*.

12303 GETVAL Get *semval*.

12304 GETALL Get all cases of *semval*.

12305 GETZCNT Get *semzcnt*.

12306 SETVAL Set *semval*.

12307 SETALL Set all cases of *semval*.

12308 The **semid\_ds** structure shall contain the following members:

12309 struct ipc\_perm sem\_perm Operation permission structure.

12310 unsigned short sem\_nsems Number of semaphores in set.

12311 time\_t sem\_otime Last *semop()* time.

12312 time\_t sem\_ctime Last time changed by *semctl()*.

12313 The **pid\_t**, **time\_t**, **key\_t**, and **size\_t** types shall be defined as described in <sys/types.h>.

12314 A semaphore shall be represented by an anonymous structure containing the following  
12315 members:

12316 unsigned short semval Semaphore value.

12317 pid\_t sempid Process ID of last operation.

12318 unsigned short semncnt Number of processes waiting for *semval*  
12319 to become greater than current value.

12320 unsigned short semzcnt Number of processes waiting for *semval*  
12321 to become 0.

12322 The **sembuf** structure shall contain the following members:

12323 unsigned short sem\_num Semaphore number.

12324 short sem\_op Semaphore operation.

12325 short sem\_flg Operation flags.

12326 The following shall be declared as functions and may also be defined as macros. Function |  
12327 prototypes shall be provided. |

12328 int semctl(int, int, int, ...);

12329 int semget(key\_t, int, int);

12330 int semop(int, struct sembuf \*, size\_t);

12331 In addition, all of the symbols from <sys/ipc.h> shall be defined when this header is included.

12332 **APPLICATION USAGE**

12333 None.

12334 **RATIONALE**

12335 None.

12336 **FUTURE DIRECTIONS**

12337 None.

12338 **SEE ALSO**

12339 <sys/types.h>, *semctl()*, *semget()*, *semop()*

12340 **CHANGE HISTORY**

12341 First released in Issue 2. Derived from System V Release 2.0.

12342 **NAME**

12343 sys/shm.h — XSI shared memory facility

12344 **SYNOPSIS**

12345 XSI `#include <sys/shm.h>`

12346

12347 **DESCRIPTION**

12348 The <sys/shm.h> header shall define the following symbolic constants:

12349 SHM\_RDONLY Attach read-only (else read-write).

12350 SHM\_RND Round attach address to SHMLBA.

12351 The <sys/shm.h> header shall define the following symbolic value:

12352 SHMLBA Segment low boundary address multiple.

12353 The following data types shall be defined through **typedef**:

12354 **shmatt\_t** Unsigned integer used for the number of current attaches that must be able to  
12355 store values at least as large as a type **unsigned short**.

12356 The **shmid\_ds** structure shall contain the following members:

12357 struct ipc\_perm shm\_perm Operation permission structure.

12358 size\_t shm\_segsz Size of segment in bytes.

12359 pid\_t shm\_lpid Process ID of last shared memory operation.

12360 pid\_t shm\_cpid Process ID of creator.

12361 shmatt\_t shm\_nattch Number of current attaches.

12362 time\_t shm\_atime Time of last *shmat()*.

12363 time\_t shm\_dtime Time of last *shmdt()*.

12364 time\_t shm\_ctime Time of last change by *shmctl()*.

12365 The **pid\_t**, **time\_t**, **key\_t**, and **size\_t** types shall be defined as described in <sys/types.h>.

12366 The following shall be declared as functions and may also be defined as macros. Function |  
12367 prototypes shall be provided. |

12368 void \*shmat(int, const void \*, int);

12369 int shmctl(int, int, struct shmid\_ds \*);

12370 int shmdt(const void \*);

12371 int shmget(key\_t, size\_t, int);

12372 In addition, all of the symbols from <sys/ipc.h> shall be defined when this header is included.

12373 **APPLICATION USAGE**

12374 None.

12375 **RATIONALE**

12376 None.

12377 **FUTURE DIRECTIONS**

12378 None.

12379 **SEE ALSO**

12380 <sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-200x, *shmat()*, *shmctl()*, *shmdt()*,

12381 *shmget()*

12382 **CHANGE HISTORY**

12383           First released in Issue 2. Derived from System V Release 2.0.

12384 **Issue 5**

12385           The type of *shm\_segsz* is changed from **int** to **size\_t**.

12386 **NAME**

12387 sys/socket.h — main sockets header

12388 **SYNOPSIS**

12389 #include <sys/socket.h>

12390 **DESCRIPTION**

12391 The <sys/socket.h> header shall define the type **socklen\_t**, which is an integer type of width of  
 12392 at least 32 bits; see APPLICATION USAGE.

12393 The <sys/socket.h> header shall define the unsigned integer type **sa\_family\_t**.

12394 The <sys/socket.h> header shall define the **sockaddr** structure that includes at least the  
 12395 following members:

12396 sa\_family\_t sa\_family Address family.  
 12397 char sa\_data[] Socket address (variable-length data).

12398 The **sockaddr** structure is used to define a socket address which is used in the *bind()*, *connect()*,  
 12399 *getpeername()*, *getsockname()*, *recvfrom()*, and *sendto()* functions.

12400 The <sys/socket.h> header shall define the **sockaddr\_storage** structure. This structure shall be:

- 12401 • Large enough to accommodate all supported protocol-specific address structures
- 12402 • Aligned at an appropriate boundary so that pointers to it can be cast as pointers to protocol-  
 12403 specific address structures and used to access the fields of those structures without  
 12404 alignment problems

12405 The **sockaddr\_storage** structure shall contain at least the following members:

12406 sa\_family\_t ss\_family

12407 When a **sockaddr\_storage** structure is cast as a **sockaddr** structure, the *ss\_family* field of the  
 12408 **sockaddr\_storage** structure shall map onto the *sa\_family* field of the **sockaddr** structure. When a  
 12409 **sockaddr\_storage** structure is cast as a protocol-specific address structure, the *ss\_family* field  
 12410 shall map onto a field of that structure that is of type **sa\_family\_t** and that identifies the  
 12411 protocol's address family.

12412 The <sys/socket.h> header shall define the **msghdr** structure that includes at least the following  
 12413 members:

12414 void \*msg\_name Optional address.  
 12415 socklen\_t msg\_namelen Size of address.  
 12416 struct iovec \*msg\_iov Scatter/gather array.  
 12417 int msg\_iovlen Members in msg\_iov.  
 12418 void \*msg\_control Ancillary data; see below.  
 12419 socklen\_t msg\_controllen Ancillary data buffer len.  
 12420 int msg\_flags Flags on received message.

12421 The **msghdr** structure is used to minimize the number of directly supplied parameters to the  
 12422 *recvmsg()* and *sendmsg()* functions. This structure is used as a *value-result* parameter in the  
 12423 *recvmsg()* function and *value* only for the *sendmsg()* function.

12424 The **iovec** structure shall be defined as described in <sys/uio.h>.

12425 The <sys/socket.h> header shall define the **cmsghdr** structure that includes at least the following  
 12426 members:

12427 socklen\_t cmsg\_len Data byte count, including the cmsghdr.  
 12428 int cmsg\_level Originating protocol.

12429           int            msg\_type    Protocol-specific type.

12430           The **msg\_hdr** structure is used for storage of ancillary data object information.

12431           Ancillary data consists of a sequence of pairs, each consisting of a **msg\_hdr** structure followed  
12432           by a data array. The data array contains the ancillary data message, and the **msg\_hdr** structure  
12433           contains descriptive information that allows an application to correctly parse the data.

12434           The values for *msg\_level* shall be legal values for the *level* argument to the *getsockopt()* and  
12435           *setsockopt()* functions. The system documentation shall specify the *msg\_type* definitions for the  
12436           supported protocols.

12437           Ancillary data is also possible at the socket level. The <sys/socket.h> header defines the  
12438           following macro for use as the *msg\_type* value when *msg\_level* is SOL\_SOCKET:

12439           SCM\_RIGHTS       Indicates that the data array contains the access rights to be sent or  
12440                               received.

12441           The <sys/socket.h> header defines the following macros to gain access to the data arrays in the  
12442           ancillary data associated with a message header:

12443           MSG\_DATA(*msg*)  
12444                If the argument is a pointer to a **msg\_hdr** structure, this macro shall return an unsigned  
12445                character pointer to the data array associated with the **msg\_hdr** structure.

12446           MSG\_NXTHDR(*mhdr, msg*)  
12447                If the first argument is a pointer to a **msg\_hdr** structure and the second argument is a pointer  
12448                to a **msg\_hdr** structure in the ancillary data pointed to by the *msg\_control* field of that  
12449                **msg\_hdr** structure, this macro shall return a pointer to the next **msg\_hdr** structure, or a null  
12450                pointer if this structure is the last **msg\_hdr** in the ancillary data.

12451           MSG\_FIRSTHDR(*mhdr*)  
12452                If the argument is a pointer to a **msg\_hdr** structure, this macro shall return a pointer to the  
12453                first **msg\_hdr** structure in the ancillary data associated with this **msg\_hdr** structure, or a null  
12454                pointer if there is no ancillary data associated with the **msg\_hdr** structure.

12455           The <sys/socket.h> header shall define the **linger** structure that includes at least the following  
12456           members:

12457           int   l\_onoff    Indicates whether linger option is enabled.  
12458           int   l\_linger   Linger time, in seconds.

12459           The <sys/socket.h> header shall define the following macros, with distinct integer values:

12460           SOCK\_DGRAM      Datagram socket. |

12461 RS       SOCK\_RAW       Raw Protocol Interface. |

12462           SOCK\_SEQPACKET   Sequenced-packet socket. |

12463           SOCK\_STREAM      Byte-stream socket. |

12464           The <sys/socket.h> header shall define the following macro for use as the *level* argument of  
12465           *setsockopt()* and *getsockopt()*.

12466           SOL\_SOCKET       Options to be accessed at socket level, not protocol level.

12467           The <sys/socket.h> header shall define the following macros, with distinct integer values, for  
12468           use as the *option\_name* argument in *getsockopt()* or *setsockopt()* calls:

12469           SO\_ACCEPTCONN    Socket is accepting connections.

12470	SO_BROADCAST	Transmission of broadcast messages is supported.	
12471	SO_DEBUG	Debugging information is being recorded.	
12472	SO_DONTROUTE	Bypass normal routing.	
12473	SO_ERROR	Socket error status.	
12474	SO_KEEPALIVE	Connections are kept alive with periodic messages.	
12475	SO_LINGER	Socket lingers on close.	
12476	SO_OOBINLINE	Out-of-band data is transmitted in line.	
12477	SO_RCVBUF	Receive buffer size.	
12478	SO_RCVLOWAT	Receive “low water mark”.	
12479	SO_RCVTIMEO	Receive timeout.	
12480	SO_REUSEADDR	Reuse of local addresses is supported.	
12481	SO_SNDBUF	Send buffer size.	
12482	SO_SNDLOWAT	Send “low water mark”.	
12483	SO_SNDTIMEO	Send timeout.	
12484	SO_TYPE	Socket type.	
12485	The <sys/socket.h> header shall define the following macro as the maximum <i>backlog</i> queue		
12486	length which may be specified by the <i>backlog</i> field of the <i>listen()</i> function:		
12487	SOMAXCONN	The maximum <i>backlog</i> queue length.	
12488	The <sys/socket.h> header shall define the following macros, with distinct integer values, for		
12489	use as the valid values for the <i>msg_flags</i> field in the <b>msghdr</b> structure, or the <i>flags</i> parameter in		
12490	<i>recvfrom()</i> , <i>recvmsg()</i> , <i>sendmsg()</i> , or <i>sendto()</i> calls:		
12491	MSG_CTRUNC	Control data truncated.	
12492	MSG_DONTROUTE	Send without using routing tables.	
12493	MSG_EOR	Terminates a record (if supported by the protocol).	
12494	MSG_OOB	Out-of-band data.	
12495	MSG_PEEK	Leave received data in queue.	
12496	MSG_TRUNC	Normal data truncated.	
12497	MSG_WAITALL	Attempt to fill the read buffer.	
12498	The <sys/socket.h> header shall define the following macros, with distinct integer values:		
12499	AF_INET	Internet domain sockets for use with IPv4 addresses.	
12500	IP6 AF_INET6	Internet domain sockets for use with IPv6 addresses.	
12501	AF_UNIX	UNIX domain sockets.	
12502	AF_UNSPEC	Unspecified .	
12503	The <sys/socket.h> header shall define the following macros, with distinct integer values:		
12504	SHUT_RD	Disables further receive operations.	

12505 SHUT\_RDWR Disables further send and receive operations. |

12506 SHUT\_WR Disables further send operations. |

12507 The following shall be declared as functions and may also be defined as macros. Function |

12508 prototypes shall be provided. |

```

12509 int accept(int, struct sockaddr *restrict, socklen_t *restrict);
12510 int bind(int, const struct sockaddr *, socklen_t);
12511 int connect(int, const struct sockaddr *, socklen_t);
12512 int getpeername(int, struct sockaddr *restrict, socklen_t *restrict);
12513 int getsockname(int, struct sockaddr *restrict, socklen_t *restrict);
12514 int getsockopt(int, int, int, void *restrict, socklen_t *restrict);
12515 int listen(int, int);
12516 ssize_t recv(int, void *, size_t, int);
12517 ssize_t recvfrom(int, void *restrict, size_t, int,
12518 struct sockaddr *restrict, socklen_t *restrict);
12519 ssize_t recvmsg(int, struct msghdr *, int);
12520 ssize_t send(int, const void *, size_t, int);
12521 ssize_t sendmsg(int, const struct msghdr *, int);
12522 ssize_t sendto(int, const void *, size_t, int, const struct sockaddr *,
12523 socklen_t);
12524 int setsockopt(int, int, int, const void *, socklen_t);
12525 int shutdown(int, int);
12526 int socket(int, int, int);
12527 int socketpair(int, int, int, int[2]); |

```

12528 Inclusion of <sys/socket.h> may also make visible all symbols from <sys/unistd.h>. |

12529 APPLICATION USAGE

12530 To forestall portability problems, it is recommended that applications not use values larger than |

12531  $2^{31} - 1$  for the `socklen_t` type. |

12532 The `sockaddr_storage` structure solves the problem of declaring storage for automatic variables |

12533 which is both large enough and aligned enough for storing the socket address data structure of |

12534 any family. For example, code with a file descriptor and without the context of the address |

12535 family can pass a pointer to a variable of this type, where a pointer to a socket address structure |

12536 is expected in calls such as `getpeername()`, and determine the address family by accessing the |

12537 received content after the call.

12538 The example below illustrates a data structure which aligns on a 64-bit boundary. An |

12539 implementation-defined field `_ss_align` following `_ss_pad1` is used to force a 64-bit alignment |

12540 which covers proper alignment good enough for needs of at least `sockaddr_in6` (IPv6) and |

12541 `sockaddr_in` (IPv4) address data structures. The size of padding field `_ss_pad1` depends on the |

12542 chosen alignment boundary. The size of padding field `_ss_pad2` depends on the value of overall |

12543 size chosen for the total size of the structure. This size and alignment are represented in the |

12544 above example by implementation-defined (not required) constants `_SS_MAXSIZE` (chosen |

12545 value 128) and `_SS_ALIGNMENT` (with chosen value 8). Constants `_SS_PAD1SIZE` (derived |

12546 value 6) and `_SS_PAD2SIZE` (derived value 112) are also for illustration and not required. The |

12547 implementation-defined definitions and structure field names above start with an underscore to |

12548 denote implementation private name space. Portable code is not expected to access or reference |

12549 those fields or constants. |

```

12550 /*
12551  * Desired design of maximum size and alignment.
12552  */

```

```

12553     #define _SS_MAXSIZE 128
12554         /* Implementation-defined maximum size. */
12555     #define _SS_ALIGNSIZE (sizeof(int64_t))
12556         /* Implementation-defined desired alignment. */
12557
12557     /*
12558      * Definitions used for sockaddr_storage structure paddings design.
12559      */
12560     #define _SS_PAD1SIZE (_SS_ALIGNSIZE - sizeof(sa_family_t))
12561     #define _SS_PAD2SIZE (_SS_MAXSIZE - (sizeof(sa_family_t)+
12562         _SS_PAD1SIZE + _SS_ALIGNSIZE))
12563     struct sockaddr_storage {
12564         sa_family_t  ss_family; /* Address family. */
12565     /*
12566      * Following fields are implementation-defined. */
12567     /*
12568         char _ss_pad1[_SS_PAD1SIZE];
12569         /* 6-byte pad; this is to make implementation-defined
12570          pad up to alignment field that follows explicit in
12571          the data structure. */
12572         int64_t  _ss_align; /* Field to force desired structure
12573          storage alignment. */
12574         char _ss_pad2[_SS_PAD2SIZE];
12575         /* 112-byte pad to achieve desired size,
12576          _SS_MAXSIZE value minus size of ss_family
12577          __ss_pad1, __ss_align fields is 112. */
12578     };
12579 RATIONALE
12580     None.
12581 FUTURE DIRECTIONS
12582     None.
12583 SEE ALSO
12584     <sys/uid.h>, the System Interfaces volume of IEEE Std 1003.1-200x, accept(), bind(), connect(),
12585     getpeername(), getsockname(), getsockopt(), listen(), recv(), recvfrom(), recvmsg(), send(),
12586     sendmsg(), sendto(), setsockopt(), shutdown(), socket(), socketpair()
12587 CHANGE HISTORY
12588     First released in Issue 6. Derived from the XNS, Issue 5.2 specification.
12589     The restrict keyword is added to the prototypes for accept(), getpeername(), getsockname(),
12590     getsockopt(), and recvfrom().

```

12591 NAME

12592 sys/stat.h — data returned by the stat() function

12593 SYNOPSIS

12594 #include <sys/stat.h>

12595 DESCRIPTION

12596 The <sys/stat.h> header shall define the structure of the data returned by the functions *fstat()*,  
12597 *lstat()*, and *stat()*.

12598 The **stat** structure shall contain at least the following members:

12599	dev_t	st_dev	ID of device containing file.
12600	ino_t	st_ino	File serial number.
12601	mode_t	st_mode	Mode of file (see below).
12602	nlink_t	st_nlink	Number of hard links to the file.
12603	uid_t	st_uid	User ID of file.
12604	gid_t	st_gid	Group ID of file.
12605 XSI	dev_t	st_rdev	Device ID (if file is character or block special).
12606	off_t	st_size	For regular files, the file size in bytes.
12607			For symbolic links, the length in bytes of the
12608			pathname contained in the symbolic link.
12609 SHM			For a shared memory object, the length in bytes.
12610 TYM			For a typed memory object, the length in bytes.
12611			For other file types, the use of this field is
12612			unspecified
12613	time_t	st_atime	Time of last access.
12614	time_t	st_mtime	Time of last data modification.
12615	time_t	st_ctime	Time of last status change.
12616 XSI	blksize_t	st_blksize	A file system-specific preferred I/O block size for
12617			this object. In some file system types, this may
12618			vary from file to file.
12619	blkcnt_t	st_blocks	Number of blocks allocated for this object.
12620			

12621 File serial number and device ID taken together uniquely identify the file within the system. The  
12622 **blkcnt\_t**, **blksize\_t**, **dev\_t**, **ino\_t**, **mode\_t**, **nlink\_t**, **uid\_t**, **gid\_t**, **off\_t**, and **time\_t** types shall be  
12623 defined as described in <sys/types.h>. Times shall be given in seconds since the Epoch.

12624 Unless otherwise specified, the structure members *st\_mode*, *st\_ino*, *st\_dev*, *st\_uid*, *st\_gid*, *st\_atime*,  
12625 *st\_ctime*, and *st\_mtime* shall have meaningful values for all file types defined in  
12626 IEEE Std 1003.1-200x.

12627 For symbolic links, the *st\_mode* member shall contain meaningful information, which can be  
12628 used with the file type macros described below, that take a *mode* argument. The *st\_size* member  
12629 shall contain the length, in bytes, of the pathname contained in the symbolic link. File mode bits  
12630 and the contents of the remaining members of the **stat** structure are unspecified. The value  
12631 returned in the *st\_size* field shall be the length of the contents of the symbolic link, and shall not  
12632 count a trailing null if one is present.

12633 The following symbolic names for the values of type *mode\_t* shall also be defined.

12634 File type:

12635 XSI	S_IFMT	Type of file.
12636	S_IFBLK	Block special.

12637		S_IFCHR	Character special.
12638		S_IFIFO	FIFO special.
12639		S_IFREG	Regular.
12640		S_IFDIR	Directory.
12641		S_IFLNK	Symbolic link.
12642		S_IFSOCK	Socket.
12643		File mode bits:	
12644	S_IRWXU	Read, write, execute/search by owner.	
12645		S_IRUSR	Read permission, owner.
12646		S_IWUSR	Write permission, owner.
12647		S_IXUSR	Execute/search permission, owner.
12648	S_IRWXG	Read, write, execute/search by group.	
12649		S_IRGRP	Read permission, group.
12650		S_IWGRP	Write permission, group.
12651		S_IXGRP	Execute/search permission, group.
12652	S_IRWXO	Read, write, execute/search by others.	
12653		S_IROTH	Read permission, others.
12654		S_IWOTH	Write permission, others.
12655		S_IXOTH	Execute/search permission, others.
12656	S_ISUID	Set-user-ID on execution.	
12657	S_ISGID	Set-group-ID on execution.	
12658	XSI	S_ISVTX	On directories, restricted deletion flag.
12659		The bits defined by S_IRUSR, S_IWUSR, S_IXUSR, S_IRGRP, S_IWGRP, S_IXGRP, S_IROTH,	
12660	XSI	S_IWOTH, S_IXOTH, S_ISUID, S_ISGID, and S_ISVTX shall be unique.	
12661		S_IRWXU is the bitwise-inclusive OR of S_IRUSR, S_IWUSR, and S_IXUSR.	
12662		S_IRWXG is the bitwise-inclusive OR of S_IRGRP, S_IWGRP, and S_IXGRP.	
12663		S_IRWXO is the bitwise-inclusive OR of S_IROTH, S_IWOTH, and S_IXOTH.	
12664		Implementations may OR other implementation-defined bits into S_IRWXU, S_IRWXG, and	
12665		S_IRWXO, but they shall not overlap any of the other bits defined in this volume of	
12666		IEEE Std 1003.1-200x. The <i>file permission bits</i> are defined to be those corresponding to the	
12667		bitwise-inclusive OR of S_IRWXU, S_IRWXG, and S_IRWXO.	
12668		The following macros shall be provided to test whether a file is of the specified type. The value	
12669		<i>m</i> supplied to the macros is the value of <i>st_mode</i> from a <b>stat</b> structure. The macro shall evaluate	
12670		to a non-zero value if the test is true; 0 if the test is false.	
12671	S_ISBLK( <i>m</i> )	Test for a block special file.	
12672	S_ISCHR( <i>m</i> )	Test for a character special file.	

12673        **S\_ISDIR(*m*)**        Test for a directory.

12674        **S\_ISFIFO(*m*)**        Test for a pipe or FIFO special file.

12675        **S\_ISREG(*m*)**        Test for a regular file.

12676        **S\_ISLNK(*m*)**        Test for a symbolic link.

12677        **S\_ISSOCK(*m*)**        Test for a socket.

12678        The implementation may implement message queues, semaphores, or shared memory objects as distinct file types. The following macros shall be provided to test whether a file is of the specified type. The value of the *buf* argument supplied to the macros is a pointer to a **stat** structure. The macro shall evaluate to a non-zero value if the specified object is implemented as a distinct file type and the specified file type is contained in the **stat** structure referenced by *buf*. Otherwise, the macro shall evaluate to zero.

12684        **S\_TYPEISMQ(*buf*)**    Test for a message queue.

12685        **S\_TYPEISSEM(*buf*)**    Test for a semaphore.

12686        **S\_TYPEISSHM(*buf*)**    Test for a shared memory object.

12687 TYP    The implementation may implement typed memory objects as distinct file types, and the following macro shall test whether a file is of the specified type. The value of the *buf* argument supplied to the macros is a pointer to a **stat** structure. The macro shall evaluate to a non-zero value if the specified object is implemented as a distinct file type and the specified file type is contained in the **stat** structure referenced by *buf*. Otherwise, the macro shall evaluate to zero.

12692        **S\_TYPEISTMO(*buf*)**    Test macro for a typed memory object.

12694        The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

12696        `int     chmod(const char *, mode_t);`

12697        `int     fchmod(int, mode_t);`

12698        `int     fstat(int, struct stat *);`

12699        `int     lstat(const char *restrict, struct stat *restrict);`

12700        `int     mkdir(const char *, mode_t);`

12701        `int     mkfifo(const char *, mode_t);`

12702 XSI    `int     mknod(const char *, mode_t, dev_t);`

12703        `int     stat(const char *restrict, struct stat *restrict);`

12704        `mode_t umask(mode_t);`

**12705 APPLICATION USAGE**

12706        Use of the macros is recommended for determining the type of a file.

**12707 RATIONALE**

12708        A conforming C-language application must include **<sys/stat.h>** for functions that have arguments or return values of type **mode\_t**, so that symbolic values for that type can be used. An alternative would be to require that these constants are also defined by including **<sys/types.h>**.

12712        The **S\_ISUID** and **S\_ISGID** bits may be cleared on any write, not just on *open()*, as some historical implementations do it.

12714        System calls that update the time entry fields in the **stat** structure must be documented by the implementors. POSIX-conforming systems should not update the time entry fields for functions listed in the System Interfaces volume of IEEE Std 1003.1-200x unless the standard requires that

- 12717 they do, except in the case of documented extensions to the standard.
- 12718 Note that *st\_dev* must be unique within a Local Area Network (LAN) in a “system” made up of  
12719 multiple computers’ file systems connected by a LAN.
- 12720 Networked implementations of a POSIX-conforming system must guarantee that all files visible  
12721 within the file tree (including parts of the tree that may be remotely mounted from other  
12722 machines on the network) on each individual processor are uniquely identified by the  
12723 combination of the *st\_ino* and *st\_dev* fields.
- 12724 **FUTURE DIRECTIONS**
- 12725 No new *S\_IFMT* symbolic names for the file type values of **mode\_t** will be defined by  
12726 IEEE Std 1003.1-200x; if new file types are required, they will only be testable through *S\_ISxx()*  
12727 macros instead.
- 12728 **SEE ALSO**
- 12729 <sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-200x, *chmod()*, *fchmod()*, *fstat()*,  
12730 *lstat()*, *mkdir()*, *mkfifo()*, *mknod()*, *stat()*, *umask()*
- 12731 **CHANGE HISTORY**
- 12732 First released in Issue 1. Derived from Issue 1 of the SVID.
- 12733 **Issue 5**
- 12734 The DESCRIPTION is updated for alignment with POSIX Realtime Extension.
- 12735 The type of *st\_blksize* is changed from **long** to **blksize\_t**; the type of *st\_blocks* is changed from  
12736 **long** to **blkcnt\_t**.
- 12737 **Issue 6**
- 12738 The *S\_TYPEISMQ()*, *S\_TYPEISSEM()*, and *S\_TYPEISSHM()* macros are unconditionally  
12739 mandated.
- 12740 The Open Group Corrigendum U035/4 is applied. In the DESCRIPTION, the types **blksize\_t**  
12741 and **blkcnt\_t** have been described.
- 12742 The following new requirements on POSIX implementations derive from alignment with the  
12743 Single UNIX Specification:
- 12744 • The **dev\_t**, **ino\_t**, **mode\_t**, **nlink\_t**, **uid\_t**, **gid\_t**, **off\_t**, and **time\_t** types are mandated.
- 12745 *S\_IFSOCK* and *S\_ISSOCK* are added for sockets.
- 12746 The description of **stat** structure members is changed to reflect contents when file type is a  
12747 symbolic link.
- 12748 The test macro *S\_TYPEISTMO* is added for alignment with IEEE Std 1003.1j-2000.
- 12749 The **restrict** keyword is added to the prototypes for *lstat()* and *stat()*.
- 12750 The *lstat()* function is now mandatory.

12751 NAME

12752 sys/statvfs.h — VFS File System information structure

12753 SYNOPSIS

12754 XSI #include <sys/statvfs.h>

12755

12756 DESCRIPTION

12757 The <sys/statvfs.h> header shall define the **statvfs** structure that includes at least the following  
12758 members:

- 12759 unsigned long f\_bsize File system block size.
- 12760 unsigned long f\_frsize Fundamental file system block size.
- 12761 fsblkcnt\_t f\_blocks Total number of blocks on file system in units of *f\_frsize*.
- 12762 fsblkcnt\_t f\_bfree Total number of free blocks.
- 12763 fsblkcnt\_t f\_bavail Number of free blocks available to  
12764 non-privileged process.
- 12765 fsfilcnt\_t f\_files Total number of file serial numbers.
- 12766 fsfilcnt\_t f\_ffree Total number of free file serial numbers.
- 12767 fsfilcnt\_t f\_favail Number of file serial numbers available to  
12768 non-privileged process.
- 12769 unsigned long f\_fsid File system ID.
- 12770 unsigned long f\_flag Bit mask of *f\_flag* values.
- 12771 unsigned long f\_namemax Maximum filename length.

12772 The **fsblkcnt\_t** and **fsfilcnt\_t** types shall be defined as described in <sys/types.h>.

12773 The following flags for the *f\_flag* member shall be defined:

- 12774 ST\_RDONLY Read-only file system.
- 12775 ST\_NOSUID Does not support setuid/setgid semantics.

12776 The following shall be declared as functions and may also be defined as macros. Function |  
12777 prototypes shall be provided. |

```
12778 int statvfs(const char *restrict, struct statvfs *restrict);
12779 int fstatvfs(int, struct statvfs *);
```

12780 APPLICATION USAGE

12781 None.

12782 RATIONALE

12783 None.

12784 FUTURE DIRECTIONS

12785 None.

12786 SEE ALSO

12787 <sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-200x, *fstatvfs()*, *statvfs()*

12788 CHANGE HISTORY

12789 First released in Issue 4, Version 2.

12790 Issue 5

12791 The type of *f\_blocks*, *f\_bfree*, and *f\_bavail* is changed from **unsigned long** to **fsblkcnt\_t**; the type  
12792 of *f\_files*, *f\_ffree*, and *f\_favail* is changed from **unsigned long** to **fsfilcnt\_t**.

12793 **Issue 6**

12794 The Open Group Corrigendum U035/5 is applied. In the DESCRIPTION, the types **fsblkcnt\_t**  
12795 and **fsfilcnt\_t** have been described.

12796 The **restrict** keyword is added to the prototype for *statvfs()*.

12797 NAME

12798 sys/time.h — time types

12799 SYNOPSIS

12800 XSI #include <sys/time.h>

12801

12802 DESCRIPTION

12803 The <sys/time.h> header shall define the **timeval** structure that includes at least the following  
12804 members:

12805 time\_t tv\_sec Seconds.

12806 suseconds\_t tv\_usec Microseconds.

12807 The <sys/time.h> header shall define the **itimerval** structure that includes at least the following  
12808 members:

12809 struct timeval it\_interval Timer interval.

12810 struct timeval it\_value Current value.

12811 The **time\_t** and **suseconds\_t** types shall be defined as described in <sys/types.h>.

12812 The **fd\_set** type shall be defined as described in <sys/select.h>.

12813 The <sys/time.h> header shall define the following values for the *which* argument of *getitimer()*  
12814 and *setitimer()*:

12815 ITIMER\_REAL Decrements in real time.

12816 ITIMER\_VIRTUAL Decrements in process virtual time.

12817 ITIMER\_PROF Decrements both in process virtual time and when the system is running  
12818 on behalf of the process.

12819 The following shall be defined as described in <sys/select.h>:

12820 FD\_CLR()

12821 FD\_ISSET()

12822 FD\_SET()

12823 FD\_ZERO()

12824 FD\_SETSIZE()

12825 The following shall be declared as functions and may also be defined as macros. Function  
12826 prototypes shall be provided.

12827 int getitimer(int, struct itimerval \*);

12828 int gettimeofday(struct timeval \*restrict, void \*restrict);

12829 int select(int, fd\_set \*restrict, fd\_set \*restrict, fd\_set \*restrict,  
12830 struct timeval \*restrict);

12831 int setitimer(int, const struct itimerval \*restrict,  
12832 struct itimerval \*restrict);

12833 int utimes(const char \*, const struct timeval [2]); (**LEGACY**)

12834 Inclusion of the <sys/time.h> header may make visible all symbols from the <sys/select.h>  
12835 header.

12836 **APPLICATION USAGE**

12837 None.

12838 **RATIONALE**

12839 None.

12840 **FUTURE DIRECTIONS**

12841 None.

12842 **SEE ALSO**12843 <sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-200x, *getitimer()*, *gettimeofday()*,12844 *select()*, *setitimer()*12845 **CHANGE HISTORY**

12846 First released in Issue 4, Version 2.

12847 **Issue 5**12848 The type of *tv\_usec* is changed from **long** to **suseconds\_t**.12849 **Issue 6**12850 The **restrict** keyword is added to the prototypes for *gettimeofday()*, *select()*, and *setitimer()*. |

12851 The note is added that inclusion of this header may also make symbols visible from |

12852 &lt;sys/socket.h&gt;. |

12853 The *utimes()* function is marked LEGACY. |

12854 **NAME**

12855 sys/timeb.h — additional definitions for date and time

12856 **SYNOPSIS**

12857 XSI #include <sys/timeb.h>

12858

12859 **DESCRIPTION**

12860 The <sys/timeb.h> header shall define the **timeb** structure that includes at least the following  
12861 members:

12862	time_t	time	The seconds portion of the current time.
12863	unsigned short	millitm	The milliseconds portion of the current time.
12864	short	timezone	The local timezone in minutes west of Greenwich.
12865	short	dstflag	TRUE if Daylight Savings Time is in effect.

12866 The **time\_t** type shall be defined as described in <sys/types.h>.

12867 The following shall be declared as a function and may also be defined as a macro. A function |  
12868 prototype shall be provided. |

12869 int ftime(struct timeb \*); (**LEGACY**)

12870 **APPLICATION USAGE**

12871 None.

12872 **RATIONALE**

12873 None.

12874 **FUTURE DIRECTIONS**

12875 None.

12876 **SEE ALSO**

12877 <sys/types.h>, <time.h>

12878 **CHANGE HISTORY**

12879 First released in Issue 4, Version 2.

12880 **Issue 6**

12881 The *ftime()* function is marked LEGACY.

12882 **NAME**

12883 sys/times.h — file access and modification times structure

12884 **SYNOPSIS**

12885 #include <sys/times.h>

12886 **DESCRIPTION**

12887 The <sys/times.h> header shall define the structure **tms**, which is returned by *times()* and  
 12888 includes at least the following members:

- 12889 clock\_t tms\_utime User CPU time.
- 12890 clock\_t tms\_stime System CPU time.
- 12891 clock\_t tms\_cutime User CPU time of terminated child processes.
- 12892 clock\_t tms\_cstime System CPU time of terminated child processes.

12893 The **clock\_t** type shall be defined as described in <sys/types.h>.

12894 The following shall be declared as a function and may also be defined as a macro. A function |  
 12895 prototype shall be provided. |

12896 clock\_t times(struct tms \*);

12897 **APPLICATION USAGE**

12898 None.

12899 **RATIONALE**

12900 None.

12901 **FUTURE DIRECTIONS**

12902 None.

12903 **SEE ALSO**

12904 <sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-200x, *times()*

12905 **CHANGE HISTORY**

12906 First released in Issue 1. Derived from Issue 1 of the SVID.

12907 **NAME**

12908 sys/types.h — data types

12909 **SYNOPSIS**

12910 #include &lt;sys/types.h&gt;

12911 **DESCRIPTION**

12912 The &lt;sys/types.h&gt; header shall include definitions for at least the following types:

12913	<b>blkcnt_t</b>	Used for file block counts.
12914	<b>blksize_t</b>	Used for block sizes.
12915 XSI	<b>clock_t</b>	Used for system times in clock ticks or CLOCKS_PER_SEC; see
12916		<time.h>.
12917 TMR	<b>clockid_t</b>	Used for clock ID type in the clock and timer functions.
12918	<b>dev_t</b>	Used for device IDs.
12919 XSI	<b>fsblkcnt_t</b>	Used for file system block counts.
12920 XSI	<b>fsfilcnt_t</b>	Used for file system file counts.
12921	<b>gid_t</b>	Used for group IDs.
12922 XSI	<b>id_t</b>	Used as a general identifier; can be used to contain at least a <b>pid_t</b> ,
12923		<b>uid_t</b> , or <b>gid_t</b> .
12924	<b>ino_t</b>	Used for file serial numbers.
12925 XSI	<b>key_t</b>	Used for XSI interprocess communication.
12926	<b>mode_t</b>	Used for some file attributes.
12927	<b>nlink_t</b>	Used for link counts.
12928	<b>off_t</b>	Used for file sizes.
12929	<b>pid_t</b>	Used for process IDs and process group IDs.
12930 THR	<b>pthread_attr_t</b>	Used to identify a thread attribute object.
12931 BAR	<b>pthread_barrier_t</b>	Used to identify a barrier.
12932 BAR	<b>pthread_barrierattr_t</b>	Used to define a barrier attributes object.
12933 THR	<b>pthread_cond_t</b>	Used for condition variables.
12934 THR	<b>pthread_condattr_t</b>	Used to identify a condition attribute object.
12935 THR	<b>pthread_key_t</b>	Used for thread-specific data keys.
12936 THR	<b>pthread_mutex_t</b>	Used for mutexes.
12937 THR	<b>pthread_mutexattr_t</b>	Used to identify a mutex attribute object.
12938 THR	<b>pthread_once_t</b>	Used for dynamic package initialization.
12939 THR	<b>pthread_rwlock_t</b>	Used for read-write locks.
12940 THR	<b>pthread_rwlockattr_t</b>	Used for read-write lock attributes.
12941 SPI	<b>pthread_spinlock_t</b>	Used to identify a spin lock.
12942 THR	<b>pthread_t</b>	Used to identify a thread.

12943	<b>size_t</b>	Used for sizes of objects.
12944	<b>ssize_t</b>	Used for a count of bytes or an error indication.
12945 XSI	<b>suseconds_t</b>	Used for time in microseconds
12946	<b>time_t</b>	Used for time in seconds.
12947 TMR	<b>timer_t</b>	Used for timer ID returned by <i>timer_create()</i> .
12948	<b>uid_t</b>	Used for user IDs.
12949 XSI	<b>useconds_t</b>	Used for time in microseconds.
12950	All of the types shall be defined as arithmetic types of an appropriate length, with the following	
12951	exceptions:	
12952 XSI	<b>key_t</b>	
12953 THR	<b>pthread_attr_t</b>	
12954 BAR	<b>pthread_barrier_t</b>	
12955	<b>pthread_barrierattr_t</b>	
12956 THR	<b>pthread_cond_t</b>	
12957	<b>pthread_condattr_t</b>	
12958	<b>pthread_key_t</b>	
12959	<b>pthread_mutex_t</b>	
12960	<b>pthread_mutexattr_t</b>	
12961	<b>pthread_once_t</b>	
12962	<b>pthread_rwlock_t</b>	
12963	<b>pthread_rwlockattr_t</b>	
12964 SPI	<b>pthread_spinlock_t</b>	
12965 TRC	<b>trace_attr_t</b>	
12966	<b>trace_event_id_t</b>	
12967 TRC TEF	<b>trace_event_set_t</b>	
12968 TRC	<b>trace_id_t</b>	
12969		
12970	Additionally:	
12971	• <b>mode_t</b> shall be an integer type.	
12972	• <b>nlink_t</b> , <b>uid_t</b> , <b>gid_t</b> , and <b>id_t</b> shall be integer types.	
12973	• <b>blkcnt_t</b> and <b>off_t</b> shall be signed integer types.	
12974 XSI	• <b>fsblkcnt_t</b> , <b>fsfilcnt_t</b> , and <b>ino_t</b> shall be defined as unsigned integer types.	
12975	• <b>size_t</b> shall be an unsigned integer type.	
12976	• <b>blksize_t</b> , <b>pid_t</b> , and <b>ssize_t</b> shall be signed integer types.	
12977	• <b>time_t</b> and <b>clock_t</b> shall be integer or real-floating types.	
12978 XSI	The type <b>ssize_t</b> shall be capable of storing values at least in the range $[-1, \{\text{SSIZE\_MAX}\}]$ . The	
12979	type <b>useconds_t</b> shall be an unsigned integer type capable of storing values at least in the range	
12980	$[0, 1\,000\,000]$ . The type <b>suseconds_t</b> shall be a signed integer type capable of storing values at	
12981	least in the range $[-1, 1\,000\,000]$ .	
12982	The implementation shall support one or more programming environments in which the widths	
12983	of <b>blksize_t</b> , <b>pid_t</b> , <b>size_t</b> , <b>ssize_t</b> , <b>suseconds_t</b> , and <b>useconds_t</b> are no greater than the width of	
12984	type <b>long</b> . The names of these programming environments can be obtained using the <i>confstr()</i>	
12985	function or the <i>getconf</i> utility.	

12986 There are no defined comparison or assignment operators for the following types: |

- 12987 THR **pthread\_attr\_t**
- 12988 BAR **pthread\_barrier\_t**
- 12989 **pthread\_barrierattr\_t**
- 12990 THR **pthread\_cond\_t**
- 12991 **pthread\_condattr\_t**
- 12992 **pthread\_mutex\_t**
- 12993 **pthread\_mutexattr\_t**
- 12994 **pthread\_rwlock\_t**
- 12995 **pthread\_rwlockattr\_t**
- 12996 SPI **pthread\_spinlock\_t**
- 12997 TRC **trace\_attr\_t**
- 12998

12999 **APPLICATION USAGE**

13000 None.

13001 **RATIONALE**

13002 None.

13003 **FUTURE DIRECTIONS**

13004 None.

13005 **SEE ALSO**

13006 <time.h>, the System Interfaces volume of IEEE Std 1003.1-200x, *confstr()*, the Shell and Utilities |  
13007 volume of IEEE Std 1003.1-200x, *getconf* |

13008 **CHANGE HISTORY**

13009 First released in Issue 1. Derived from Issue 1 of the SVID.

13010 **Issue 5**

13011 The **clockid\_t** and **timer\_t** types are defined for alignment with the POSIX Realtime Extension.

13012 The types **blkcnt\_t**, **blksize\_t**, **fsblkcnt\_t**, **fsfilcnt\_t**, and **suseconds\_t** are added.

13013 Large File System extensions are added.

13014 Updated for alignment with the POSIX Threads Extension.

13015 **Issue 6**

13016 The **pthread\_barrier\_t**, **pthread\_barrierattr\_t**, and **pthread\_spinlock\_t** types are added for  
13017 alignment with IEEE Std 1003.1j-2000.

13018 The margin code is changed from XSI to THR for the **pthread\_rwlock\_t** and  
13019 **pthread\_rwlockattr\_t** types as Read-Write Locks have been absorbed into the POSIX Threads  
13020 option. The threads types are now marked THR.

13021 **NAME**

13022 sys/uio.h — definitions for vector I/O operations

13023 **SYNOPSIS**13024 XSI 

```
#include <sys/uio.h>
```

13025

13026 **DESCRIPTION**13027 The <sys/uio.h> header shall define the **iovec** structure that includes at least the following  
13028 members:

13029 void \*iov\_base Base address of a memory region for input or output.

13030 size\_t iov\_len The size of the memory pointed to by *iov\_base*.13031 The <sys/uio.h> header uses the **iovec** structure for scatter/gather I/O.13032 The **ssize\_t** and **size\_t** types shall be defined as described in <sys/types.h>.13033 The following shall be declared as functions and may also be defined as macros. Function |  
13034 prototypes shall be provided. |

13035 ssize\_t readv(int, const struct iovec \*, int);

13036 ssize\_t writev(int, const struct iovec \*, int);

13037 **APPLICATION USAGE**13038 The implementation can put a limit on the number of scatter/gather elements which can be  
13039 processed in one call. The symbol {IOV\_MAX} defined in <limits.h> should always be used to  
13040 learn about the limits instead of assuming a fixed value.13041 **RATIONALE**13042 Traditionally, the maximum number of scatter/gather elements the system can process in one  
13043 call were described by the symbolic value {UIO\_MAXIOV}. In IEEE Std 1003.1-200x this value  
13044 was replaced by the constant {IOV\_MAX} which can be found in <limits.h>.13045 **FUTURE DIRECTIONS**

13046 None.

13047 **SEE ALSO**13048 <limits.h>, <sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-200x, *read()*, *write()*13049 **CHANGE HISTORY**

13050 First released in Issue 4, Version 2.

13051 **Issue 6**

13052 Text referring to scatter/gather I/O is added to the DESCRIPTION.

13053 **NAME**

13054 sys/un.h — definitions for UNIX domain sockets

13055 **SYNOPSIS**

13056 #include <sys/un.h>

13057 **DESCRIPTION**

13058 The <sys/un.h> header shall define the **sockaddr\_un** structure that includes at least the  
13059 following members:

13060 sa\_family\_t sun\_family Address family.  
13061 char sun\_path[] Socket pathname.

13062 The **sockaddr\_un** structure is used to store addresses for UNIX domain sockets. Values of this  
13063 type shall be cast by applications to **struct sockaddr** for use with socket functions.

13064 The **sa\_family\_t** type shall be defined as described in <sys/socket.h>.

13065 **APPLICATION USAGE**

13066 The size of *sun\_path* has intentionally been left undefined. This is because different  
13067 implementations use different sizes. For example, BSD4.3 uses a size of 108, and BSD4.4 uses a  
13068 size of 104. Since most implementations originate from BSD versions, the size is typically in the  
13069 range 92 to 108.

13070 Applications should not assume a particular length for *sun\_path* or assume that it can hold  
13071 `_POSIX_PATH_MAX` characters (255).

13072 **RATIONALE**

13073 None.

13074 **FUTURE DIRECTIONS**

13075 None.

13076 **SEE ALSO**

13077 <sys/socket.h>, the System Interfaces volume of IEEE Std 1003.1-200x, *bind()*, *socket()*,  
13078 *socketpair()*

13079 **CHANGE HISTORY**

13080 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

13081 **NAME**

13082 sys/utsname.h — system name structure

13083 **SYNOPSIS**

13084 #include &lt;sys/utsname.h&gt;

13085 **DESCRIPTION**13086 The <sys/utsname.h> header shall define the structure **utsname** which shall include at least the  
13087 following members:

13088 char sysname[] Name of this implementation of the operating system.  
13089 char nodename[] Name of this node within an implementation-defined  
13090 communications network.  
13091 char release[] Current release level of this implementation.  
13092 char version[] Current version level of this release.  
13093 char machine[] Name of the hardware type on which the system is running.

13094 The character arrays are of unspecified size, but the data stored in them shall be terminated by a  
13095 null byte.

13096 The following shall be declared as a function and may also be defined as a macro:

13097 int uname(struct utsname \*);

13098 **APPLICATION USAGE**

13099 None.

13100 **RATIONALE**

13101 None.

13102 **FUTURE DIRECTIONS**

13103 None.

13104 **SEE ALSO**13105 The System Interfaces volume of IEEE Std 1003.1-200x, *uname()*13106 **CHANGE HISTORY**

13107 First released in Issue 1. Derived from Issue 1 of the SVID.

13108 NAME

13109 sys/wait.h — declarations for waiting

13110 SYNOPSIS

13111 #include <sys/wait.h>

13112 DESCRIPTION

13113 The <sys/wait.h> header shall define the following symbolic constants for use with *waitpid()*:

13114 WNOHANG Do not hang if no status is available; return immediately.

13115 WUNTRACED Report status of stopped child process.

13116 The <sys/wait.h> header shall define the following macros for analysis of process status values:

13117 WEXITSTATUS Return exit status.

13118 XSI WIFCONTINUED True if child has been continued

13119 WIFEXITED True if child exited normally.

13120 WIFSIGNALED True if child exited due to uncaught signal.

13121 WIFSTOPPED True if child is currently stopped.

13122 WSTOPSIG Return signal number that caused process to stop.

13123 WTERMSIG Return signal number that caused process to terminate.

13124 XSI The following symbolic constants shall be defined as possible values for the *options* argument to *waitid()*:

13126 WEXITED Wait for processes that have exited.

13127 WSTOPPED Status is returned for any child that has stopped upon receipt of a signal.

13128 WCONTINUED Status is returned for any child that was stopped and has been continued.

13129 WNOHANG Return immediately if there are no children to wait for.

13130 WNOWAIT Keep the process whose status is returned in *infp* in a waitable state.

13131 The type *idtype\_t* shall be defined as an enumeration type whose possible values shall include at least the following:

13133 P\_ALL

13134 P\_PID

13135 P\_PGID

13136

13137 The *id\_t* and *pid\_t* types shall be defined as described in <sys/types.h>.

13138 XSI The *siginfo\_t* type shall be defined as described in <signal.h>.

13139 The *rusage* structure shall be defined as described in <sys/resource.h>.

13140 Inclusion of the <sys/wait.h> header may also make visible all symbols from <signal.h> and <sys/resource.h>.

13142 The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

13144 pid\_t wait(int \*);

13145 XSI int waitid(idtype\_t, id\_t, siginfo\_t \*, int);

13146 pid\_t waitpid(pid\_t, int \*, int);

13147 **APPLICATION USAGE**

13148           None.

13149 **RATIONALE**

13150           None.

13151 **FUTURE DIRECTIONS**

13152           None.

13153 **SEE ALSO**13154           <signal.h>, <sys/resource.h>, <sys/types.h>, <sys/wait.h>, the System Interfaces volume of  
13155           IEEE Std 1003.1-200x, *wait()*, *waitid()*13156 **CHANGE HISTORY**

13157           First released in Issue 3.

13158           Entry included for alignment with the POSIX.1-1988 standard.

13159 **Issue 6**13160           The *wait3()* function is removed.

13161 **NAME**13162        **syslog** — definitions for system error logging13163 **SYNOPSIS**13164 **XSI**        #include <syslog.h>

13165

13166 **DESCRIPTION**13167        The **<syslog.h>** header shall define the following symbolic constants, zero or more of which may  
13168        be OR'ed together to form the *logopt* option of *openlog()*:13169        **LOG\_PID**                Log the process ID with each message.13170        **LOG\_CONS**               Log to the system console on error.13171        **LOG\_NDELAY**             Connect to syslog daemon immediately.13172        **LOG\_ODELAY**             Delay open until *syslog()* is called.13173        **LOG\_NOWAIT**            Do not wait for child processes.13174        The following symbolic constants shall be defined as possible values of the *facility* argument to  
13175        *openlog()*:13176        **LOG\_KERN**               Reserved for message generated by the system.13177        **LOG\_USER**               Message generated by a process.13178        **LOG\_MAIL**               Reserved for message generated by mail system.13179        **LOG\_NEWS**               Reserved for message generated by news system.13180        **LOG\_UUCP**               Reserved for message generated by UUCP system.13181        **LOG\_DAEMON**            Reserved for message generated by system daemon.13182        **LOG\_AUTH**               Reserved for message generated by authorization daemon.13183        **LOG\_CRON**               Reserved for message generated by the clock daemon.13184        **LOG\_LPR**                 Reserved for message generated by printer system.13185        **LOG\_LOCAL0**             Reserved for local use.13186        **LOG\_LOCAL1**             Reserved for local use.13187        **LOG\_LOCAL2**             Reserved for local use.13188        **LOG\_LOCAL3**             Reserved for local use.13189        **LOG\_LOCAL4**             Reserved for local use.13190        **LOG\_LOCAL5**             Reserved for local use.13191        **LOG\_LOCAL6**             Reserved for local use.13192        **LOG\_LOCAL7**             Reserved for local use.13193        The following shall be declared as macros for constructing the *maskpri* argument to *setlogmask()*.  
13194        The following macros expand to an expression of type **int** when the argument *pri* is an  
13195        expression of type **int**:13196        **LOG\_MASK(*pri*)**        A mask for priority *pri*.13197        The following constants shall be defined as possible values for the *priority* argument of *syslog()*:

- 13198        LOG\_EMERG        A panic condition was reported to all processes.
- 13199        LOG\_ALERT        A condition that should be corrected immediately.
- 13200        LOG\_CRIT        A critical condition.
- 13201        LOG\_ERR        An error message.
- 13202        LOG\_WARNING     A warning message.
- 13203        LOG\_NOTICE     A condition requiring special handling.
- 13204        LOG\_INFO       A general information message.
- 13205        LOG\_DEBUG      A message useful for debugging programs.

13206        The following shall be declared as functions and may also be defined as macros. Function |  
 13207        prototypes shall be provided. |

```
13208        void    closelog(void);
13209        void    openlog(const char *, int, int);
13210        int    setlogmask(int);
13211        void    syslog(int, const char *, ...);
```

13212 **APPLICATION USAGE**

13213        None.

13214 **RATIONALE**

13215        None.

13216 **FUTURE DIRECTIONS**

13217        None.

13218 **SEE ALSO**

13219        The System Interfaces volume of IEEE Std 1003.1-200x, *closelog()*

13220 **CHANGE HISTORY**

13221        First released in Issue 4, Version 2.

13222 **Issue 5**

13223        Moved to X/Open UNIX to BASE.

13224 **NAME**

13225 tar.h — extended tar definitions

13226 **SYNOPSIS**

13227 #include &lt;tar.h&gt;

13228 **DESCRIPTION**

13229 The &lt;tar.h&gt; header shall define header block definitions as follows.

13230 General definitions:

13231

13232

13233

13234

13235

13236

Name	Description	Value
TMAGIC	"ustar"	ustar plus null byte.
TMAGLEN	6	Length of the above.
TVERSION	"00"	00 without a null byte.
TVERSLEN	2	Length of the above.

13237

*Typeflag* field definitions:

13238

13239

13240

13241

13242

13243

13244

13245

13246

13247

13248

Name	Description	Value
REGTYPE	'0'	Regular file.
AREGTYPE	'\0'	Regular file.
LNKTYPE	'1'	Link.
SYMTYPE	'2'	Symbolic link.
CHRTYPE	'3'	Character special.
BLKTYPE	'4'	Block special.
DIRTYPE	'5'	Directory.
FIFOTYPE	'6'	FIFO special.
CONTTYPE	'7'	Reserved.

13249

*Mode* field bit definitions (octal):

13250

13251

13252

13253

13254 XSI

13255

13256

13257

13258

13259

13260

13261

13262

13263

Name	Description	Value
TSUID	04000	Set UID on execution.
TSGID	02000	Set GID on execution.
TSVTX	01000	On directories, restricted deletion flag.
TUREAD	00400	Read by owner.
TUWRITE	00200	Write by owner special.
TUEXEC	00100	Execute/search by owner.
TGREAD	00040	Read by group.
TGWRITE	00020	Write by group.
TGEXEC	00010	Execute/search by group.
TOREAD	00004	Read by other.
TOWRITE	00002	Write by other.
TOEXEC	00001	Execute/search by other.

13264 **APPLICATION USAGE**

13265           None.

13266 **RATIONALE**

13267           None.

13268 **FUTURE DIRECTIONS**

13269           None.

13270 **SEE ALSO**13271           The Shell and Utilities volume of IEEE Std 1003.1-200x, *pax*13272 **CHANGE HISTORY**

13273           First released in Issue 3. Derived from the entry in the POSIX.1-1988 standard.

13274 **Issue 6**13275           The **SEE ALSO** section now refers to *pax* since the Shell and Utilities volume of  
13276           IEEE Std 1003.1-200x no longer contains the *tar* utility.

13277 **NAME**

13278 termios.h — define values for termios

13279 **SYNOPSIS**

13280 #include <termios.h>

13281 **DESCRIPTION**

13282 The <termios.h> header contains the definitions used by the terminal I/O interfaces (see  
13283 Chapter 11 (on page 183) for the structures and names defined).

13284 **The termios Structure**

13285 The following data types shall be defined through **typedef**:

13286 **cc\_t** Used for terminal special characters.

13287 **speed\_t** Used for terminal baud rates.

13288 **tcflag\_t** Used for terminal modes.

13289 The above types shall be all unsigned integer types.

13290 The implementation shall support one or more programming environments in which the widths |  
13291 of **cc\_t**, **speed\_t**, and **tcflag\_t** are no greater than the width of type **long**. The names of these |  
13292 programming environments can be obtained using the *confstr()* function or the *getconf* utility. |

13293 The **termios** structure shall be defined, and shall include at least the following members: |

- 13294 tcflag\_t c\_iflag Input modes.
- 13295 tcflag\_t c\_oflag Output modes.
- 13296 tcflag\_t c\_cflag Control modes.
- 13297 tcflag\_t c\_lflag Local modes.
- 13298 cc\_t c\_cc[NCCS] Control characters.

13299 A definition shall be provided for:

13300 NCCS Size of the array *c\_cc* for control characters.

13301 The following subscript names for the array *c\_cc* shall be defined:

13302

13303

13304

13305

13306

13307

13308

13309

13310

13311

13312

13313

13314

13315

Subscript Usage		Description
Canonical Mode	Non-Canonical Mode	
VEOF		EOF character.
VEOL		EOL character.
VERASE		ERASE character.
VINTR	VINTR	INTR character.
VKILL		KILL character.
	VMIN	MIN value.
VQUIT	VQUIT	QUIT character.
VSTART	VSTART	START character.
VSTOP	VSTOP	STOP character.
VSUSP	VSUSP	SUSP character.
	VTIME	TIME value.

13316 The subscript values shall be unique, except that the VMIN and VTIME subscripts may have the  
13317 same values as the VEOF and VEOL subscripts, respectively.

13318 The following flags shall be provided.

13319 **Input Modes**

13320 The *c\_iflag* field describes the basic terminal input control:

- 13321 BRKINT           Signal interrupt on break.
- 13322 ICRNL            Map CR to NL on input.
- 13323 IGNBRK          Ignore break condition.
- 13324 IGNCR           Ignore CR.
- 13325 IGNPAR          Ignore characters with parity errors.
- 13326 INLCR           Map NL to CR on input.
- 13327 INPCK           Enable input parity check.
- 13328 ISTRIP          Strip character.
- 13329 XSI IXANY        Enable any character to restart output.
- 13330 IXOFF           Enable start/stop input control.
- 13331 IXON            Enable start/stop output control.
- 13332 PARMRK          Mark parity errors.

13333 **Output Modes**

13334 The *c\_oflag* field specifies the system treatment of output:

- 13335 OPOST           Post-process output.
- 13336 XSI ONLCR        Map NL to CR-NL on output.
- 13337 OCRNL          Map CR to NL on output.
- 13338 ONOCR          No CR output at column 0.
- 13339 ONLRET         NL performs CR function.
- 13340 OFILL          Use fill characters for delay.
- 13341 NLDLY          Select newline delays:
- 13342                NL0     <newline> type 0.
- 13343                NL1     <newline> type 1.
- 13344 CRDLY          Select carriage-return delays:
- 13345                CR0     Carriage-return delay type 0.
- 13346                CR1     Carriage-return delay type 1.
- 13347                CR2     Carriage-return delay type 2.
- 13348                CR3     Carriage-return delay type 3.
- 13349 TABDLY         Select horizontal-tab delays:
- 13350                TAB0     Horizontal-tab delay type 0.
- 13351                TAB1     Horizontal-tab delay type 1.
- 13352                TAB2     Horizontal-tab delay type 2.

13353	TAB3	Expand tabs to spaces.
13354	BSDLY	Select backspace delays:
13355	BS0	Backspace-delay type 0.
13356	BS1	Backspace-delay type 1.
13357	VTDLY	Select vertical-tab delays:
13358	VT0	Vertical-tab delay type 0.
13359	VT1	Vertical-tab delay type 1.
13360	FFDLY	Select form-feed delays:
13361	FF0	Form-feed delay type 0.
13362	FF1	Form-feed delay type 1.

13363 **Baud Rate Selection**

13364 The input and output baud rates are stored in the **termios** structure. These are the valid values  
13365 for objects of type **speed\_t**. The following values shall be defined, but not all baud rates need be  
13366 supported by the underlying hardware.

13367	B0	Hang up
13368	B50	50 baud
13369	B75	75 baud
13370	B110	110 baud
13371	B134	134.5 baud
13372	B150	150 baud
13373	B200	200 baud
13374	B300	300 baud
13375	B600	600 baud
13376	B1200	1200 baud
13377	B1800	1800 baud
13378	B2400	2400 baud
13379	B4800	4800 baud
13380	B9600	9600 baud
13381	B19200	19200 baud
13382	B38400	38400 baud



13415 **Line Control**

13416 The following symbolic constants for use with *tflush()* shall be defined:

13417 TCIFLUSH Flush pending input. Flush untransmitted output.

13418 TCIOFLUSH Flush both pending input and untransmitted output.

13419 TCOFLUSH Flush untransmitted output.

13420 The following symbolic constants for use with *tflow()* shall be defined:

13421 TCIOFF Transmit a STOP character, intended to suspend input data.

13422 TCION Transmit a START character, intended to restart input data.

13423 TCOOFF Suspend output.

13424 TCOON Restart output.

13425 The following shall be declared as functions and may also be defined as macros. Function  
13426 prototypes shall be provided.

```

13427 speed_t cfgetispeed(const struct termios *);
13428 speed_t cfgetospeed(const struct termios *);
13429 int cfsetispeed(struct termios *, speed_t);
13430 int cfsetospeed(struct termios *, speed_t);
13431 int tcdrain(int);
13432 int tcflow(int, int);
13433 int tcflush(int, int);
13434 int tcgetattr(int, struct termios *);
13435 xsi pid_t tcgetsid(int);
13436 int tcsendbreak(int, int);
13437 int tcsetattr(int, int, const struct termios *);

```

13438 **APPLICATION USAGE**

13439 The following names are reserved for XSI-conformant systems to use as an extension to the  
13440 above; therefore strictly conforming applications shall not use them:

13441	CBAUD	EXTB	VDSUSP
13442	DEFECHO	FLUSHO	VLNEXT
13443	ECHOCTL	LOBLK	VREPRINT
13444	ECHOKE	PENDIN	VSTATUS
13445	ECHOPRT	SWTCH	VWERASE
13446	EXTA	VDISCARD	

13447 **RATIONALE**

13448 None.

13449 **FUTURE DIRECTIONS**

13450 None.

13451 **SEE ALSO**

13452 The System Interfaces volume of IEEE Std 1003.1-200x, *cfgetispeed()*, *cfgetospeed()*, *cfsetispeed()*,  
13453 *cfsetospeed()*, *getconf()*, *tcdrain()*, *tcflow()*, *tcflush()*, *tcgetattr()*, *tcgetsid()*, *tcsendbreak()*, *tcsetattr()*,  
13454 the Shell and Utilities volume of IEEE Std 1003.1-200x, *getconf*, Chapter 11 (on page 183)

13455 **CHANGE HISTORY**

13456 First released in Issue 3.

13457 Entry included for alignment with the ISO POSIX-1 standard.

13458 **Issue 6**

13459 The LEGACY symbols IUCLC, ULCUC, and XCASE are removed. |

13460 FIPS 151-2 requirements for the symbols CS7, CS8, CSTOPB, PARODD, and PARENB are |

13461 reaffirmed. |

13462 NAME

13463 tgmath.h — type-generic macros

13464 SYNOPSIS

13465 #include <tgmath.h>

13466 DESCRIPTION

13467 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
13468 conflict between the requirements described here and the ISO C standard is unintentional. This  
13469 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

13470 The <tgmath.h> header shall include the headers <math.h> and <complex.h> and shall define  
13471 several type-generic macros.

13472 Of the functions contained within the <math.h> and <complex.h> headers without an *f* (**float**) or  
13473 *l* (**long double**) suffix, several have one or more parameters whose corresponding real type is  
13474 **double**. For each such function, except *modf()*, there shall be a corresponding type-generic  
13475 macro. The parameters whose corresponding real type is **double** in the function synopsis are  
13476 generic parameters. Use of the macro invokes a function whose corresponding real type and  
13477 type domain are determined by the arguments for the generic parameters.

13478 Use of the macro invokes a function whose generic parameters have the corresponding real type  
13479 determined as follows:

- 13480 • First, if any argument for generic parameters has type **long double**, the type determined is  
13481 **long double**.
- 13482 • Otherwise, if any argument for generic parameters has type **double** or is of integer type, the  
13483 type determined is **double**.
- 13484 • Otherwise, the type determined is **float**.

13485 For each unsuffixed function in the <math.h> header for which there is a function in the  
13486 <complex.h> header with the same name except for a *c* prefix, the corresponding type-generic  
13487 macro (for both functions) has the same name as the function in the <math.h> header. The  
13488 corresponding type-generic macro for *fabs()* and *cabs()* is *fabs()*.

13489  
13490  
13491  
13492  
13493  
13494  
13495  
13496  
13497  
13498  
13499  
13500  
13501  
13502  
13503  
13504  
13505  
13506  
13507

<math.h> Function	<complex.h> Function	Type-Generic Macro
<i>acos()</i>	<i>cacos()</i>	<i>acos()</i>
<i>asin()</i>	<i>casin()</i>	<i>asin()</i>
<i>atan()</i>	<i>catan()</i>	<i>atan()</i>
<i>acosh()</i>	<i>cacosh()</i>	<i>acosh()</i>
<i>asinh()</i>	<i>casinh()</i>	<i>asinh()</i>
<i>atanh()</i>	<i>catanh()</i>	<i>atanh()</i>
<i>cos()</i>	<i>ccos()</i>	<i>cos()</i>
<i>sin()</i>	<i>csin()</i>	<i>sin()</i>
<i>tan()</i>	<i>ctan()</i>	<i>tan()</i>
<i>cosh()</i>	<i>ccosh()</i>	<i>cosh()</i>
<i>sinh()</i>	<i>csinh()</i>	<i>sinh()</i>
<i>tanh()</i>	<i>ctanh()</i>	<i>tanh()</i>
<i>exp()</i>	<i>cexp()</i>	<i>exp()</i>
<i>log()</i>	<i>clog()</i>	<i>log()</i>
<i>pow()</i>	<i>cpow()</i>	<i>pow()</i>
<i>sqrt()</i>	<i>csqrt()</i>	<i>sqrt()</i>
<i>fabs()</i>	<i>cabs()</i>	<i>fabs()</i>

13508 If at least one argument for a generic parameter is complex, then use of the macro invokes a  
 13509 complex function; otherwise, use of the macro invokes a real function.

13510 For each unsuffixed function in the <math.h> header without a *c*-prefixed counterpart in the  
 13511 <complex.h> header, the corresponding type-generic macro has the same name as the function.  
 13512 These type-generic macros are:

13513	<i>atan2()</i>	<i>fma()</i>	<i>llround()</i>	<i>remainder()</i>	
13514	<i>cbrt()</i>	<i>fmax()</i>	<i>log10()</i>	<i>remquo()</i>	
13515	<i>ceil()</i>	<i>fmin()</i>	<i>log1p()</i>	<i>rint()</i>	
13516	<i>copysign()</i>	<i>fmod()</i>	<i>log2()</i>	<i>round()</i>	
13517	<i>erf()</i>	<i>frexp()</i>	<i>logb()</i>	<i>scalbn()</i>	
13518	<i>erfc()</i>	<i>hypot()</i>	<i>lrint()</i>	<i>scalbln()</i>	
13519	<i>exp2()</i>	<i>ilogb()</i>	<i>lround()</i>	<i>tgamma()</i>	
13520	<i>expm1()</i>	<i>ldexp()</i>	<i>nearbyint()</i>	<i>trunc()</i>	
13521	<i>fdim()</i>	<i>lgamma()</i>	<i>nextafter()</i>		
13522	<i>floor()</i>	<i>llrint()</i>	<i>nexttoward()</i>		

13523 If all arguments for generic parameters are real, then use of the macro invokes a real function;  
 13524 otherwise, use of the macro results in undefined behavior.

13525 For each unsuffixed function in the <complex.h> header that is not a *c*-prefixed counterpart to a  
 13526 function in the <math.h> header, the corresponding type-generic macro has the same name as  
 13527 the function. These type-generic macros are:

13528	<i>carg()</i>	
13529	<i>cimag()</i>	
13530	<i>conj()</i>	
13531	<i>cproj()</i>	
13532	<i>creal()</i>	

13533 Use of the macro with any real or complex argument invokes a complex function.

13534 **APPLICATION USAGE**

13535 With the declarations:

```
13536 #include <tgmath.h>
13537 int n;
13538 float f;
13539 double d;
13540 long double ld;
13541 float complex fc;
13542 double complex dc;
13543 long double complex ldc;
```

13544 functions invoked by use of type-generic macros are shown in the following table:

Macro	Use Invokes
<i>exp(n)</i>	<i>exp(n)</i> , the function
<i>acosh(f)</i>	<i>acosh(f)</i>
<i>sin(d)</i>	<i>sin(d)</i> , the function
<i>atan(ld)</i>	<i>atanl(ld)</i>

13551

13552

13553

13554

13555

13556

13557

13558

13559

13560

13561

13562

13563

13564

13565

13566

13567

13568

13569

Macro	Use Invokes
<i>log(fc)</i>	<i>clogf(fc)</i>
<i>sqrt(dc)</i>	<i>csqrt(dc)</i>
<i>pow(ldc,f)</i>	<i>cpowl(ldc, f)</i>
<i>remainder(n,n)</i>	<i>remainder(n, n)</i> , the function
<i>nextafter(d,f)</i>	<i>nextafter(d, f)</i> , the function
<i>nexttoward(f,ld)</i>	<i>nexttowardf(f, ld)</i>
<i>copysign(n,ld)</i>	<i>copysignl(n, ld)</i>
<i>ceil(fc)</i>	Undefined behavior
<i>rint(dc)</i>	Undefined behavior
<i>fmax(ldc,ld)</i>	Undefined behavior
<i>carg(n)</i>	<i>carg(n)</i> , the function
<i>cproj(f)</i>	<i>cprojf(f)</i>
<i>creal(d)</i>	<i>creal(d)</i> , the function
<i>cimag(ld)</i>	<i>cimagl(ld)</i>
<i>cabs(fc)</i>	<i>cabsf(fc)</i>
<i>carg(dc)</i>	<i>carg(dc)</i> , the function
<i>cproj(ldc)</i>	<i>cprojl(ldc)</i>

13570 **RATIONALE**

13571

13572

13573

13574

13575

Type-generic macros allow calling a function whose type is determined by the argument type, as is the case for C operators such as '+' and '\*'. For example, with a type-generic *cos()* macro, the expression *cos(float)x* will have type **float**. This feature enables writing more portably efficient code and alleviates need for awkward casting and suffixing in the process of porting or adjusting precision. Generic math functions are a widely appreciated feature of Fortran.

13576

13577

13578

13579

The only arguments that affect the type resolution are the arguments corresponding to the parameters that have type **double** in the synopsis. Hence the type of a type-generic call to *nexttoward()*, whose second parameter is **long double** in the synopsis, is determined solely by the type of the first argument.

13580

13581

13582

The term “type-generic” was chosen over the proposed alternatives of intrinsic and overloading. The term is more specific than intrinsic, which already is widely used with a more general meaning, and reflects a closer match to Fortran’s generic functions than to C++ overloading.

13583

13584

The macros are placed in their own header in order not to silently break old programs that include the **<math.h>** header; for example, with:

13585

```
printf ("%e", sin(x))
```

13586

13587

*modf(double, double \*)* is excluded because no way was seen to make it safe without complicating the type resolution.

13588

13589

13590

The implementation might, as an extension, endow appropriate ones of the macros that IEEE Std 1003.1-200x specifies only for real arguments with the ability to invoke the complex functions.

13591

13592

13593

IEEE Std 1003.1-200x does not prescribe any particular implementation mechanism for generic macros. It could be implemented simply with built-in macros. The generic macro for *sqrt()*, for example, could be implemented with:

13594

13595

```
#undef sqrt
#define sqrt(x) __BUILTIN_GENERIC_sqrt(x)
```

13596

13597

Generic macros are designed for a useful level of consistency with C++ overloaded math functions.

13598 The great majority of existing C programs are expected to be unaffected when the <tgmath.h>  
13599 header is included instead of the <math.h> or <complex.h> headers. Generic macros are similar  
13600 to the ISO/IEC 9899:1999 standard library masking macros, though the semantic types of return  
13601 values differ.

13602 The ability to overload on integer as well as floating types would have been useful for some  
13603 functions; for example, *copysign()*. Overloading with different numbers of arguments would  
13604 have allowed reusing names; for example, *remainder()* for *remquo()*. However, these facilities  
13605 would have complicated the specification; and their natural consistent use, such as for a floating  
13606 *abs()* or a two-argument *atan()*, would have introduced further inconsistencies with the  
13607 ISO/IEC 9899:1999 standard for insufficient benefit.

13608 The ISO C standard in no way limits the implementation's options for efficiency, including  
13609 inlining library functions.

#### 13610 **FUTURE DIRECTIONS**

13611 None.

#### 13612 **SEE ALSO**

13613 <math.h>, <complex.h>, the System Interfaces volume of IEEE Std 1003.1-200x, *cabs()*, *fabs()*,  
13614 *modf()*

#### 13615 **CHANGE HISTORY**

13616 First released in Issue 6. Included for alignment with the ISO/IEC 9899:1999 standard.

13617 NAME

13618 time.h — time types

13619 SYNOPSIS

13620 #include <time.h>

13621 DESCRIPTION

13622 CX Some of the functionality described on this reference page extends the ISO C standard.  
13623 Applications shall define the appropriate feature test macro (see the System Interfaces volume of  
13624 IEEE Std 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of these  
13625 symbols in this header.

13626 The <time.h> header shall declare the structure **tm**, which shall include at least the following  
13627 members:

- 13628 int tm\_sec Seconds [0,60].
- 13629 int tm\_min Minutes [0,59].
- 13630 int tm\_hour Hour [0,23].
- 13631 int tm\_mday Day of month [1,31].
- 13632 int tm\_mon Month of year [0,11].
- 13633 int tm\_year Years since 1900.
- 13634 int tm\_wday Day of week [0,6] (Sunday =0).
- 13635 int tm\_yday Day of year [0,365].
- 13636 int tm\_isdst Daylight savings flag.

13637 The value of *tm\_isdst* shall be positive if Daylight Saving Time is in effect, 0 if Daylight Saving  
13638 Time is not in effect, and negative if the information is not available.

13639 The <time.h> header shall define the following symbolic names:

- 13640 NULL Null pointer constant.
- 13641 CLOCKS\_PER\_SEC A number used to convert the value returned by the *clock()* function into  
13642 seconds.

13643 TMR|CPT **CLOCK\_PROCESS\_CPUTIME\_ID**  
13644 The identifier of the CPU-time clock associated with the process making a  
13645 *clock()* or *timer\*()* function call.

13646 TMR|TCT **CLOCK\_THREAD\_CPUTIME\_ID**  
13647 The identifier of the CPU-time clock associated with the thread making a  
13648 *clock()* or *timer\*()* function call.

13649 TMR The <time.h> header shall declare the structure **timespec**, which has at least the following  
13650 members:

- 13651 time\_t tv\_sec Seconds.
- 13652 long tv\_nsec Nanoseconds.

13653 The <time.h> header shall also declare the **itimerspec** structure, which has at least the following  
13654 members:

- 13655 struct timespec it\_interval Timer period.
- 13656 struct timespec it\_value Timer expiration.

13657 The following manifest constants shall be defined:

- 13658 **CLOCK\_REALTIME** The identifier of the system-wide realtime clock.
- 13659 **TIMER\_ABSTIME** Flag indicating time is absolute with respect to the clock associated with a  
13660 timer.

13661 MON CLOCK\_MONOTONIC  
 13662 The identifier for the system-wide monotonic clock, which is defined as a  
 13663 clock whose value cannot be set via *clock\_settime()* and which cannot  
 13664 have backward clock jumps. The maximum possible clock jump shall be  
 13665 implementation-defined.

13666 TMR The *clock\_t*, *size\_t*, *time\_t*, *clockid\_t*, and *timer\_t* types shall be defined as described in  
 13667 <sys/types.h>.

13668 XSI Although the value of CLOCKS\_PER\_SEC is required to be 1 million on all XSI-conformant  
 13669 systems, it may be variable on other systems, and it should not be assumed that  
 13670 CLOCKS\_PER\_SEC is a compile-time constant.

13671 XSI The <time.h> header shall provide a declaration for *getdate\_err*.

13672 The following shall be declared as functions and may also be defined as macros. Function  
 13673 prototypes shall be provided.

```

13674 char      *asctime(const struct tm *);
13675 TSF char      *asctime_r(const struct tm *restrict, char *restrict);
13676 clock_t    clock(void);
13677 CPT int      clock_getcpuclockid(pid_t, clockid_t *);
13678 TMR int      clock_getres(clockid_t, struct timespec *);
13679 int      clock_gettime(clockid_t, struct timespec *);
13680 CS int      clock_nanosleep(clockid_t, int, const struct timespec *,
13681                          struct timespec *);
13682 TMR int      clock_settime(clockid_t, const struct timespec *);
13683 char      *ctime(const time_t *);
13684 TSF char      *ctime_r(const time_t *, char *);
13685 double     difftime(time_t, time_t);
13686 XSI struct tm *getdate(const char *);
13687 struct tm *gmtime(const time_t *);
13688 TSF struct tm *gmtime_r(const time_t *restrict, struct tm *restrict);
13689 struct tm *localtime(const time_t *);
13690 TSF struct tm *localtime_r(const time_t *restrict, struct tm *restrict);
13691 time_t     mktime(struct tm *);
13692 TMR int      nanosleep(const struct timespec *, struct timespec *);
13693 size_t     strftime(char *restrict, size_t, const char *restrict,
13694                   const struct tm *restrict);
13695 XSI char      *strptime(const char *restrict, const char *restrict,
13696                      struct tm *restrict);
13697 time_t     time(time_t *);
13698 TMR int      timer_create(clockid_t, struct sigevent *restrict,
13699                      timer_t *restrict);
13700 int      timer_delete(timer_t);
13701 int      timer_gettime(timer_t, struct itimerspec *);
13702 int      timer_getoverrun(timer_t);
13703 int      timer_settime(timer_t, int, const struct itimerspec *restrict,
13704                      struct itimerspec *restrict);
13705 CX void     tzset(void);
13706
13707 The following shall be declared as variables:
13708 XSI extern int    daylight;
13709 extern long   timezone;
```

13710 cx     extern char   \*tzname[];  
13711

13712 cx     Inclusion of the **<time.h>** header may make visible all symbols from the **<signal.h>** header.

### 13713 APPLICATION USAGE

13714         The range [0,60] for *tm\_sec* allows for the occasional leap second.

13715         *tm\_year* is a signed value; therefore, years before 1900 may be represented.

13716         To obtain the number of clock ticks per second returned by the *times()* function, applications  
13717         should call *sysconf(\_SC\_CLK\_TCK)*.

### 13718 RATIONALE

13719         The range [0,60] seconds allows for positive or negative leap seconds. The formal definition of  
13720         UTC does not permit double leap seconds, so all mention of double leap seconds has been  
13721         removed, and the range shortened from the former [0,61] seconds seen in previous versions of  
13722         POSIX.

### 13723 FUTURE DIRECTIONS

13724         None.

### 13725 SEE ALSO

13726         **<sys/types.h>**, the System Interfaces volume of IEEE Std 1003.1-200x, *asctime()*, *clock()*,  
13727         *clock\_getcpuclockid()*, *clock\_getres()*, *clock\_nanosleep()*, *ctime()*, *difftime()*, *getdate()*, *gmtime()*,  
13728         *localtime()*, *mktime()*, *nanosleep()*, *strftime()*, *strptime()*, *sysconf()*, *time()*, *timer\_create()*,  
13729         *timer\_delete()*, *timer\_getoverrun()*, *tzname*, *tzset()*, *utime()*

### 13730 CHANGE HISTORY

13731         First released in Issue 1. Derived from Issue 1 of the SVID.

#### 13732 Issue 5

13733         The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX  
13734         Threads Extension.

#### 13735 Issue 6

13736         The Open Group Corrigendum U035/6 is applied. In the DESCRIPTION, the types **clockid\_t**  
13737         and **timer\_t** have been described.

13738         The following changes are made for alignment with the ISO POSIX-1: 1996 standard:

- 13739
  - The POSIX timer-related functions are now marked as part of the Timers option.

13740         The symbolic name CLK\_TCK is removed. Application usage is added describing how its  
13741         equivalent functionality can be obtained using *sysconf()*.

13742         The *clock\_getcpuclockid()* function and manifest constants CLOCK\_PROCESS\_CPUTIME\_ID and  
13743         CLOCK\_THREAD\_CPUTIME\_ID are added for alignment with IEEE Std 1003.1d-1999.

13744         The manifest constant CLOCK\_MONOTONIC and the *clock\_nanosleep()* function are added for  
13745         alignment with IEEE Std 1003.1j-2000.

13746         The following changes are made for alignment with the ISO/IEC 9899: 1999 standard:

- 13747
  - The range for seconds is changed from [0,61] to [0,60].
  - The **restrict** keyword is added to the prototypes for *asctime\_r()*, *gmtime\_r()*, *localtime\_r()*,  
13748         *strftime()*, *strptime()*, *timer\_create()*, and *timer\_settime()*.

13750         IEEE PASC Interpretation 1003.1 #84 is applied adding the statement that symbols from the  
13751         **<signal.h>** header may be made visible when the **<time.h>** header is included.

13752

Extensions beyond the ISO C standard are now marked.

13753 NAME

13754 trace.h — tracing

13755 SYNOPSIS

13756 TRC #include <trace.h>

13757

13758 DESCRIPTION

13759 The <trace.h> header shall define the **posix\_trace\_event\_info** structure that includes at least the  
13760 following members:

13761	trace_event_id_t	posix_event_id
13762	pid_t	posix_pid
13763	void	*posix_prog_address
13764	int	posix_truncation_status
13765	struct timespec	posix_timestamp
13766 THR	pthread_t	posix_thread_id

13767

13768 The <trace.h> header shall define the **posix\_trace\_status\_info** structure that includes at least the  
13769 following members:

13770	int	posix_stream_status
13771	int	posix_stream_full_status
13772	int	posix_stream_overrun_status
13773 TRL	int	posix_stream_flush_status
13774	int	posix_stream_flush_error
13775	int	posix_log_overrun_status
13776	int	posix_log_full_status

13777

13778 The <trace.h> header shall define the following symbols:

13779	POSIX_TRACE_RUNNING
13780	POSIX_TRACE_SUSPENDED
13781	POSIX_TRACE_FULL
13782	POSIX_TRACE_NOT_FULL
13783	POSIX_TRACE_NO_OVERRUN
13784	POSIX_TRACE_OVERRUN
13785 TRL	POSIX_TRACE_FLUSHING
13786	POSIX_TRACE_NOT_FLUSHING
13787	POSIX_TRACE_NOT_TRUNCATED
13788	POSIX_TRACE_TRUNCATED_READ
13789	POSIX_TRACE_TRUNCATED_RECORD
13790 TRL	POSIX_TRACE_FLUSH
13791	POSIX_TRACE_LOOP
13792	POSIX_TRACE_UNTIL_FULL
13793 TRI	POSIX_TRACE_CLOSE_FOR_CHILD
13794	POSIX_TRACE_INHERITED
13795 TRL	POSIX_TRACE_APPEND
13796	POSIX_TRACE_LOOP
13797	POSIX_TRACE_UNTIL_FULL
13798 TEF	POSIX_TRACE_FILTER
13799 TRL	POSIX_TRACE_FLUSH_START
13800	POSIX_TRACE_FLUSH_STOP
13801	POSIX_TRACE_OVERFLOW

13802 POSIX\_TRACE\_RESUME  
 13803 POSIX\_TRACE\_START  
 13804 POSIX\_TRACE\_STOP  
 13805 POSIX\_TRACE\_UNNAMED\_USER\_EVENT

13806 The following types shall be defined as described in <sys/types.h>:

13807 **trace\_attr\_t**  
 13808 **trace\_id\_t**  
 13809 **trace\_event\_id\_t**  
 13810 TEF **trace\_event\_set\_t**  
 13811

13812 The following shall be declared as functions and may also be defined as macros. Function |  
 13813 prototypes shall be provided. |

```
13814 int posix_trace_attr_destroy(trace_attr_t *);
13815 int posix_trace_attr_getclockres(const trace_attr_t *,
13816     struct timespec *);
13817 int posix_trace_attr_getcreatetime(const trace_attr_t *,
13818     struct timespec *);
13819 int posix_trace_attr_getgenversion(const trace_attr_t *, char *);
13820 TRI int posix_trace_attr_getinherited(const trace_attr_t *restrict,
13821     int *restrict);
13822 TRL int posix_trace_attr_getlogfullpolicy(const trace_attr_t *restrict,
13823     int *restrict);
13824 int posix_trace_attr_getlogsize(const trace_attr_t *restrict,
13825     size_t *restrict);
13826 int posix_trace_attr_getmaxdatasize(const trace_attr_t *restrict,
13827     size_t *restrict);
13828 int posix_trace_attr_getmaxsystemeventsz(const trace_attr_t *restrict,
13829     size_t *restrict);
13830 int posix_trace_attr_getmaxusereventsiz(const trace_attr_t *restrict,
13831     size_t, size_t *restrict);
13832 int posix_trace_attr_getname(const trace_attr_t *, char *);
13833 int posix_trace_attr_getstreamfullpolicy(const trace_attr_t *restrict,
13834     int *restrict);
13835 int posix_trace_attr_getstreamsize(const trace_attr_t *restrict,
13836     size_t *restrict);
13837 int posix_trace_attr_init(trace_attr_t *);
13838 TRI int posix_trace_attr_setinherited(trace_attr_t *, int);
13839 TRL int posix_trace_attr_setlogfullpolicy(trace_attr_t *, int);
13840 int posix_trace_attr_setlogsize(trace_attr_t *, size_t);
13841 int posix_trace_attr_setmaxdatasize(trace_attr_t *, size_t);
13842 int posix_trace_attr_setname(trace_attr_t *, const char *);
13843 int posix_trace_attr_setstreamsize(trace_attr_t *, size_t);
13844 int posix_trace_attr_setstreamfullpolicy(trace_attr_t *, int);
13845 int posix_trace_clear(trace_id_t);
13846 TRL int posix_trace_close(trace_id_t);
13847 int posix_trace_create(pid_t, const trace_attr_t *restrict,
13848     trace_id_t *restrict);
13849 TRL int posix_trace_create_withlog(pid_t, const trace_attr_t *restrict,
13850     int, trace_id_t *restrict);
13851 void posix_trace_event(trace_event_id_t, const void *restrict, size_t);
```

```

13852     int  posix_trace_eventid_equal(trace_id_t, trace_event_id_t,
13853                                     trace_event_id_t);
13854     int  posix_trace_eventid_get_name(trace_id_t, trace_event_id_t, char *);
13855     int  posix_trace_eventid_open(const char *restrict,
13856                                     trace_event_id_t *restrict);
13857     int  posix_trace_eventtypelist_getnext_id(trace_id_t,
13858                                     trace_event_id_t *restrict, int *restrict);
13859     int  posix_trace_eventtypelist_rewind(trace_id_t);
13860 TEF    int  posix_trace_eventset_add(trace_event_id_t, trace_event_set_t *);
13861     int  posix_trace_eventset_del(trace_event_id_t, trace_event_set_t *);
13862     int  posix_trace_eventset_empty(trace_event_set_t *);
13863     int  posix_trace_eventset_fill(trace_event_set_t *, int);
13864     int  posix_trace_eventset_ismember(trace_event_id_t,
13865                                     const trace_event_set_t *restrict, int *restrict);
13866     int  posix_trace_flush(trace_id_t);
13867     int  posix_trace_get_attr(trace_id_t, trace_attr_t *);
13868 TEF    int  posix_trace_get_filter(trace_id_t, trace_event_set_t *);
13869     int  posix_trace_get_status(trace_id_t,
13870                                     struct posix_trace_status_info *);
13871     int  posix_trace_getnext_event(trace_id_t,
13872                                     struct posix_trace_event_info *restrict, void *restrict,
13873                                     size_t, size_t *restrict, int *restrict);
13874 TRL    int  posix_trace_open(int, trace_id_t *);
13875     int  posix_trace_rewind(trace_id_t);
13876 TEF    int  posix_trace_set_filter(trace_id_t, const trace_event_set_t *, int);
13877     int  posix_trace_shutdown(trace_id_t);
13878     int  posix_trace_start(trace_id_t);
13879     int  posix_trace_stop(trace_id_t);
13880 TMO    int  posix_trace_timedgetnext_event(trace_id_t,
13881                                     struct posix_trace_event_info *restrict, void *restrict,
13882                                     size_t, size_t *restrict, int *restrict,
13883                                     const struct timespec *restrict);
13884 TEF    int  posix_trace_trid_eventid_open(trace_id_t, const char *restrict,
13885                                     trace_event_id_t *restrict);
13886     int  posix_trace_trygetnext_event(trace_id_t,
13887                                     struct posix_trace_event_info *restrict, void *restrict, size_t,
13888                                     size_t *restrict, int *restrict);

```

**13889 APPLICATION USAGE**

13890 None.

**13891 RATIONALE**

13892 None.

**13893 FUTURE DIRECTIONS**

13894 None.

**13895 SEE ALSO**

13896 **<sys/types.h>**, the System Interfaces volume of IEEE Std 1003.1-200x, Section 2.11, Tracing, the  
13897 System Interfaces volume of IEEE Std 1003.1-200x, *posix\_trace\_attr\_destroy()*,  
13898 *posix\_trace\_attr\_getclockres()*, *posix\_trace\_attr\_getcreatetime()*, *posix\_trace\_attr\_getgenversion()*,  
13899 *posix\_trace\_attr\_getinherited()*, *posix\_trace\_attr\_getlogfullpolicy()*, *posix\_trace\_attr\_getlogsize()*,  
13900 *posix\_trace\_attr\_getmaxdatasize()*, *posix\_trace\_attr\_getmaxsystemeventsizesize()*,  
13901 *posix\_trace\_attr\_getmaxusereventsizesize()*, *posix\_trace\_attr\_getname()*,

13902 *posix\_trace\_attr\_getstreamfullpolicy()*, *posix\_trace\_attr\_getstreamsize()*, *posix\_trace\_attr\_init()*,  
13903 *posix\_trace\_attr\_setinherited()*, *posix\_trace\_attr\_setlogfullpolicy()*, *posix\_trace\_attr\_setlogsize()*,  
13904 *posix\_trace\_attr\_setmaxdatasize()*, *posix\_trace\_attr\_setname()*, *posix\_trace\_attr\_setstreamsize()*,  
13905 *posix\_trace\_attr\_setstreamfullpolicy()*, *posix\_trace\_clear()*, *posix\_trace\_close()*, *posix\_trace\_create()*,  
13906 *posix\_trace\_create\_withlog()*, *posix\_trace\_event()*, *posix\_trace\_eventid\_equal()*,  
13907 *posix\_trace\_eventid\_get\_name()*, *posix\_trace\_eventid\_open()*, *posix\_trace\_eventtypelist\_getnext\_id()*,  
13908 *posix\_trace\_eventtypelist\_rewind()*, *posix\_trace\_eventset\_add()*, *posix\_trace\_eventset\_del()*,  
13909 *posix\_trace\_eventset\_empty()*, *posix\_trace\_eventset\_fill()*, *posix\_trace\_eventset\_ismember()*,  
13910 *posix\_trace\_flush()*, *posix\_trace\_get\_attr()*, *posix\_trace\_get\_filter()*, *posix\_trace\_get\_status()*,  
13911 *posix\_trace\_getnext\_event()*, *posix\_trace\_open()*, *posix\_trace\_rewind()*, *posix\_trace\_set\_filter()*,  
13912 *posix\_trace\_shutdown()*, *posix\_trace\_start()*, *posix\_trace\_stop()*, *posix\_trace\_timedgetnext\_event()*,  
13913 *posix\_trace\_trid\_eventid\_open()*, *posix\_trace\_trygetnext\_event()*

13914 **CHANGE HISTORY**

13915 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

13916 **NAME**13917 `ucontext.h` — user context13918 **SYNOPSIS**13919 XSI `#include <ucontext.h>`

13920

13921 **DESCRIPTION**13922 The `<ucontext.h>` header shall define the `mcontext_t` type through **typedef**.13923 The `<ucontext.h>` header shall define the `ucontext_t` type as a structure that shall include at least  
13924 the following members:

13925	<code>ucontext_t *uc_link</code>	Pointer to the context that is resumed 13926 when this context returns.
13927	<code>sigset_t uc_sigmask</code>	The set of signals that are blocked when this 13928 context is active.
13929	<code>stack_t uc_stack</code>	The stack used by this context.
13930	<code>mcontext_t uc_mcontext</code>	A machine-specific representation of the saved 13931 context.

13932 The types `sigset_t` and `stack_t` shall be defined as in `<signal.h>`.13933 The following shall be declared as functions and may also be defined as macros, Function |  
13934 prototypes shall be provided. |

```
13935 int getcontext(ucontext_t *);  
13936 int setcontext(const ucontext_t *);  
13937 void makecontext(ucontext_t *, void (*)(void), int, ...);  
13938 int swapcontext(ucontext_t *restrict, const ucontext_t *restrict);
```

13939 **APPLICATION USAGE**

13940 None.

13941 **RATIONALE**

13942 None.

13943 **FUTURE DIRECTIONS**

13944 None.

13945 **SEE ALSO**13946 `<signal.h>`, the System Interfaces volume of IEEE Std 1003.1-200x, `getcontext()`, `makecontext()`,  
13947 `sigaction()`, `sigprocmask()`, `sigaltstack()`13948 **CHANGE HISTORY**

13949 First released in Issue 4, Version 2.

13950 **NAME**

13951 ulimit.h — ulimit commands

13952 **SYNOPSIS**

13953 XSI #include &lt;ulimit.h&gt;

13954

13955 **DESCRIPTION**13956 The <ulimit.h> header shall define the symbolic constants used by the *ulimit()* function.

13957 Symbolic constants:

13958 UL\_GETFSIZE Get maximum file size.

13959 UL\_SETFSIZE Set maximum file size.

13960 The following shall be declared as a function and may also be defined as a macro. A function |  
13961 prototype shall be provided. |

13962 long ulimit(int, ...);

13963 **APPLICATION USAGE**

13964 None.

13965 **RATIONALE**

13966 None.

13967 **FUTURE DIRECTIONS**

13968 None.

13969 **SEE ALSO**13970 The System Interfaces volume of IEEE Std 1003.1-200x, *ulimit()*13971 **CHANGE HISTORY**

13972 First released in Issue 3.

13973 **NAME**

13974         unistd.h — standard symbolic constants and types

13975 **SYNOPSIS**

13976         #include &lt;unistd.h&gt;

13977 **DESCRIPTION**

13978         The **<unistd.h>** header defines miscellaneous symbolic constants and types, and declares  
 13979         miscellaneous functions. The actual value of the constants are unspecified except as shown. The  
 13980         contents of this header are shown below.

13981         **Version Test Macros**

13982         The following symbolic constants shall be defined:

## 13983         \_POSIX\_VERSION

13984                 Integer value indicating version of IEEE Std 1003.1 (C-language binding) to which the  
 13985                 implementation conforms. For implementations conforming to IEEE Std 1003.1-200x, the  
 13986                 value shall be 200xxxL.

## 13987         \_POSIX2\_VERSION

13988                 Integer value indicating version of the Shell and Utilities volume of IEEE Std 1003.1 to  
 13989                 which the implementation conforms. For implementations conforming to  
 13990                 IEEE Std 1003.1-200x, the value shall be 200xxxL.

13991         The following symbolic constant shall be defined only if the implementation supports the XSI  
 13992         option; see Section 2.1.4 (on page 19).

## 13993 XSI         \_XOPEN\_VERSION

13994                 Integer value indicating version of the X/Open Portability Guide to which the  
 13995                 implementation conforms. The value shall be 600.

13996         **Constants for Options and Option Groups**

13997         The following symbolic constants, if defined in **<unistd.h>**, shall have a value of  $-1$ ,  $0$ , or greater,  
 13998         unless otherwise specified below. If these are undefined, the *fpathconf()*, *pathconf()*, or *sysconf()*  
 13999         functions can be used to determine whether the option is provided for a particular invocation of  
 14000         the application.

14001         If a symbolic constant is defined with the value  $-1$ , the option is not supported. Headers, data  
 14002         types, and function interfaces required only for the option need not be supplied. An application  
 14003         that attempts to use anything associated only with the option is considered to be requiring an  
 14004         extension.

14005         If a symbolic constant is defined with a value greater than zero, the option shall always be  
 14006         supported when the application is executed. All headers, data types, and functions shall be  
 14007         present and shall operate as specified.

14008         If a symbolic constant is defined with the value zero, all headers, data types, and functions shall  
 14009         be present. The application can check at runtime to see whether the option is supported by  
 14010         calling *fpathconf()*, *pathconf()*, or *sysconf()* with the indicated *name* parameter.

14011         Unless explicitly specified otherwise, the behavior of functions associated with an unsupported  
 14012         option is unspecified, and an application that uses such functions without first checking  
 14013         *fpathconf()*, *pathconf()*, or *sysconf()* is considered to be requiring an extension.

14014         For conformance requirements, refer to Chapter 2 (on page 15).

14015	ADV	<b>_POSIX_ADVISORY_INFO</b>	
14016		The implementation supports the Advisory Information option. If this symbol has a value	
14017		other than -1 or 0, it shall have the value 200xxxL.	
14018	AIO	<b>_POSIX_ASYNCHRONOUS_IO</b>	
14019		The implementation supports the Asynchronous Input and Output option. If this symbol	
14020		has a value other than -1 or 0, it shall have the value 200xxxL.	
14021	BAR	<b>_POSIX_BARRIERS</b>	
14022		The implementation supports the Barriers option. If this symbol has a value other than -1 or	
14023		0, it shall have the value 200xxxL.	
14024		<b>_POSIX_CHOWN_RESTRICTED</b>	
14025		The use of <i>chown()</i> and <i>fchown()</i> is restricted to a process with appropriate privileges, and	
14026		to changing the group ID of a file only to the effective group ID of the process or to one of	
14027		its supplementary group IDs. This symbol shall always be set to a value other than -1.	
14028	CS	<b>_POSIX_CLOCK_SELECTION</b>	
14029		The implementation supports the Clock Selection option. If this symbol has a value other	
14030		than -1 or 0, it shall have the value 200xxxL.	
14031	CPT	<b>_POSIX_CPUTIME</b>	
14032		The implementation supports the Process CPU-Time Clocks option. If this symbol has a	
14033		value other than -1 or 0, it shall have the value 200xxxL.	
14034	FSC	<b>_POSIX_FSYNC</b>	
14035		The implementation supports the File Synchronization option. If this symbol has a value	
14036		other than -1 or 0, it shall have the value 200xxxL.	
14037		<b>_POSIX_JOB_CONTROL</b>	
14038		The implementation supports job control. This symbol shall always be set to a value greater	
14039		than zero.	
14040	MF	<b>_POSIX_MAPPED_FILES</b>	
14041		The implementation supports the Memory Mapped Files option. If this symbol has a value	
14042		other than -1 or 0, it shall have the value 200xxxL.	
14043	ML	<b>_POSIX_MEMLOCK</b>	
14044		The implementation supports the Process Memory Locking option. If this symbol has a	
14045		value other than -1 or 0, it shall have the value 200xxxL.	
14046	MLR	<b>_POSIX_MEMLOCK_RANGE</b>	
14047		The implementation supports the Range Memory Locking option. If this symbol has a value	
14048		other than -1 or 0, it shall have the value 200xxxL.	
14049	MPR	<b>_POSIX_MEMORY_PROTECTION</b>	
14050		The implementation supports the Memory Protection option. If this symbol has a value	
14051		other than -1 or 0, it shall have the value 200xxxL.	
14052	MSG	<b>_POSIX_MESSAGE_PASSING</b>	
14053		The implementation supports the Message Passing option. If this symbol has a value other	
14054		than -1 or 0, it shall have the value 200xxxL.	
14055	MON	<b>_POSIX_MONOTONIC_CLOCK</b>	
14056		The implementation supports the Monotonic Clock option. If this symbol has a value other	
14057		than -1 or 0, it shall have the value 200xxxL.	
14058		<b>_POSIX_NO_TRUNC</b>	
14059		Pathname components longer than {NAME_MAX} generate an error. This symbol shall	

14060		always be set to a value other than -1.	
14061	PIO	<b>_POSIX_PRIORITIZED_IO</b>	
14062		The implementation supports the Prioritized Input and Output option. If this symbol has a	
14063		value other than -1 or 0, it shall have the value 200xxxL.	
14064	PS	<b>_POSIX_PRIORITY_SCHEDULING</b>	
14065		The implementation supports the Process Scheduling option. If this symbol has a value	
14066		other than -1 or 0, it shall have the value 200xxxL.	
14067	RS	<b>_POSIX_RAW_SOCKETS</b>	
14068		The implementation supports the Raw Sockets option. If this symbol has a value other than	
14069		-1 or 0, it shall have the value 200xxxL.	
14070	THR	<b>_POSIX_READER_WRITER_LOCKS</b>	
14071		The implementation supports the Read-Write Locks option. This is always set to a value	
14072		greater than zero if the Threads option is supported. If this symbol has a value other than -1	
14073		or 0, it shall have the value 200xxxL.	
14074	RTS	<b>_POSIX_REALTIME_SIGNALS</b>	
14075		The implementation supports the Realtime Signals Extension option. If this symbol has a	
14076		value other than -1 or 0, it shall have the value 200xxxL.	
14077		<b>_POSIX_REGEX</b>	
14078		The implementation supports the Regular Expression Handling option. This symbol shall	
14079		always be set to a value greater than zero.	
14080		<b>_POSIX_SAVED_IDS</b>	
14081		Each process has a saved set-user-ID and a saved set-group-ID. This symbol shall always	
14082		be set to a value greater than zero.	
14083	SEM	<b>_POSIX_SEMAPHORES</b>	
14084		The implementation supports the Semaphores option. If this symbol has a value other than	
14085		-1 or 0, it shall have the value 200xxxL.	
14086	SHM	<b>_POSIX_SHARED_MEMORY_OBJECTS</b>	
14087		The implementation supports the Shared Memory Objects option. If this symbol has a value	
14088		other than -1 or 0, it shall have the value 200xxxL.	
14089		<b>_POSIX_SHELL</b>	
14090		The implementation supports the POSIX shell. This symbol shall always be set to a value	
14091		greater than zero.	
14092	SPN	<b>_POSIX_SPAWN</b>	
14093		The implementation supports the Spawn option. If this symbol has a value other than -1 or	
14094		0, it shall have the value 200xxxL.	
14095	SPI	<b>_POSIX_SPIN_LOCKS</b>	
14096		The implementation supports the Spin Locks option. If this symbol has a value other than	
14097		-1 or 0, it shall have the value 200xxxL.	
14098	SS	<b>_POSIX_SPORADIC_SERVER</b>	
14099		The implementation supports the Process Sporadic Server option. If this symbol has a value	
14100		other than -1 or 0, it shall have the value 200xxxL.	
14101	SIO	<b>_POSIX_SYNCHRONIZED_IO</b>	
14102		The implementation supports the Synchronized Input and Output option. If this symbol	
14103		has a value other than -1 or 0, it shall have the value 200xxxL.	

14104	TSA	<b>_POSIX_THREAD_ATTR_STACKADDR</b>	
14105		The implementation supports the Thread Stack Address Attribute option. If this symbol	
14106		has a value other than -1 or 0, it shall have the value 200xxxL.	
14107	TSS	<b>_POSIX_THREAD_ATTR_STACKSIZE</b>	
14108		The implementation supports the Thread Stack Address Size option. If this symbol has a	
14109		value other than -1 or 0, it shall have the value 200xxxL.	
14110	TCT	<b>_POSIX_THREAD_CPUTIME</b>	
14111		The implementation supports the Thread CPU-Time Clocks option. If this symbol has a	
14112		value other than -1 or 0, it shall have the value 200xxxL.	
14113	TPI	<b>_POSIX_THREAD_PRIO_INHERIT</b>	
14114		The implementation supports the Thread Priority Inheritance option. If this symbol has a	
14115		value other than -1 or 0, it shall have the value 200xxxL.	
14116	TPP	<b>_POSIX_THREAD_PRIO_PROTECT</b>	
14117		The implementation supports the Thread Priority Protection option. If this symbol has a	
14118		value other than -1 or 0, it shall have the value 200xxxL.	
14119	TPS	<b>_POSIX_THREAD_PRIORITY_SCHEDULING</b>	
14120		The implementation supports the Thread Execution Scheduling option. If this symbol has a	
14121		value other than -1 or 0, it shall have the value 200xxxL.	
14122	TSH	<b>_POSIX_THREAD_PROCESS_SHARED</b>	
14123		The implementation supports the Thread Process-Shared Synchronization option. If this	
14124		symbol has a value other than -1 or 0, it shall have the value 200xxxL.	
14125	TSF	<b>_POSIX_THREAD_SAFE_FUNCTIONS</b>	
14126		The implementation supports the Thread-Safe Functions option. If this symbol has a value	
14127		other than -1 or 0, it shall have the value 200xxxL.	
14128	TSP	<b>_POSIX_THREAD_SPORADIC_SERVER</b>	
14129		The implementation supports the Thread Sporadic Server option. If this symbol has a value	
14130		other than -1 or 0, it shall have the value 200xxxL.	
14131	THR	<b>_POSIX_THREADS</b>	
14132		The implementation supports the Threads option. If this symbol has a value other than -1	
14133		or 0, it shall have the value 200xxxL.	
14134	TMO	<b>_POSIX_TIMEOUTS</b>	
14135		The implementation supports the Timeouts option. If this symbol has a value other than -1	
14136		or 0, it shall have the value 200xxxL.	
14137	TMR	<b>_POSIX_TIMERS</b>	
14138		The implementation supports the Timers option. If this symbol has a value other than -1 or	
14139		0, it shall have the value 200xxxL.	
14140	TRC	<b>_POSIX_TRACE</b>	
14141		The implementation supports the Trace option. If this symbol has a value other than -1 or 0,	
14142		it shall have the value 200xxxL.	
14143	TEF	<b>_POSIX_TRACE_EVENT_FILTER</b>	
14144		The implementation supports the Trace Event Filter option. If this symbol has a value other	
14145		than -1 or 0, it shall have the value 200xxxL.	
14146	TRI	<b>_POSIX_TRACE_INHERIT</b>	
14147		The implementation supports the Trace Inherit option. If this symbol has a value other than	
14148		-1 or 0, it shall have the value 200xxxL.	

14149	TRL	<b>_POSIX_TRACE_LOG</b>	
14150		The implementation supports the Trace Log option. If this symbol has a value other than -1	
14151		or 0, it shall have the value 200xxxL.	
14152	TYM	<b>_POSIX_TYPED_MEMORY_OBJECTS</b>	
14153		The implementation supports the Typed Memory Objects option. If this symbol has a value	
14154		other than -1 or 0, it shall have the value 200xxxL.	
14155		<b>_POSIX_VDISABLE</b>	
14156		This symbol shall be defined to be the value of a character that shall disable terminal special	
14157		character handling as described in <termios.h>. This symbol shall always be set to a value	
14158		other than -1.	
14159		<b>_POSIX2_C_BIND</b>	
14160		The implementation supports the C-Language Binding option. This symbol shall always	
14161		have the value 200xxxL.	
14162	CD	<b>_POSIX2_C_DEV</b>	
14163		The implementation supports the C-Language Development Utilities option. If this symbol	
14164		has a value other than -1 or 0, it shall have the value 200xxxL.	
14165		<b>_POSIX2_CHAR_TERM</b>	
14166		The implementation supports at least one terminal type.	
14167	FD	<b>_POSIX2_FORT_DEV</b>	
14168		The implementation supports the FORTRAN Development Utilities option. If this symbol	
14169		has a value other than -1 or 0, it shall have the value 200xxxL.	
14170	FR	<b>_POSIX2_FORT_RUN</b>	
14171		The implementation supports the FORTRAN Runtime Utilities option. If this symbol has a	
14172		value other than -1 or 0, it shall have the value 200xxxL.	
14173		<b>_POSIX2_LOCALEDEF</b>	
14174		The implementation supports the creation of locales by the <i>localedef</i> utility. If this symbol	
14175		has a value other than -1 or 0, it shall have the value 200xxxL.	
14176	BE	<b>_POSIX2_PBS</b>	
14177		The implementation supports the Batch Environment Services and Utilities option. If this	
14178		symbol has a value other than -1 or 0, it shall have the value 200xxxL.	
14179	BE	<b>_POSIX2_PBS_ACCOUNTING</b>	
14180		The implementation supports the Batch Accounting option. If this symbol has a value other	
14181		than -1 or 0, it shall have the value 200xxxL.	
14182	BE	<b>_POSIX2_PBS_CHECKPOINT</b>	
14183		The implementation supports the Batch Checkpoint/Restart option. If this symbol has a	
14184		value other than -1 or 0, it shall have the value 200xxxL.	
14185	BE	<b>_POSIX2_PBS_LOCATE</b>	
14186		The implementation supports the Locate Batch Job Request option. If this symbol has a	
14187		value other than -1 or 0, it shall have the value 200xxxL.	
14188	BE	<b>_POSIX2_PBS_MESSAGE</b>	
14189		The implementation supports the Batch Job Message Request option. If this symbol has a	
14190		value other than -1 or 0, it shall have the value 200xxxL.	
14191	BE	<b>_POSIX2_PBS_TRACK</b>	
14192		The implementation supports the Track Batch Job Request option. If this symbol has a value	
14193		other than -1 or 0, it shall have the value 200xxxL.	

14194 SD	<b>_POSIX2_SW_DEV</b>	
14195		The implementation supports the Software Development Utilities option. If this symbol has a value other than <code>-1</code> or <code>0</code> , it shall have the value <code>200xxxL</code> .
14196		
14197 UP	<b>_POSIX2_UPE</b>	
14198		The implementation supports the User Portability Utilities option. If this symbol has a value other than <code>-1</code> or <code>0</code> , it shall have the value <code>200xxxL</code> .
14199		
14200	<b>_V6_ILP32_OFF32</b>	
14201		The implementation provides a C-language compilation environment with 32-bit <b>int</b> , <b>long</b> , <b>pointer</b> , and <b>off_t</b> types.
14202		
14203	<b>_V6_ILP32_OFFBIG</b>	
14204		The implementation provides a C-language compilation environment with 32-bit <b>int</b> , <b>long</b> , and <b>pointer</b> types and an <b>off_t</b> type using at least 64 bits.
14205		
14206	<b>_V6_LP64_OFF64</b>	
14207		The implementation provides a C-language compilation environment with 32-bit <b>int</b> and 64-bit <b>long</b> , <b>pointer</b> , and <b>off_t</b> types.
14208		
14209	<b>_V6_LPBIG_OFFBIG</b>	
14210		The implementation provides a C-language compilation environment with an <b>int</b> type using at least 32 bits and <b>long</b> , <b>pointer</b> , and <b>off_t</b> types using at least 64 bits.
14211		
14212 XSI	<b>_XBS5_ILP32_OFF32 (LEGACY)</b>	
14213		The implementation provides a C-language compilation environment with 32-bit <b>int</b> , <b>long</b> , <b>pointer</b> , and <b>off_t</b> types.
14214		
14215 XSI	<b>_XBS5_ILP32_OFFBIG (LEGACY)</b>	
14216		The implementation provides a C-language compilation environment with 32-bit <b>int</b> , <b>long</b> , and <b>pointer</b> types and an <b>off_t</b> type using at least 64 bits.
14217		
14218 XSI	<b>_XBS5_LP64_OFF64 (LEGACY)</b>	
14219		The implementation provides a C-language compilation environment with 32-bit <b>int</b> and 64-bit <b>long</b> , <b>pointer</b> , and <b>off_t</b> types.
14220		
14221 XSI	<b>_XBS5_LPBIG_OFFBIG (LEGACY)</b>	
14222		The implementation provides a C-language compilation environment with an <b>int</b> type using at least 32 bits and <b>long</b> , <b>pointer</b> , and <b>off_t</b> types using at least 64 bits.
14223		
14224 XSI	<b>_XOPEN_CRYPT</b>	
14225		The implementation supports the X/Open Encryption Option Group.
14226	<b>_XOPEN_ENH_I18N</b>	
14227		The implementation supports the Issue 4, Version 2 Enhanced Internationalization Option Group. This symbol shall always be set to a value other than <code>-1</code> .
14228		
14229	<b>_XOPEN_LEGACY</b>	
14230		The implementation supports the Legacy Option Group.
14231	<b>_XOPEN_REALTIME</b>	
14232		The implementation supports the X/Open Realtime Option Group.
14233	<b>_XOPEN_REALTIME_THREADS</b>	
14234		The implementation supports the X/Open Realtime Threads Option Group.
14235	<b>_XOPEN_SHM</b>	
14236		The implementation supports the Issue 4, Version 2 Shared Memory Option Group. This symbol shall always be set to a value other than <code>-1</code> .
14237		

14238 **XOPEN\_STREAMS**  
 14239 The implementation supports the XSI STREAMS Option Group. |

14240 XSI **XOPEN\_UNIX**  
 14241 The implementation supports the XSI extension. |

14242 **Execution-Time Symbolic Constants** |

14243 If any of the following constants are not defined in the <unistd.h> header, the value shall vary  
 14244 depending on the file to which it is applied.

14245 If any of the following constants are defined to have value -1 in the <unistd.h> header, the  
 14246 implementation shall not provide the option on any file; if any are defined to have a value other  
 14247 than -1 in the <unistd.h> header, the implementation shall provide the option on all applicable  
 14248 files.

14249 All of the following constants, whether defined in <unistd.h> or not, may be queried with  
 14250 respect to a specific file using the *pathconf()* or *fpathconf()* functions:

14251 **\_POSIX\_ASYNC\_IO**  
 14252 Asynchronous input or output operations may be performed for the associated file.

14253 **\_POSIX\_PRIO\_IO**  
 14254 Prioritized input or output operations may be performed for the associated file.

14255 **\_POSIX\_SYNC\_IO**  
 14256 Synchronized input or output operations may be performed for the associated file.

14257 **Constants for Functions**

14258 The following symbolic constant shall be defined:

14259 **NULL** Null pointer

14260 The following symbolic constants shall be defined for the *access()* function:

14261 **F\_OK** Test for existence of file.

14262 **R\_OK** Test for read permission.

14263 **W\_OK** Test for write permission.

14264 **X\_OK** Test for execute (search) permission.

14265 The constants **F\_OK**, **R\_OK**, **W\_OK**, and **X\_OK** and the expressions *R\_OK | W\_OK*, *R\_OK | X\_OK*,  
 14266 and *R\_OK | W\_OK | X\_OK* shall all have distinct values.

14267 The following symbolic constants shall be defined for the *confstr()* function:

14268 **\_CS\_POSIX\_PATH**  
 14269 This is the value for the *PATH* environment variable that finds all standard utilities. |

14270 **\_CS\_POSIX\_V6\_ILP32\_OFF32\_CFLAGS**  
 14271 If *sysconf(\_SC\_V6\_ILP32\_OFF32)* returns -1, the meaning of this value is unspecified.  
 14272 Otherwise, this value is the set of initial options to be given to the *cc* and *c99* utilities to  
 14273 build an application using a programming model with 32-bit **int**, **long**, **pointer**, and **off\_t**  
 14274 types. |

14275 **\_CS\_POSIX\_V6\_ILP32\_OFF32\_LDFLAGS**  
 14276 If *sysconf(\_SC\_V6\_ILP32\_OFF32)* returns -1, the meaning of this value is unspecified.  
 14277 Otherwise, this value is the set of final options to be given to the *cc* and *c99* utilities to build  
 14278 an application using a programming model with 32-bit **int**, **long**, **pointer**, and **off\_t** types. |

14279 `_CS_POSIX_V6_ILP32_OFF32_LIBS` |  
 14280 If `sysconf(_SC_V6_ILP32_OFF32)` returns `-1`, the meaning of this value is unspecified. |  
 14281 Otherwise, this value is the set of libraries to be given to the `cc` and `c99` utilities to build an |  
 14282 application using a programming model with 32-bit **int**, **long**, **pointer**, and **off\_t** types. |

14283 `_CS_POSIX_V6_ILP32_OFF32_LINTFLAGS` |  
 14284 If `sysconf(_SC_V6_ILP32_OFF32)` returns `-1`, the meaning of this value is unspecified. |  
 14285 Otherwise, this value is the set of options to be given to the `lint` utility to check application |  
 14286 source using a programming model with 32-bit **int**, **long**, **pointer**, and **off\_t** types. |

14287 `_CS_POSIX_V6_ILP32_OFFBIG_CFLAGS` |  
 14288 If `sysconf(_SC_V6_ILP32_OFFBIG)` returns `-1`, the meaning of this value is unspecified. |  
 14289 Otherwise, this value is the set of initial options to be given to the `cc` and `c99` utilities to |  
 14290 build an application using a programming model with 32-bit **int**, **long**, and **pointer** types, |  
 14291 and an **off\_t** type using at least 64 bits. |

14292 `_CS_POSIX_V6_ILP32_OFFBIG_LDFLAGS` |  
 14293 If `sysconf(_SC_V6_ILP32_OFFBIG)` returns `-1`, the meaning of this value is unspecified. |  
 14294 Otherwise, this value is the set of final options to be given to the `cc` and `c99` utilities to build |  
 14295 an application using a programming model with 32-bit **int**, **long**, and **pointer** types, and an |  
 14296 **off\_t** type using at least 64 bits. |

14297 `_CS_POSIX_V6_ILP32_OFFBIG_LIBS` |  
 14298 If `sysconf(_SC_V6_ILP32_OFFBIG)` returns `-1`, the meaning of this value is unspecified. |  
 14299 Otherwise, this value is the set of libraries to be given to the `cc` and `c99` utilities to build an |  
 14300 application using a programming model with 32-bit **int**, **long**, and **pointer** types, and an |  
 14301 **off\_t** type using at least 64 bits. |

14302 `_CS_POSIX_V6_ILP32_OFFBIG_LINTFLAGS` |  
 14303 If `sysconf(_SC_V6_ILP32_OFFBIG)` returns `-1`, the meaning of this value is unspecified. |  
 14304 Otherwise, this value is the set of options to be given to the `lint` utility to check an |  
 14305 application using a programming model with 32-bit **int**, **long**, and **pointer** types, and an |  
 14306 **off\_t** type using at least 64 bits. |

14307 `_CS_POSIX_V6_LP64_OFF64_CFLAGS` |  
 14308 If `sysconf(_SC_V6_LP64_OFF64)` returns `-1`, the meaning of this value is unspecified. |  
 14309 Otherwise, this value is the set of initial options to be given to the `cc` and `c99` utilities to |  
 14310 build an application using a programming model with 64-bit **int**, **long**, **pointer**, and **off\_t** |  
 14311 types. |

14312 `_CS_POSIX_V6_LP64_OFF64_LDFLAGS` |  
 14313 If `sysconf(_SC_V6_LP64_OFF64)` returns `-1`, the meaning of this value is unspecified. |  
 14314 Otherwise, this value is the set of final options to be given to the `cc` and `c99` utilities to build |  
 14315 an application using a programming model with 64-bit **int**, **long**, **pointer**, and **off\_t** types. |

14316 `_CS_POSIX_V6_LP64_OFF64_LIBS` |  
 14317 If `sysconf(_SC_V6_LP64_OFF64)` returns `-1`, the meaning of this value is unspecified. |  
 14318 Otherwise, this value is the set of libraries to be given to the `cc` and `c99` utilities to build an |  
 14319 application using a programming model with 64-bit **int**, **long**, **pointer**, and **off\_t** types. |

14320 `_CS_POSIX_V6_LP64_OFF64_LINTFLAGS` |  
 14321 If `sysconf(_SC_V6_LP64_OFF64)` returns `-1`, the meaning of this value is unspecified. |  
 14322 Otherwise, this value is the set of options to be given to the `lint` utility to check application |  
 14323 source using a programming model with 64-bit **int**, **long**, **pointer**, and **off\_t** types. |

14324 `_CS_POSIX_V6_LPBIG_OFFBIG_CFLAGS` |  
 14325 If `sysconf(_SC_V6_LPBIG_OFFBIG)` returns `-1`, the meaning of this value is unspecified. |

14326           Otherwise, this value is the set of initial options to be given to the *cc* and *c99* utilities to  
 14327           build an application using a programming model with an **int** type using at least 32 bits and  
 14328           **long**, **pointer**, and **off\_t** types using at least 64 bits.

14329           **\_CS\_POSIX\_V6\_LPBIG\_OFFBIG\_LDFLAGS**  
 14330           If *sysconf*(*\_SC\_V6\_LPBIG\_OFFBIG*) returns -1, the meaning of this value is unspecified.  
 14331           Otherwise, this value is the set of final options to be given to the *cc* and *c99* utilities to build  
 14332           an application using a programming model with an **int** type using at least 32 bits and **long**,  
 14333           **pointer**, and **off\_t** types using at least 64 bits.

14334           **\_CS\_POSIX\_V6\_LPBIG\_OFFBIG\_LIBS**  
 14335           If *sysconf*(*\_SC\_V6\_LPBIG\_OFFBIG*) returns -1, the meaning of this value is unspecified.  
 14336           Otherwise, this value is the set of libraries to be given to the *cc* and *c99* utilities to build an  
 14337           application using a programming model with an **int** type using at least 32 bits and **long**,  
 14338           **pointer**, and **off\_t** types using at least 64 bits.

14339           **\_CS\_POSIX\_V6\_LPBIG\_OFFBIG\_LINTFLAGS**  
 14340           If *sysconf*(*\_SC\_V6\_LPBIG\_OFFBIG*) returns -1, the meaning of this value is unspecified.  
 14341           Otherwise, this value is the set of options to be given to the *lint* utility to check application  
 14342           source using a programming model with an **int** type using at least 32 bits and **long**, **pointer**,  
 14343           and **off\_t** types using at least 64 bits.

14344           **\_CS\_V6\_WIDTH\_RESTRICTED\_ENVS**  
 14345           This value is a <newline>-separated list of names of programming environments supported  
 14346           by the implementation in which the widths of the **blksize\_t**, **cc\_t**, **mode\_t**, **nfds\_t**, **pid\_t**,  
 14347           **ptrdiff\_t**, **size\_t**, **speed\_t**, **ssize\_t**, **suseconds\_t**, **tcflag\_t**, **useconds\_t**, **wchar\_t**, and **wint\_t**  
 14348           types are no greater than the width of type **long**.

14349 XSI       **\_CS\_XBS5\_ILP32\_OFF32\_CFLAGS (LEGACY)**  
 14350           If *sysconf*(*\_SC\_XBS5\_ILP32\_OFF32*) returns -1, the meaning of this value is unspecified.  
 14351           Otherwise, this value is the set of initial options to be given to the *cc* and *c99* utilities to  
 14352           build an application using a programming model with 32-bit **int**, **long**, **pointer**, and **off\_t**  
 14353           types.

14354 XSI       **\_CS\_XBS5\_ILP32\_OFF32\_LDFLAGS (LEGACY)**  
 14355           If *sysconf*(*\_SC\_XBS5\_ILP32\_OFF32*) returns -1, the meaning of this value is unspecified.  
 14356           Otherwise, this value is the set of final options to be given to the *cc* and *c99* utilities to build  
 14357           an application using a programming model with 32-bit **int**, **long**, **pointer**, and **off\_t** types.

14358 XSI       **\_CS\_XBS5\_ILP32\_OFF32\_LIBS (LEGACY)**  
 14359           If *sysconf*(*\_SC\_XBS5\_ILP32\_OFF32*) returns -1, the meaning of this value is unspecified.  
 14360           Otherwise, this value is the set of libraries to be given to the *cc* and *c99* utilities to build an  
 14361           application using a programming model with 32-bit **int**, **long**, **pointer**, and **off\_t** types.

14362 XSI       **\_CS\_XBS5\_ILP32\_OFF32\_LINTFLAGS (LEGACY)**  
 14363           If *sysconf*(*\_SC\_XBS5\_ILP32\_OFF32*) returns -1, the meaning of this value is unspecified.  
 14364           Otherwise, this value is the set of options to be given to the *lint* utility to check application  
 14365           source using a programming model with 32-bit **int**, **long**, **pointer**, and **off\_t** types.

14366 XSI       **\_CS\_XBS5\_ILP32\_OFFBIG\_CFLAGS (LEGACY)**  
 14367           If *sysconf*(*\_SC\_XBS5\_ILP32\_OFFBIG*) returns -1, the meaning of this value is unspecified.  
 14368           Otherwise, this value is the set of initial options to be given to the *cc* and *c99* utilities to  
 14369           build an application using a programming model with 32-bit **int**, **long**, and **pointer** types,  
 14370           and an **off\_t** type using at least 64 bits.

14371 XSI       **\_CS\_XBS5\_ILP32\_OFFBIG\_LDFLAGS (LEGACY)**  
 14372           If *sysconf*(*\_SC\_XBS5\_ILP32\_OFFBIG*) returns -1, the meaning of this value is unspecified.

14373		Otherwise, this value is the set of final options to be given to the <i>cc</i> and <i>c99</i> utilities to build an application using a programming model with 32-bit <b>int</b> , <b>long</b> , and <b>pointer</b> types, and an <b>off_t</b> type using at least 64 bits.
14374		
14375		
14376	XSI	<b>_CS_XBS5_ILP32_OFFBIG_LIBS (LEGACY)</b>
14377		If <i>sysconf</i> ( <i>_SC_XBS5_ILP32_OFFBIG</i> ) returns $-1$ , the meaning of this value is unspecified.
14378		Otherwise, this value is the set of libraries to be given to the <i>cc</i> and <i>c99</i> utilities to build an application using a programming model with 32-bit <b>int</b> , <b>long</b> , and <b>pointer</b> types, and an <b>off_t</b> type using at least 64 bits.
14379		
14380		
14381	XSI	<b>_CS_XBS5_ILP32_OFFBIG_LINTFLAGS (LEGACY)</b>
14382		If <i>sysconf</i> ( <i>_SC_XBS5_ILP32_OFFBIG</i> ) returns $-1$ , the meaning of this value is unspecified.
14383		Otherwise, this value is the set of options to be given to the <i>lint</i> utility to check an application using a programming model with 32-bit <b>int</b> , <b>long</b> , and <b>pointer</b> types, and an <b>off_t</b> type using at least 64 bits.
14384		
14385		
14386	XSI	<b>_CS_XBS5_LP64_OFF64_CFLAGS (LEGACY)</b>
14387		If <i>sysconf</i> ( <i>_SC_XBS5_LP64_OFF64</i> ) returns $-1$ , the meaning of this value is unspecified.
14388		Otherwise, this value is the set of initial options to be given to the <i>cc</i> and <i>c99</i> utilities to build an application using a programming model with 64-bit <b>int</b> , <b>long</b> , <b>pointer</b> , and <b>off_t</b> types.
14389		
14390		
14391	XSI	<b>_CS_XBS5_LP64_OFF64_LDFLAGS (LEGACY)</b>
14392		If <i>sysconf</i> ( <i>_SC_XBS5_LP64_OFF64</i> ) returns $-1$ , the meaning of this value is unspecified.
14393		Otherwise, this value is the set of final options to be given to the <i>cc</i> and <i>c99</i> utilities to build an application using a programming model with 64-bit <b>int</b> , <b>long</b> , <b>pointer</b> , and <b>off_t</b> types.
14394		
14395	XSI	<b>_CS_XBS5_LP64_OFF64_LIBS (LEGACY)</b>
14396		If <i>sysconf</i> ( <i>_SC_XBS5_LP64_OFF64</i> ) returns $-1$ , the meaning of this value is unspecified.
14397		Otherwise, this value is the set of libraries to be given to the <i>cc</i> and <i>c99</i> utilities to build an application using a programming model with 64-bit <b>int</b> , <b>long</b> , <b>pointer</b> , and <b>off_t</b> types.
14398		
14399	XSI	<b>_CS_XBS5_LP64_OFF64_LINTFLAGS (LEGACY)</b>
14400		If <i>sysconf</i> ( <i>_SC_XBS5_LP64_OFF64</i> ) returns $-1$ , the meaning of this value is unspecified.
14401		Otherwise, this value is the set of options to be given to the <i>lint</i> utility to check application source using a programming model with 64-bit <b>int</b> , <b>long</b> , <b>pointer</b> , and <b>off_t</b> types.
14402		
14403	XSI	<b>_CS_XBS5_LPBIG_OFFBIG_CFLAGS (LEGACY)</b>
14404		If <i>sysconf</i> ( <i>_SC_XBS5_LPBIG_OFFBIG</i> ) returns $-1$ , the meaning of this value is unspecified.
14405		Otherwise, this value is the set of initial options to be given to the <i>cc</i> and <i>c99</i> utilities to build an application using a programming model with an <b>int</b> type using at least 32 bits and <b>long</b> , <b>pointer</b> , and <b>off_t</b> types using at least 64 bits.
14406		
14407		
14408	XSI	<b>_CS_XBS5_LPBIG_OFFBIG_LDFLAGS (LEGACY)</b>
14409		If <i>sysconf</i> ( <i>_SC_XBS5_LPBIG_OFFBIG</i> ) returns $-1$ , the meaning of this value is unspecified.
14410		Otherwise, this value is the set of final options to be given to the <i>cc</i> and <i>c99</i> utilities to build an application using a programming model with an <b>int</b> type using at least 32 bits and <b>long</b> , <b>pointer</b> , and <b>off_t</b> types using at least 64 bits.
14411		
14412		
14413	XSI	<b>_CS_XBS5_LPBIG_OFFBIG_LIBS (LEGACY)</b>
14414		If <i>sysconf</i> ( <i>_SC_XBS5_LPBIG_OFFBIG</i> ) returns $-1$ , the meaning of this value is unspecified.
14415		Otherwise, this value is the set of libraries to be given to the <i>cc</i> and <i>c99</i> utilities to build an application using a programming model with an <b>int</b> type using at least 32 bits and <b>long</b> , <b>pointer</b> , and <b>off_t</b> types using at least 64 bits.
14416		
14417		
14418	XSI	<b>_CS_XBS5_LPBIG_OFFBIG_LINTFLAGS (LEGACY)</b>
14419		If <i>sysconf</i> ( <i>_SC_XBS5_LPBIG_OFFBIG</i> ) returns $-1$ , the meaning of this value is unspecified.

14420           Otherwise, this value is the set of options to be given to the *lint* utility to check application  
 14421           source using a programming model with an **int** type using at least 32 bits and **long**, **pointer**,  
 14422           and **off\_t** types using at least 64 bits.

14423           The following symbolic constants shall be defined for the *lseek()* and *fcntl()* functions and shall  
 14424           have distinct values:

14425           SEEK\_CUR        Set file offset to current plus *offset*.

14426           SEEK\_END       Set file offset to EOF plus *offset*.

14427           SEEK\_SET       Set file offset to *offset*.

14428           The following symbolic constants shall be defined as possible values for the *function* argument  
 14429           to the *lockf()* function:

14430           F\_LOCK         Lock a section for exclusive use.

14431           F\_TEST         Test section for locks by other processes.

14432           F\_TLOCK        Test and lock a section for exclusive use.

14433           F\_ULOCK       Unlock locked sections.

14434           The following symbolic constants shall be defined for *pathconf()*:

14435 ADV       \_PC\_ALLOC\_SIZE\_MIN

14436 AIO       \_PC\_ASYNC\_IO

14437       \_PC\_CHOWN\_RESTRICTED

14438       \_PC\_FILESIZEBITS

14439       \_PC\_LINK\_MAX

14440       \_PC\_MAX\_CANON

14441       \_PC\_MAX\_INPUT

14442       \_PC\_NAME\_MAX

14443       \_PC\_NO\_TRUNC

14444       \_PC\_PATH\_MAX

14445       \_PC\_PIPE\_BUF

14446       \_PC\_PRIO\_IO

14447 ADV     \_PC\_REC\_INCR\_XFER\_SIZE

14448       \_PC\_REC\_MAX\_XFER\_SIZE

14449       \_PC\_REC\_MIN\_XFER\_SIZE

14450       \_PC\_REC\_XFER\_ALIGN

14451       \_PC\_SYNC\_IO

14452       \_PC\_VDISABLE

14453           The following symbolic constants shall be defined for *sysconf()*:

14454       \_SC\_2\_C\_BIND

14455       \_SC\_2\_C\_DEV

14456       \_SC\_2\_C\_VERSION

14457       \_SC\_2\_CHAR\_TIME

14458       \_SC\_2\_FORT\_DEV

14459       \_SC\_2\_FORT\_RUN

14460       \_SC\_2\_LOCALEDEF

14461       \_SC\_2\_PBS

14462       \_SC\_2\_PBS\_ACCOUNTING

14463       \_SC\_2\_PBS\_CHECKPOINT

14464       \_SC\_2\_PBS\_LOCATE

14465       \_SC\_2\_PBS\_MESSAGE

14466		_SC_2_PBS_TRACK	
14467		_SC_2_SW_DEV	
14468		_SC_2_UPE	
14469		_SC_2_VERSION	
14470		_SC_ADVISORY_INFO	
14471		_SC_ARG_MAX	
14472		_SC_AIO_LISTIO_MAX	
14473		_SC_AIO_MAX	
14474		_SC_AIO_PRIO_DELTA_MAX	
14475		_SC_ASYNCHRONOUS_IO	
14476	XSI	_SC_ATEXIT_MAX	
14477	BAR	_SC_BARRIERS	
14478		_SC_BASE	
14479		_SC_BC_BASE_MAX	
14480		_SC_BC_DIM_MAX	
14481		_SC_BC_SCALE_MAX	
14482		_SC_BC_STRING_MAX	
14483		_SC_C_LANG_SUPPORT	
14484		_SC_C_LANG_SUPPORT_R	
14485		_SC_CHILD_MAX	
14486		_SC_CLK_TCK	
14487	CS	_SC_CLOCK_SELECTION	
14488		_SC_COLL_WEIGHTS_MAX	
14489		_SC_CPUTIME	
14490		_SC_DELAYTIMER_MAX	
14491		_SC_DEVICE_IO	
14492		_SC_DEVICE_SPECIFIC	
14493		_SC_DEVICE_SPECIFIC_R	
14494		_SC_EXPR_NEST_MAX	
14495		_SC_FD_MGMT	
14496		_SC_FIFO	
14497		_SC_FILE_ATTRIBUTES	
14498		_SC_FILE_LOCKING	
14499		_SC_FILE_SYSTEM	
14500		_SC_FSYNC	
14501		_SC_GETGR_R_SIZE_MAX	
14502		_SC_GETPW_R_SIZE_MAX	
14503		_SC_HOST_NAME_MAX	
14504	XSI	_SC_IOV_MAX	
14505		_SC_JOB_CONTROL	
14506		_SC_LINE_MAX	
14507		_SC_LOGIN_NAME_MAX	
14508		_SC_MAPPED_FILES	
14509		_SC_MEMLOCK	
14510		_SC_MEMLOCK_RANGE	
14511		_SC_MEMORY_PROTECTION	
14512		_SC_MESSAGE_PASSING	
14513	MON	_SC_MONOTONIC_CLOCK	
14514		_SC_MQ_OPEN_MAX	
14515		_SC_MQ_PRIO_MAX	
14516		_SC_MULTIPLE_PROCESS	
14517		_SC_NETWORKING	

14518		_SC_NGROUPS_MAX	
14519		_SC_OPEN_MAX	
14520	XSI	_SC_PAGE_SIZE	
14521		_SC_PAGESIZE	
14522		_SC_PIPE	
14523		_SC_PRIORITIZED_IO	
14524		_SC_PRIORITY_SCHEDULING	
14525		_SC_RE_DUP_MAX	
14526	THR	_SC_READER_WRITER_LOCKS	
14527		_SC_REALTIME_SIGNALS	
14528		_SC_REGEX	
14529		_SC_RTSIG_MAX	
14530		_SC_SAVED_IDS	
14531		_SC_SEMAPHORES	
14532		_SC_SEM_NSEMS_MAX	
14533		_SC_SEM_VALUE_MAX	
14534		_SC_SHARED_MEMORY_OBJECTS	
14535		_SC_SHELL	
14536		_SC_SIGNALS	
14537		_SC_SIGQUEUE_MAX	
14538		_SC_SINGLE_PROCESS	
14539		_SC_SPAWN	
14540	SPI	_SC_SPIN_LOCKS	
14541		_SC_SPORADIC_SERVER	
14542		_SC_STREAM_MAX	
14543		_SC_SYNCHRONIZED_IO	
14544		_SC_SYSTEM_DATABASE	
14545		_SC_SYSTEM_DATABASE_R	
14546		_SC_THREAD_ATTR_STACKADDR	
14547		_SC_THREAD_ATTR_STACKSIZE	
14548		_SC_THREAD_CPUTIME	
14549		_SC_THREAD_DESTRUCTOR_ITERATIONS	
14550		_SC_THREAD_KEYS_MAX	
14551		_SC_THREAD_PRIO_INHERIT	
14552		_SC_THREAD_PRIO_PROTECT	
14553		_SC_THREAD_PRIORITY_SCHEDULING	
14554		_SC_THREAD_PROCESS_SHARED	
14555		_SC_THREAD_SAFE_FUNCTIONS	
14556		_SC_THREAD_SPARADIC_SERVER	
14557		_SC_THREAD_STACK_MIN	
14558		_SC_THREAD_THREADS_MAX	
14559		_SC_TIMEOUTS	
14560		_SC_THREADS	
14561		_SC_TIMER_MAX	
14562		_SC_TIMERS	
14563	TRC	_SC_TRACE	
14564	TEF	_SC_TRACE_EVENT_FILTER	
14565	TRI	_SC_TRACE_INHERIT	
14566	TRL	_SC_TRACE_LOG	
14567		_SC_TTY_NAME_MAX	
14568	TYM	_SC_TYPED_MEMORY_OBJECTS	
14569		_SC_TZNAME_MAX	

```

14570     _SC_USER_GROUPS
14571     _SC_USER_GROUPS_R
14572     _SC_V6_ILP32_OFF32
14573     _SC_V6_ILP32_OFFBIG
14574     _SC_V6_LP64_OFF64
14575     _SC_V6_LP64_OFFBIG
14576     _SC_VERSION
14577 XSI   _SC_XBS5_ILP32_OFF32 (LEGACY)
14578     _SC_XBS5_ILP32_OFFBIG (LEGACY)
14579     _SC_XBS5_LP64_OFF64 (LEGACY)
14580     _SC_XBS5_LP64_OFFBIG (LEGACY)
14581     _SC_XOPEN_CRYPT
14582     _SC_XOPEN_ENH_I18N
14583     _SC_XOPEN_LEGACY
14584     _SC_XOPEN_REALTIME
14585     _SC_XOPEN_REALTIME_THREADS
14586     _SC_XOPEN_SHM
14587     _SC_XOPEN_STREAMS
14588     _SC_XOPEN_UNIX
14589     _SC_XOPEN_VERSION
14590     _SC_XOPEN_XCU_VERSION
14591

```

14592 The two constants `_SC_PAGESIZE` and `_SC_PAGE_SIZE` may be defined to have the same  
14593 value.

14594 The following symbolic constants shall be defined for file streams:

```

14595     STDERR_FILENO    File number of stderr; 2.
14596     STDIN_FILENO     File number of stdin; 0.
14597     STDOUT_FILENO    File number of stdout; 1.

```

### 14598 Type Definitions

14599 The `size_t`, `ssize_t`, `uid_t`, `gid_t`, `off_t`, and `pid_t` types shall be defined as described in  
14600 <sys/types.h>.

14601 The `useconds_t` type shall be defined as described in <sys/types.h>.

14602 The `intptr_t` type shall be defined as described in <inttypes.h>.

### 14603 Declarations

14604 The following shall be declared as functions and may also be defined as macros. Function  
14605 prototypes shall be provided.

```

14606     int             access(const char *, int);
14607     unsigned        alarm(unsigned);
14608     int             chdir(const char *);
14609     int             chown(const char *, uid_t, gid_t);
14610     int             close(int);
14611     size_t          confstr(int, char *, size_t);
14612 XSI   char          *crypt(const char *, const char *);
14613     char           *ctermid(char *);
14614     int             dup(int);

```

```
14615     int          dup2(int, int);
14616 XSI   void          encrypt(char[64], int);
14617     int          execl(const char *, const char *, ...);
14618     int          execlp(const char *, const char *, ...);
14619     int          execlp(const char *, const char *, ...);
14620     int          execv(const char *, char *const []);
14621     int          execve(const char *, char *const [], char *const []);
14622     int          execvp(const char *, char *const []);
14623     void          _exit(int);
14624     int          fchown(int, uid_t, gid_t);
14625 XSI   int          fchdir(int);
14626 SIO   int          fdatsync(int);
14627     pid_t        fork(void);
14628     long         fpathconf(int, int);
14629     int          fsync(int);
14630     int          ftruncate(int, off_t);
14631     char         *getcwd(char *, size_t);
14632     gid_t        getegid(void);
14633     uid_t        geteuid(void);
14634     gid_t        getgid(void);
14635     int          getgroups(int, gid_t []);
14636 XSI   long         gethostid(void);
14637     int          gethostname(char *, size_t);
14638     char         *getlogin(void);
14639     int          getlogin_r(char *, size_t);
14640     int          getopt(int, char * const [], const char *);
14641 XSI   pid_t        getpgid(pid_t);
14642     pid_t        getpgrp(void);
14643     pid_t        getpid(void);
14644     pid_t        getppid(void);
14645 XSI   pid_t        getsid(pid_t);
14646     uid_t        getuid(void);
14647 XSI   char         *getwd(char *); (LEGACY)
14648     int          isatty(int);
14649 XSI   int          lchown(const char *, uid_t, gid_t);
14650     int          link(const char *, const char *);
14651 XSI   int          lockf(int, int, off_t);
14652     off_t        lseek(int, off_t, int);
14653 XSI   int          nice(int);
14654     long         pathconf(const char *, int);
14655     int          pause(void);
14656     int          pipe(int [2]);
14657 XSI   ssize_t       pread(int, void *, size_t, off_t);
14658     ssize_t      pwrite(int, const void *, size_t, off_t);
14659     ssize_t      read(int, void *, size_t);
14660     ssize_t      readlink(const char *restrict, char *restrict, size_t);
14661     int          rmdir(const char *);
14662     int          setegid(gid_t);
14663     int          seteuid(uid_t);
14664     int          setgid(gid_t);
```

```

14665     int          setpgid(pid_t, pid_t);
14666 XSI   pid_t          setpgrp(void);
14667     int          setregid(gid_t, gid_t);
14668     int          setreuid(uid_t, uid_t);
14669     pid_t        setsid(void);
14670     int          setuid(uid_t);
14671     unsigned     sleep(unsigned);
14672 XSI   void          swab(const void *restrict, void *restrict, ssize_t);
14673     int          symlink(const char *, const char *);
14674     void          sync(void);
14675     long         sysconf(int);
14676     pid_t        tcgetpgrp(int);
14677     int          tcsetpgrp(int, pid_t);
14678 XSI   int          truncate(const char *, off_t);
14679     char         *ttyname(int);
14680     int          ttyname_r(int, char *, size_t);
14681 XSI   useconds_t  ualarm(useconds_t, useconds_t);
14682     int          unlink(const char *);
14683 XSI   int          usleep(useconds_t);
14684     pid_t        vfork(void);
14685     ssize_t      write(int, const void *, size_t);

```

14686 Implementations may also include the *pthread\_atfork()* prototype as defined in <pthread.h> (on  
14687 page 286).

14688 The following external variables shall be declared:

```

14689     extern char   *optarg;
14690     extern int    optind, opterr, optopt;

```

#### 14691 APPLICATION USAGE

14692 IEEE Std 1003.1-200x only describes the behavior of systems that claim conformance to it. |  
14693 However, application developers who want to write applications that adapt to other versions of |  
14694 IEEE Std 1003.1 (or to systems that do not conform to any POSIX standard) may find it useful to |  
14695 code them so as to conditionally compile different code depending on the value of |  
14696 `_POSIX_VERSION`, for example. |

```

14697     #if _POSIX_VERSION == 200xxxL |
14698     /* Use the newer function that copes with large files. */ |
14699     off_t pos=ftello(fp); |
14700     #else |
14701     /* Either this is an old version of POSIX, or _POSIX_VERSION is |
14702        not even defined, so use the traditional function. */ |
14703     long pos=ftell(fp); |
14704     #endif |

```

14705 Earlier versions of IEEE Std 1003.1 and of the Single UNIX Specification can be identified by the |  
14706 following macros: |

```

14707     POSIX.1-1988 standard |
14708         _POSIX_VERSION==198808L |
14709     POSIX.1-1990 standard |
14710         _POSIX_VERSION==199009L |
14711     ISO POSIX-1:1996 standard |
14712         _POSIX_VERSION==199506L |

```

14713 Single UNIX Specification, Version 1  
14714 `_XOPEN_UNIX` and `_XOPEN_VERSION=` =4

14715 Single UNIX Specification, Version 2  
14716 `_XOPEN_UNIX` and `_XOPEN_VERSION=` =500

14717 IEEE Std 1003.1-200x does not make any attempt to define application binary interaction with  
14718 the underlying operating system. However, application developers may find it useful to query  
14719 `_SC_VERSION` at runtime via `sysconf()` to determine whether the current version of the  
14720 operating system supports the necessary functionality as in the following program fragment:

```
14721 if (sysconf(_SC_VERSION) < 200xxxL) {  
14722     fprintf(stderr, "POSIX.1-200x system required, terminating \n");  
14723     exit(1);  
14724 }
```

14725 **RATIONALE**

14726 As IEEE Std 1003.1-200x evolved, certain options became sufficiently standardized that it was  
14727 concluded that simply requiring one of the option choices was simpler than retaining the option.  
14728 However, for backwards compatibility, the option flags (with required constant values) are  
14729 retained.

14730 **Version Test Macros**

14731 The standard developers considered altering the definition of `_POSIX_VERSION` and removing  
14732 `_SC_VERSION` from the specification of `sysconf()` since the utility to an application was deemed  
14733 by some to be minimal, and since the implementation of the functionality is potentially  
14734 problematic. However, they recognized that support for existing application binaries is a  
14735 concern to manufacturers, application developers, and the users of implementations conforming  
14736 to IEEE Std 1003.1-200x.

14737 While the example using `_SC_VERSION` in the APPLICATION USAGE section does not provide  
14738 the greatest degree of imaginable utility to the application developer or user, it is arguably better  
14739 than a core dump or some other equally obscure result. (It is also possible for implementations  
14740 to encode and recognize application binaries compiled in various POSIX.1-conforming  
14741 environments, and modify the semantics of the underlying system to conform to the  
14742 expectations of the application.) For the reasons outlined in the preceding paragraphs and in the  
14743 APPLICATION USAGE section, the standard developers elected to retain the `_POSIX_VERSION`  
14744 and `_SC_VERSION` functionality.

14745 **Compile-Time Symbolic Constants for System-Wide Options**

14746 IEEE Std 1003.1-200x now includes support in certain areas for the newly adopted policy  
14747 governing options and stubs.

14748 This policy provides flexibility for implementations in how they support options. It also  
14749 specifies how conforming applications can adapt to different implementations that support  
14750 different sets of options. It allows the following:

- 14751 1. If an implementation has no interest in supporting an option, it does not have to provide  
14752 anything associated with that option beyond the announcement that it does not support it.
- 14753 2. An implementation can support a partial or incompatible version of an option (as a non-  
14754 standard extension) as long as it does not claim to support the option.
- 14755 3. An application can determine whether the option is supported. A strictly conforming  
14756 application must check this announcement mechanism before first using anything  
14757 associated with the option.

14758 There is an important implication of this policy. IEEE Std 1003.1-200x cannot dictate the  
 14759 behavior of interfaces associated with an option when the implementation does not claim to  
 14760 support the option. In particular, it cannot require that a function associated with an  
 14761 unsupported option will fail if it does not perform as specified. However, this policy does not  
 14762 prevent a standard from requiring certain functions to always be present, but that they shall  
 14763 always fail on some implementations. The *setpgid()* function in the POSIX.1-1990 standard, for  
 14764 example, is considered appropriate.

14765 The POSIX standards include various options, and the C language binding support for an option  
 14766 implies that the implementation must supply data types and function interfaces. An application  
 14767 must be able to discover whether the implementation supports each option.

14768 Any application must consider the following three cases for each option:

14769 1. Option never supported.

14770 The implementation advertises at compile time that the option will never be supported. In  
 14771 this case, it is not necessary for the implementation to supply any of the data types or  
 14772 function interfaces that are provided only as part of the option. The implementation might  
 14773 provide data types and functions that are similar to those defined by IEEE Std 1003.1-200x,  
 14774 but there is no guarantee for any particular behavior.

14775 2. Option always supported.

14776 The implementation advertises at compile time that the option will always be supported.  
 14777 In this case, all data types and function interfaces shall be available and shall operate as  
 14778 specified.

14779 3. Option might or might not be supported.

14780 Some implementations might not provide a mechanism to specify support of options at  
 14781 compile time. In addition, the implementation might be unable or unwilling to specify  
 14782 support or non-support at compile time. In either case, any application that might use the  
 14783 option at runtime must be able to compile and execute. The implementation must provide,  
 14784 at compile time, all data types and function interfaces that are necessary to allow this. In  
 14785 this situation, there must be a mechanism that allows the application to query, at runtime,  
 14786 whether the option is supported. If the application attempts to use the option when it is  
 14787 not supported, the result is unspecified unless explicitly specified otherwise in  
 14788 IEEE Std 1003.1-200x.

#### 14789 FUTURE DIRECTIONS

14790 None.

#### 14791 SEE ALSO

14792 <inttypes.h>, <limits.h>, <sys/socket.h>, <sys/types.h>, <termios.h>, <wctype.h>, the System  
 14793 Interfaces volume of IEEE Std 1003.1-200x, *access()*, *alarm()*, *chdir()*, *chown()*, *close()*, *crypt()*,  
 14794 *ctermid()*, *dup()*, *encrypt()*, *environ*, *exec()*, *exit()*, *fchdir()*, *fchown()*, *fcntl()*, *fork()*, *fpathconf()*,  
 14795 *fsync()*, *ftruncate()*, *getcwd()*, *getegid()*, *geteuid()*, *getgid()*, *getgroups()*, *gethostid()*, *gethostname()*,  
 14796 *getlogin()*, *getpgid()*, *getpgrp()*, *getpid()*, *getppid()*, *getsid()*, *getuid()*, *isatty()*, *lchown()*, *link()*,  
 14797 *lockf()*, *lseek()*, *nice()*, *pathconf()*, *pause()*, *pipe()*, *read()*, *readlink()*, *rmdir()*, *setgid()*, *setpgid()*,  
 14798 *setpgrp()*, *setregid()*, *setreuid()*, *setsid()*, *setuid()*, *sleep()*, *swab()*, *symlink()*, *sync()*, *sysconf()*,  
 14799 *tcgetpgrp()*, *tcsetpgrp()*, *truncate()*, *ttyname()*, *ualarm()*, *unlink()*, *usleep()*, *vfork()*, *write()*

#### 14800 CHANGE HISTORY

14801 First released in Issue 1. Derived from Issue 1 of the SVID.

14802 **Issue 5**

14803 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX  
14804 Threads Extension.

14805 The symbolic constants `_XOPEN_REALTIME` and `_XOPEN_REALTIME_THREADS` are added.  
14806 `_POSIX2_C_BIND`, `_XOPEN_ENH_I18N`, and `_XOPEN_SHM` must now be set to a value other  
14807 than `-1` by a conforming implementation.

14808 Large File System extensions are added.

14809 The type of the argument to `sbrk()` is changed from `int` to `intptr_t`.

14810 `_XBS_` constants are added to the list of constants for Options and Option Groups, to the list of  
14811 constants for the `confstr()` function, and to the list of constants to the `sysconf()` function. These  
14812 are all marked EX.

14813 **Issue 6**

14814 `_POSIX2_C_VERSION` is removed.

14815 The Open Group Corrigendum U026/4 is applied, adding the prototype for `fdatasync()`.

14816 The Open Group Corrigendum U026/1 is applied, adding the symbols `_SC_XOPEN_LEGACY`,  
14817 `_SC_XOPEN_REALTIME`, and `_SC_XOPEN_REALTIME_THREADS`.

14818 The symbols `_XOPEN_STREAMS` and `_SC_XOPEN_STREAMS` are added to support the XSI  
14819 STREAMS Option Group.

14820 Text in the DESCRIPTION relating to conformance requirements is moved elsewhere in |  
14821 IEEE Std 1003.1-200x.

14822 The legacy symbol `_SC_PASS_MAX` is removed.

14823 The following new requirements on POSIX implementations derive from alignment with the  
14824 Single UNIX Specification:

14825 • The `_CS_POSIX_*` and `_CS_XBS5_*` constants are added for the `confstr()` function. |

14826 • The `_SC_XBS5_*` constants are added for the `sysconf()` function.

14827 • The symbolic constants `F_ULOCK`, `F_LOCK`, `F_TLOCK`, and `F_TEST` are added.

14828 • The `uid_t`, `gid_t`, `off_t`, `pid_t`, and `useconds_t` types are mandated.

14829 The `gethostname()` prototype is added for sockets.

14830 New section added for System Wide Options.

14831 Function prototypes for `setegid()` and `seteuid()` are added.

14832 Option symbolic constants are added for `_POSIX_ADVISORY_INFO`, `_POSIX_CPUTIME`, |  
14833 `_POSIX_SPAWN`, `_POSIX_SPORADIC_SERVER`, `_POSIX_THREAD_CPUTIME`,  
14834 `_POSIX_THREAD_SPORADIC_SERVER`, and `_POSIX_TIMEOUTS`, and `pathconf()` variables are  
14835 added for `_PC_ALLOC_SIZE_MIN`, `_PC_REC_INCR_XFER_SIZE`, `_PC_REC_MAX_XFER_SIZE`,  
14836 `_PC_REC_MIN_XFER_SIZE`, and `_PC_REC_XFER_ALIGN` for alignment with  
14837 IEEE Std 1003.1d-1999. |

14838 The following are added for alignment with IEEE Std 1003.1j-2000:

14839 • Option symbolic constants `_POSIX_BARRIERS`, `_POSIX_CLOCK_SELECTION`,  
14840 `_POSIX_MONOTONIC_CLOCK`, `_POSIX_READER_WRITER_LOCKS`,  
14841 `_POSIX_SPIN_LOCKS`, and `_POSIX_TYPED_MEMORY_OBJECTS`

14842       • *sysconf()* variables `_SC_BARRIERS`, `_SC_CLOCK_SELECTION`,  
14843        `_SC_MONOTONIC_CLOCK`, `_SC_READER_WRITER_LOCKS`, `_SC_SPIN_LOCKS`, and  
14844        `_SC_TYPED_MEMORY_OBJECTS`

14845       The `_SC_XBS5` macros associated with the ISO/IEC 9899:1990 standard are marked LEGACY,  
14846       and new equivalent `_SC_V6` macros associated with the ISO/IEC 9899:1999 standard are  
14847       introduced.

14848       The *getwd()* function is marked LEGACY.

14849       The **restrict** keyword is added to the prototypes for *readlink()* and *swab()*. |

14850       Constants for options are now harmonized, so when supported they take the year of approval of  
14851       IEEE Std 1003.1-200x as the value.

14852       The following are added for alignment with IEEE Std 1003.1q-2000:

14853       • Optional symbolic constants `_POSIX_TRACE`, `_POSIX_TRACE_EVENT_FILTER`,  
14854        `_POSIX_TRACE_LOG`, and `_POSIX_TRACE_INHERIT`

14855       • The *sysconf()* symbolic constants `_SC_TRACE`, `_SC_TRACE_EVENT_FILTER`,  
14856        `_SC_TRACE_LOG`, and `_SC_TRACE_INHERIT`.

14857       The *brk()* and *sbrk()* legacy functions are removed. |

14858       The Open Group Base Resolution bwg2001-006 is applied, which reworks the XSI versioning |  
14859       information. |

14860       The Open Group Base Resolution bwg2001-008 is applied, changing the *namelen* parameter for |  
14861       *gethostname()* from **socklen\_t** to **size\_t**. |

14862 **NAME**14863 `utime.h` — access and modification times structure14864 **SYNOPSIS**14865 `#include <utime.h>`14866 **DESCRIPTION**14867 The **<utime.h>** header shall declare the structure **utimbuf**, which shall include the following  
14868 members:14869 `time_t` `actime` Access time.  
14870 `time_t` `modtime` Modification time.

14871 The times shall be measured in seconds since the Epoch.

14872 The type **time\_t** shall be defined as described in **<sys/types.h>**.14873 The following shall be declared as a function and may also be defined as a macro. A function |  
14874 prototype shall be provided. |14875 `int utime(const char *, const struct utimbuf *);`14876 **APPLICATION USAGE**

14877 None.

14878 **RATIONALE**

14879 None.

14880 **FUTURE DIRECTIONS**

14881 None.

14882 **SEE ALSO**14883 **<sys/types.h>**, the System Interfaces volume of IEEE Std 1003.1-200x, *utime()*14884 **CHANGE HISTORY**

14885 First released in Issue 3.

14886 **Issue 6**14887 The following new requirements on POSIX implementations derive from alignment with the  
14888 Single UNIX Specification:

- 14889
- The **time\_t** type is defined.

14890 **NAME**

14891 utmpx.h — user accounting database definitions

14892 **SYNOPSIS**

14893 XSI `#include <utmpx.h>`

14894

14895 **DESCRIPTION**

14896 The <utmpx.h> header shall define the **utmpx** structure that shall include at least the following  
 14897 members:

14898	char	ut_user[]	User login name.
14899	char	ut_id[]	Unspecified initialization process identifier.
14900	char	ut_line[]	Device name.
14901	pid_t	ut_pid	Process ID.
14902	short	ut_type	Type of entry.
14903	struct timeval	ut_tv	Time entry was made.

14904 The **pid\_t** type shall be defined through **typedef** as described in <sys/types.h>.

14905 The **timeval** structure shall be defined as described in <sys/time.h>.

14906 Inclusion of the <utmpx.h> header may also make visible all symbols from <sys/time.h>.

14907 The following symbolic constants shall be defined as possible values for the *ut\_type* member of  
 14908 the **utmpx** structure:

14909	EMPTY	No valid user accounting information.
14910	BOOT_TIME	Identifies time of system boot.
14911	OLD_TIME	Identifies time when system clock changed.
14912	NEW_TIME	Identifies time after system clock changed.
14913	USER_PROCESS	Identifies a process.
14914	INIT_PROCESS	Identifies a process spawned by the init process.
14915	LOGIN_PROCESS	Identifies the session leader of a logged in user.
14916	DEAD_PROCESS	Identifies a session leader who has exited.

14917 The following shall be declared as functions and may also be defined as macros. Function |  
 14918 prototypes shall be provided. |

```

14919 void          endutxent(void);
14920 struct utmpx *getutxent(void);
14921 struct utmpx *getutxid(const struct utmpx *);
14922 struct utmpx *getutxline(const struct utmpx *);
14923 struct utmpx *pututxline(const struct utmpx *);
14924 void          setutxent(void);
    
```

14925 **APPLICATION USAGE**

14926       None.

14927 **RATIONALE**

14928       None.

14929 **FUTURE DIRECTIONS**

14930       None.

14931 **SEE ALSO**

14932       <sys/time.h>, <sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-200x, *endutxent()*

14933 **CHANGE HISTORY**

14934       First released in Issue 4, Version 2.

14935 **NAME**

14936 `wchar.h` — wide-character handling

14937 **SYNOPSIS**

14938 `#include <wchar.h>`

14939 **DESCRIPTION**

14940 CX Some of the functionality described on this reference page extends the ISO C standard.  
 14941 Applications shall define the appropriate feature test macro (see the System Interfaces volume of  
 14942 IEEE Std 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of these  
 14943 symbols in this header.

14944 The <wchar.h> header shall define the following types:

14945 **wchar\_t** As described in <stddef.h>.

14946 **wint\_t** An integer type capable of storing any valid value of **wchar\_t** or WEOF.

14947 XSI **wctype\_t** A scalar type of a data object that can hold values which represent locale-  
 14948 specific character classification.

14949 **mbstate\_t** An object type other than an array type that can hold the conversion state  
 14950 information necessary to convert between sequences of (possibly multi-byte)  
 14951 XSI characters and wide characters. If a codeset is being used such that an  
 14952 **mbstate\_t** needs to preserve more than 2 levels of reserved state, the results  
 14953 are unspecified.

14954 XSI **FILE** As described in <stdio.h>.

14955 **size\_t** As described in <stddef.h>.

14956 XSI **va\_list** As described in <stdarg.h>.

14957 The implementation shall support one or more programming environments in which the width  
 14958 of **wint\_t** is no greater than the width of type **long**. The names of these programming  
 14959 environments can be obtained using the *confstr()* function or the *getconf* utility.

14960 The following shall be declared as functions and may also be defined as macros. Function  
 14961 prototypes shall be provided.

```

14962 wint_t      btowc(int);
14963 wint_t      fgetwc(FILE *);
14964 wchar_t     *fgetws(wchar_t *restrict, int, FILE *restrict);
14965 wint_t      fputwc(wchar_t, FILE *);
14966 int         fputws(const wchar_t *restrict, FILE *restrict);
14967 int         fwide(FILE *, int);
14968 int         fwprintf(FILE *restrict, const wchar_t *restrict, ...);
14969 int         fwscanf(FILE *restrict, const wchar_t *restrict, ...);
14970 wint_t      getwc(FILE *);
14971 wint_t      getwchar(void);
14972 XSI int     iswalnum(wint_t);
14973 int         iswalpha(wint_t);
14974 int         iswcntrl(wint_t);
14975 int         iswctype(wint_t, wctype_t);
14976 int         iswdigit(wint_t);
14977 int         iswgraph(wint_t);
14978 int         iswlower(wint_t);
14979 int         iswprint(wint_t);
14980 int         iswpunct(wint_t);
    
```

```

14981     int             iswspace(wint_t);
14982     int             iswupper(wint_t);
14983     int             iswxdigit(wint_t);
14984     size_t          mbrlen(const char *restrict, size_t, mbstate_t *restrict);
14985     size_t          mbrtowc(wchar_t *restrict, const char *restrict, size_t,
14986                          mbstate_t *restrict);
14987     int             mbsinit(const mbstate_t *);
14988     size_t          mbsrtowcs(wchar_t *restrict, const char **restrict, size_t,
14989                          mbstate_t *restrict);
14990     wint_t          putwc(wchar_t, FILE *);
14991     wint_t          putwchar(wchar_t);
14992     int             swprintf(wchar_t *restrict, size_t,
14993                          const wchar_t *restrict, ...);
14994     int             swscanf(const wchar_t *restrict,
14995                          const wchar_t *restrict, ...);
14996 XSI    wint_t          tolower(wint_t);
14997     wint_t          towupper(wint_t);
14998     wint_t          ungetwc(wint_t, FILE *);
14999     int             vfwprintf(FILE *restrict, const wchar_t *restrict, va_list);
15000     int             vwscanf(FILE *restrict, const wchar_t *restrict, va_list);
15001     int             vwprintf(const wchar_t *restrict, va_list);
15002     int             vswprintf(wchar_t *restrict, size_t,
15003                          const wchar_t *restrict, va_list);
15004     int             vswscanf(const wchar_t *restrict, const wchar_t *restrict,
15005                          va_list);
15006     int             vwscanf(const wchar_t *restrict, va_list);
15007     size_t          wcrctomb(char *restrict, wchar_t, mbstate_t *restrict);
15008     wchar_t         *wcscat(wchar_t *restrict, const wchar_t *restrict);
15009     wchar_t         *wcschr(const wchar_t *, wchar_t);
15010     int             wcscmp(const wchar_t *, const wchar_t *);
15011     int             wscoll(const wchar_t *, const wchar_t *);
15012     wchar_t         *wcscpy(wchar_t *restrict, const wchar_t *restrict);
15013     size_t          wcscspn(const wchar_t *, const wchar_t *);
15014     size_t          wcsftime(wchar_t *restrict, size_t,
15015                          const wchar_t *restrict, const struct tm *restrict);
15016     size_t          wcslen(const wchar_t *);
15017     wchar_t         *wcsncat(wchar_t *restrict, const wchar_t *restrict, size_t);
15018     int             wcsncmp(const wchar_t *, const wchar_t *, size_t);
15019     wchar_t         *wcsncpy(wchar_t *restrict, const wchar_t *restrict, size_t);
15020     wchar_t         *wcpbrk(const wchar_t *, const wchar_t *);
15021     wchar_t         *wcsrchr(const wchar_t *, wchar_t);
15022     size_t          wcsrtombs(char *restrict, const wchar_t **restrict,
15023                          size_t, mbstate_t *restrict);
15024     size_t          wcsspncpy(const wchar_t *, const wchar_t *);
15025     wchar_t         *wcsstr(const wchar_t *restrict, const wchar_t *restrict);
15026     double          wcstod(const wchar_t *restrict, wchar_t **restrict);
15027     float           wcstof(const wchar_t *restrict, wchar_t **restrict);
15028     wchar_t         *wcstok(wchar_t *restrict, const wchar_t *restrict,
15029                          wchar_t **restrict);
15030     long            wcstol(const wchar_t *restrict, wchar_t **restrict, int);
15031     long double     wcstold(const wchar_t *restrict, wchar_t **restrict);
15032     long long       wcstoll(const wchar_t *restrict, wchar_t **restrict, int);

```

```

15033 unsigned long wcstoul(const wchar_t *restrict, wchar_t **restrict, int);
15034 unsigned long long
15035 wcstoull(const wchar_t *restrict, wchar_t **restrict, int);
15036 XSI wchar_t *wcswcs(const wchar_t *, const wchar_t *);
15037 int wcswidth(const wchar_t *, size_t);
15038 size_t wcsxfrm(wchar_t *restrict, const wchar_t *restrict, size_t);
15039 int wctob(wint_t);
15040 XSI wctype_t wctype(const char *);
15041 int wcwidth(wchar_t);
15042 wchar_t *wmemchr(const wchar_t *, wchar_t, size_t);
15043 int wmemcmp(const wchar_t *, const wchar_t *, size_t);
15044 wchar_t *wmemcpy(wchar_t *restrict, const wchar_t *restrict, size_t);
15045 wchar_t *wmemmove(wchar_t *, const wchar_t *, size_t);
15046 wchar_t *wmemset(wchar_t *, wchar_t, size_t);
15047 int wprintf(const wchar_t *restrict, ...);
15048 int wscanf(const wchar_t *restrict, ...);

```

15049 The <wchar.h> header shall define the following macros:

```

15050 WCHAR_MAX The maximum value representable by an object of type wchar_t.
15051 WCHAR_MIN The minimum value representable by an object of type wchar_t.
15052 WEOF Constant expression of type wint_t that is returned by several WP functions
15053 to indicate end-of-file.
15054 NULL As described in <stddef.h>.

```

15055 The tag **tm** shall be declared as naming an incomplete structure type, the contents of which are  
15056 described in the header <time.h>.

15057 CX Inclusion of the <wchar.h> header may make visible all symbols from the headers <ctype.h>,  
15058 <stdio.h>, <stdarg.h>, <stdlib.h>, <string.h>, <stddef.h>, and <time.h>.

#### 15059 APPLICATION USAGE

15060 None.

#### 15061 RATIONALE

15062 In the ISO C standard, the symbols referenced as XSI extensions are in <wctype.h>. Their  
15063 presence here is thus an extension. |

#### 15064 FUTURE DIRECTIONS

15065 None.

#### 15066 SEE ALSO

15067 <ctype.h>, <stdarg.h>, <stddef.h>, <stdio.h>, <stdlib.h>, <string.h>, <time.h>, the System  
15068 Interfaces volume of IEEE Std 1003.1-200x, *btowc()*, *confstr()*, *fgetwc()*, *fgetws()*, *fputwc()*,  
15069 *fputws()*, *fwide()*, *fwprintf()*, *fwscanf()*, *getwc()*, *getwchar()*, *iswalnum()*, *iswalpna()*, *iswcntrl()*,  
15070 *iswctype()*, *iswdigit()*, *iswgraph()*, *iswlower()*, *iswprint()*, *iswpunct()*, *iswspace()*, *iswupper()*,  
15071 *iswxdigit()*, *iswctype()*, *mbsinit()*, *mbrlen()*, *mbrtowc()*, *mbsrtowcs()*, *putwc()*, *putwchar()*,  
15072 *swprintf()*, *swscanf()*, *towlower()*, *towupper()*, *ungetwc()*, *vfwprintf()*, *vfwscanf()*, *vswprintf()*,  
15073 *vswscanf()*, *vwscanf()*, *wcrtomb()*, *wcsrtombs()*, *wcscat()*, *wcschr()*, *wcscmp()*, *wscoll()*, *wcscpy()*,  
15074 *wcscspn()*, *wcsftime()*, *wcslen()*, *wcsncat()*, *wcsncmp()*, *wcsncpy()*, *wcsprbk()*, *wcsrchr()*, *wcsspn()*,  
15075 *wcsstr()*, *wcstod()*, *wcstof()*, *wcstok()*, *wcstol()*, *wcstold()*, *wcstoll()*, *wcstoul()*, *wcstoull()*, *wcswcs()*,  
15076 *wcswidth()*, *wcsxfrm()*, *wctob()*, *wctype()*, *wcwidth()*, *wmemchr()*, *wmemcmp()*, *wmemcpy()*,  
15077 *wmemmove()*, *wmemset()*, *wprintf()*, *wscanf()*, the Shell and Utilities volume of  
15078 IEEE Std 1003.1-200x, *getconf* |

15079 **CHANGE HISTORY**

15080 First released in Issue 4.

15081 **Issue 5**

15082 Aligned with the ISO/IEC 9899:1990/Amendment 1:1995 (E).

15083 **Issue 6**

15084 The Open Group Corrigendum U021/10 is applied. The prototypes for *wcswidth()* and  
15085 *wcwidth()* are marked as extensions.

15086 The Open Group Corrigendum U028/5 is applied, correcting the prototype for the *mbsinit()*  
15087 function.

15088 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 15089 • Various function prototypes are updated to add the **restrict** keyword.
- 15090 • The functions *vwscanf()*, *vswscanf()*, *wcstof()*, *wcstold()*, *wcstoll()*, and *wcstoull()* are added.

15091 The type **wctype\_t**, the *isw\*()*, *to\*()*, and *wctype()* functions are marked as XSI extensions.

15092 **NAME**

15093 wctype.h — wide-character classification and mapping utilities

15094 **SYNOPSIS**

15095 #include <wctype.h>

15096 **DESCRIPTION**

15097 **CX** Some of the functionality described on this reference page extends the ISO C standard.  
 15098 Applications shall define the appropriate feature test macro (see the System Interfaces volume of  
 15099 IEEE Std 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of these  
 15100 symbols in this header.

15101 The <wctype.h> header shall define the following types:

15102 **wint\_t** As described in <wchar.h>.

15103 **wctrans\_t** A scalar type that can hold values which represent locale-specific character  
 15104 mappings.

15105 **wctype\_t** As described in <wchar.h>.

15106 The following shall be declared as functions and may also be defined as macros. Function  
 15107 prototypes shall be provided.

```

15108 int      iswalnum(wint_t);
15109 int      iswalpha(wint_t);
15110 int      iswblank(wint_t);
15111 int      iswcntrl(wint_t);
15112 int      iswdigit(wint_t);
15113 int      iswgraph(wint_t);
15114 int      iswlower(wint_t);
15115 int      iswprint(wint_t);
15116 int      iswpunct(wint_t);
15117 int      iswspace(wint_t);
15118 int      iswupper(wint_t);
15119 int      iswxdigit(wint_t);
15120 int      iswctype(wint_t, wctype_t);
15121 wint_t   towctrans(wint_t, wctrans_t);
15122 wint_t   tolower(wint_t);
15123 wint_t   toupper(wint_t);
15124 wctrans_t wctrans(const char *);
15125 wctype_t wctype(const char *);
    
```

15126 The <wctype.h> header shall define the following macro name:

15127 **WEOF** Constant expression of type **wint\_t** that is returned by several MSE functions  
 15128 to indicate end-of-file.

15129 For all functions described in this header that accept an argument of type **wint\_t**, the value is  
 15130 representable as a **wchar\_t** or equals the value of **WEOF**. If this argument has any other value,  
 15131 the behavior is undefined.

15132 The behavior of these functions shall be affected by the *LC\_CTYPE* category of the current locale.

15133 **CX** Inclusion of the <wctype.h> header may make visible all symbols from the headers <ctype.h>,  
 15134 <stdio.h>, <stdarg.h>, <stdlib.h>, <string.h>, <stddef.h>, <time.h>, and <wchar.h>.

15135 **APPLICATION USAGE**

15136       None.

15137 **RATIONALE**

15138       None.

15139 **FUTURE DIRECTIONS**

15140       None.

15141 **SEE ALSO**

15142       <locale.h>, <wchar.h>, the System Interfaces volume of IEEE Std 1003.1-200x, *iswalnum()*,  
15143       *iswalph()*, *iswblank()*, *iswcntrl()*, *iswctype()*, *iswdigit()*, *iswgraph()*, *iswlower()*, *iswprint()*,  
15144       *iswpunct()*, *iswspace()*, *iswupper()*, *iswxdigit()*, *setlocale()*, *towctrans()*, *towlower()*, *towupper()*,  
15145       *wctrans()*, *wctype()*

15146 **CHANGE HISTORY**

15147       First released in Issue 5. Derived from the ISO/IEC 9899:1990/Amendment 1:1995 (E).

15148 **Issue 6**

15149       The *iswblank()* function is added for alignment with the ISO/IEC 9899:1999 standard.

15150 NAME

15151 wordexp.h — word-expansion types

15152 SYNOPSIS

15153 #include <wordexp.h>

15154 DESCRIPTION

15155 The <wordexp.h> header shall define the structures and symbolic constants used by the  
15156 *wordexp()* and *wordfree()* functions.

15157 The structure type **wordexp\_t** shall contain at least the following members:

15158 size\_t we\_wordc Count of words matched by *words*.  
15159 char \*\*we\_wordv Pointer to list of expanded words.  
15160 size\_t we\_offs Slots to reserve at the beginning of *we\_wordv*.

15161 The *flags* argument to the *wordexp()* function shall be the bitwise-inclusive OR of the following  
15162 flags:

- 15163 WRDE\_APPEND Append words to those previously generated.
- 15164 WRDE\_DOOFFS Number of null pointers to prepend to *we\_wordv*.
- 15165 WRDE\_NOCMD Fail if command substitution is requested.
- 15166 WRDE\_REUSE The *pwordexp* argument was passed to a previous successful call to  
15167 *wordexp()*, and has not been passed to *wordfree()*. The result is the same  
15168 as if the application had called *wordfree()* and then called *wordexp()*  
15169 without WRDE\_REUSE.
- 15170 WRDE\_SHOWERR Do not redirect *stderr* to */dev/null*.
- 15171 WRDE\_UNDEF Report error on an attempt to expand an undefined shell variable.

15172 The following constants shall be defined as error return values:

- 15173 WRDE\_BADCHAR One of the unquoted characters—<newline>, ' | ', '&', ';', '<', '>',  
15174 ' ( ', ' ) ', '{ ', ' } '—appears in *words* in an inappropriate context.
- 15175 WRDE\_BADVAL Reference to undefined shell variable when WRDE\_UNDEF is set in *flags*.
- 15176 WRDE\_CMDSUB Command substitution requested when WRDE\_NOCMD was set in *flags*.
- 15177 WRDE\_NOSPACE Attempt to allocate memory failed.
- 15178 OB XSI WRDE\_NOSYS Reserved.
- 15179 WRDE\_SYNTAX Shell syntax error, such as unbalanced parentheses or unterminated  
15180 string.

15181 The <wordexp.h> header shall define the following type: |

15182 XSI **size\_t** As described in <stddef.h>. |

15183 The following shall be declared as functions and may also be defined as macros. Function |  
15184 prototypes shall be provided. |

15185 int wordexp(const char \*restrict, wordexp\_t \*restrict, int);  
15186 void wordfree(wordexp\_t \*);

15187 The implementation may define additional macros or constants using names beginning with  
15188 WRDE\_.

15189 **APPLICATION USAGE**

15190           None.

15191 **RATIONALE**

15192           None.

15193 **FUTURE DIRECTIONS**

15194           None.

15195 **SEE ALSO**

15196           <stddef.h>, the System Interfaces volume of IEEE Std 1003.1-200x, *wordexp()*, the Shell and |

15197           Utilities volume of IEEE Std 1003.1-200x

15198 **CHANGE HISTORY**

15199           First released in Issue 4. Derived from the ISO POSIX-2 standard.

15200 **Issue 6**

15201           The **restrict** keyword is added to the prototype for *wordexp()*.

15202           The WRDE\_NOSYS constant is marked obsolescent.

# Index

1

2	(time) resolution .....	77	<poll.h>.....	284
3	/ .....	181	<pthread.h> .....	286
4	/dev .....	181	<pwd.h> .....	291
5	/dev/console .....	181	<regex.h>.....	292
6	/dev/null.....	181	<sched.h> .....	294
7	/dev/tty .....	181	<search.h>.....	296
8	/tmp .....	181	<semaphore.h> .....	298
9	<aio.h> .....	202	<setjmp.h> .....	299
10	<alert> .....	34	<signal.h>.....	300
11	<arpa/inet.h> .....	204	<space> .....	81
12	<assert.h> .....	205	<spawn.h> .....	307
13	<backspace> .....	37	<stdarg.h> .....	309
14	<blank> .....	42	<stdbool.h> .....	311
15	<carriage-return>.....	44	<stddef.h> .....	312
16	<complex.h> .....	206	<stdint.h> .....	313
17	<cpio.h> .....	209	<stdio.h>.....	320
18	<ctype.h>.....	210	<stdlib.h> .....	324
19	<dirent.h>.....	212	<string.h> .....	328
20	<dlfcn.h> .....	214	<strings.h> .....	330
21	<errno.h>.....	215	<stropts.h> .....	331
22	<fcntl.h> .....	219	<sys/dir.h> .....	212
23	<fenv.h>.....	222	<sys/ipc.h>.....	336
24	<float.h> .....	225	<sys/mman.h>.....	338
25	<fmtmsg.h> .....	229	<sys/msg.h>.....	341
26	<fnmatch.h> .....	231	<sys/resource.h> .....	343
27	<form-feed> .....	58	<sys/select.h> .....	345
28	<ftw.h> .....	232	<sys/sem.h> .....	347
29	<glob.h>.....	234	<sys/shm.h>.....	349
30	<grp.h> .....	236	<sys/socket.h> .....	351
31	<iconv.h>.....	237	<sys/stat.h> .....	356
32	<inttypes.h>.....	238	<sys/statvfs.h> .....	360
33	<iso646.h> .....	240	<sys/time.h> .....	362
34	<langinfo.h> .....	241	<sys/timeb.h>.....	364
35	<libgen.h> .....	244	<sys/times.h> .....	365
36	<limits.h> .....	245	<sys/types.h> .....	366
37	<locale.h> .....	260	<sys/uio.h>.....	369
38	<math.h> .....	262	<sys/un.h>.....	370
39	<monetary.h> .....	269	<sys/utsname.h>.....	371
40	<mqueue.h>.....	270	<sys/wait.h> .....	372
41	<ndbm.h>.....	272	<syslog.h> .....	374
42	<net/if.h>.....	273	<tab> .....	86
43	<netdb.h> .....	274	<tar.h>.....	376
44	<netinet/in.h>.....	278	<termios.h>.....	378
45	<netinet/tcp.h>.....	282	<tgmath.h> .....	384
46	<newline>.....	65	<time.h> .....	388
47	<nl_types.h> .....	283	<trace.h>.....	392

48	<ucontext.h>.....	<b>396</b>	_POSIX2_LINE_MAX.....	250, 254, 257
49	<ulimit.h>.....	<b>397</b>	_POSIX2_LOCALEDEF.....	402
50	<unistd.h>.....	<b>398</b>	_POSIX2_PBS.....	402
51	<utime.h>.....	<b>418</b>	_POSIX2_PBS_ACCOUNTING.....	402
52	<utmpx.h>.....	<b>419</b>	_POSIX2_PBS_CHECKPOINT.....	402
53	<vertical-tab>.....	91	_POSIX2_PBS_LOCATE.....	402
54	<wchar.h>.....	<b>421</b>	_POSIX2_PBS_MESSAGE.....	402
55	<wctype.h>.....	<b>425</b>	_POSIX2_PBS_TRACK.....	402
56	<wordexp.h>.....	<b>427</b>	_POSIX2_RE_DUP_MAX.....	247, 250, 254
57	±0.....	93	_POSIX2_SW_DEV.....	403
58	_Complex_I.....	206	_POSIX2_UPE.....	403
59	_CS_XBS5_ILP32_OFF32_CFLAGS.....	406	_POSIX2_VERSION.....	398
60	_CS_XBS5_ILP32_OFF32_LDFLAGS.....	406	_POSIX_ADVISORY_INFO.....	17, 23, 399
61	_CS_XBS5_ILP32_OFF32_LIBS.....	406	_POSIX_AIO_LISTIO_MAX.....	246, 251
62	_CS_XBS5_ILP32_OFF32_LINTFLAGS.....	406	_POSIX_AIO_MAX.....	246, 251
63	_CS_XBS5_ILP32_OFFBIG_CFLAGS.....	406	_POSIX_ARG_MAX.....	246, 251
64	_CS_XBS5_ILP32_OFFBIG_LDFLAGS.....	406	_POSIX_ASYNCHRONOUS_IO.....	17, 22, 399
65	_CS_XBS5_ILP32_OFFBIG_LIBS.....	407	_POSIX_ASYNC_IO.....	404
66	_CS_XBS5_ILP32_OFFBIG_LINTFLAGS.....	407	_POSIX_BARRIERS.....	17, 24, 399
67	_CS_XBS5_LP64_OFF64_CFLAGS.....	407	_POSIX_CHILD_MAX.....	246, 251
68	_CS_XBS5_LP64_OFF64_LDFLAGS.....	407	_POSIX_CHOWN_RESTRICTED.....	16, 399
69	_CS_XBS5_LP64_OFF64_LIBS.....	407	_POSIX_CLOCKRES_MIN.....	251
70	_CS_XBS5_LP64_OFF64_LINTFLAGS.....	407	_POSIX_CLOCK_SELECTION.....	17, 23, 399
71	_CS_XBS5_LPBIG_OFFBIG_CFLAGS.....	407	_POSIX_CPUTIME.....	17, 23, 399
72	_CS_XBS5_LPBIG_OFFBIG_LDFLAGS.....	407	_POSIX_DELAYTIMER_MAX.....	246, 251
73	_CS_XBS5_LPBIG_OFFBIG_LIBS.....	407	_POSIX_FSYNC.....	17, 19, 22, 399
74	_CS_XBS5_LPBIG_OFFBIG_LINTFLAGS.....	407	_POSIX_IPV6.....	17
75	_Imaginary_I.....	206	_POSIX_JOB_CONTROL.....	16, 399
76	_IOFBF.....	320	_POSIX_LINK_MAX.....	249, 251
77	_IOLBF.....	320	_POSIX_LOGIN_NAME_MAX.....	246, 251
78	_IONBF.....	320	_POSIX_MAPPED_FILES.....	17, 19, 22, 399
79	_MIN.....	245	_POSIX_MAPPED_FILES,.....	22
80	_PC constants		_POSIX_MAX_CANON.....	249, 252
81	defined in <unistd.h>.....	<b>408</b>	_POSIX_MAX_INPUT.....	249, 252
82	_POSIX.....	245	_POSIX_MEMLOCK.....	17, 22, 399
83	_POSIX maximum values		_POSIX_MEMLOCK_RANGE.....	17, 22, 399
84	in <limits.h>.....	<b>251</b>	_POSIX_MEMORY_PROTECTION.....	17, 19, 22, 399
85	_POSIX minimum values		_POSIX_MESSAGE_PASSING.....	17, 22, 399
86	in <limits.h>.....	<b>251</b>	_POSIX_MONOTONIC_CLOCK.....	17, 23, 399
87	_POSIX2_BC_BASE_MAX.....	250, 254	_POSIX_MQ_OPEN_MAX.....	246, 252
88	_POSIX2_BC_DIM_MAX.....	250, 254	_POSIX_MQ_PRIO_MAX.....	247, 252
89	_POSIX2_BC_SCALE_MAX.....	250, 254	_POSIX_NAME_MAX.....	249, 252
90	_POSIX2_BC_STRING_MAX.....	250, 254	_POSIX_NGROUPS_MAX.....	250, 252
91	_POSIX2_CHARCLASS_NAME_MAX.....	250, 254	_POSIX_NO_TRUNC.....	16, 98, 399
92	_POSIX2_CHAR_TERM.....	402	_POSIX_OPEN_MAX.....	247, 252
93	_POSIX2_COLL_WEIGHTS_MAX.....	250, 254	_POSIX_PATH_MAX.....	249, 252
94	_POSIX2_C_BIND.....	402	_POSIX_PIPE_BUF.....	249, 252
95	_POSIX2_C_DEV.....	402	_POSIX_PRIORITIZED_IO.....	17, 22, 400
96	_POSIX2_EXPR_NEST_MAX.....	250, 254	_POSIX_PRIORITY_SCHEDULING.....	17, 22-23, 400
97	_POSIX2_FORT_DEV.....	402	_POSIX_PRIO_IO.....	404
98	_POSIX2_FORT_RUN.....	402	_POSIX_RAW_SOCKETS.....	17

## Index

99	<code>_POSIX_READER_WRITER_LOCKS</code> .....	400	<code>_POSIX_TRACE_INHERIT</code> .....	18, 25, 401
100	<code>_POSIX_REALTIME_SIGNALS</code> .....	17, 22, 400	<code>_POSIX_TRACE_LOG</code> .....	18, 25
101	<code>_POSIX_REGEX</code> .....	400	<code>_POSIX_TRACE_NAME_MAX</code> .....	248, 253
102	<code>_POSIX_RE_DUP_MAX</code> .....	252	<code>_POSIX_TRACE_SYS_MAX</code> .....	248, 253
103	<code>_POSIX_RTSIG_MAX</code> .....	247, 252	<code>_POSIX_TRACE_USER_EVENT_MAX</code> ...	248, 253
104	<code>_POSIX_SAVED_IDS</code> .....	16, 400	<code>_POSIX_TTY_NAME_MAX</code> .....	248, 253
105	<code>_POSIX_SEMAPHORES</code> .....	17, 22, 400	<code>_POSIX_TYPED_MEMORY_OBJECTS</code> .18, 23, 402	
106	<code>_POSIX_SEM_NSEMS_MAX</code> .....	247, 252	<code>_POSIX_TZNAME_MAX</code> .....	248, 254
107	<code>_POSIX_SEM_VALUE_MAX</code> .....	247, 252	<code>_POSIX_VDISABLE</code> .....	16, 402
108	<code>_POSIX_SHARED_MEMORY_OBJECTS</code> .....	17	<code>_POSIX_VERSION</code> .....	398
109	.....	22, 400	<code>_SC constants</code>	
110	<code>_POSIX_SHELL</code> .....	400	defined in <code>&lt;unistd.h&gt;</code> .....	<b>408</b>
111	<code>_POSIX_SIGQUEUE_MAX</code> .....	247, 252	<code>_V6_ILP32_OFF32</code> .....	403
112	<code>_POSIX_SPAWN</code> .....	17, 23, 400	<code>_V6_ILP32_OFFBIG</code> .....	403
113	<code>_POSIX_SPIN_LOCKS</code> .....	17, 24, 400	<code>_V6_LP64_OFF64</code> .....	403
114	<code>_POSIX_SPARADIC_SERVER</code> .....	17, 23, 400	<code>_V6_LPBIG_OFFBIG</code> .....	403
115	<code>_POSIX_SSIZE_MAX</code> .....	253, 255	<code>_XBS5_ILP32_OFF32</code> .....	403
116	<code>_POSIX_SS_REPL_MAX</code> .....	247, 253	<code>_XBS5_ILP32_OFFBIG</code> .....	403
117	<code>_POSIX_STREAM_MAX</code> .....	248, 253	<code>_XBS5_LP64_OFF64</code> .....	403
118	<code>_POSIX_SYMLINK_MAX</code> .....	249, 253	<code>_XBS5_LPBIG_OFFBIG</code> .....	403
119	<code>_POSIX_SYMLINK_MAX</code> .....	248, 253	<code>_XOPEN_CRYPT</code> .....	18, 21, 403
120	<code>_POSIX_SYNCHRONIZED_IO</code> .....	17, 22, 400	<code>_XOPEN_ENH_I18N</code> .....	403
121	<code>_POSIX_SYNC_IO</code> .....	404	<code>_XOPEN_IOV_MAX</code> .....	246, 254
122	<code>_POSIX_THREADS</code> .....	17, 19, 24, 401	<code>_XOPEN_LEGACY</code> .....	18, 25-26, 403
123	<code>_POSIX_THREAD_ATTR_STACKADDR</code> .....	17	<code>_XOPEN_NAME_MAX</code> .....	249, 254
124	.....	19, 401	<code>_XOPEN_PATH_MAX</code> .....	249, 255
125	<code>_POSIX_THREAD_ATTR_STACKSIZE</code> .....	17	<code>_XOPEN_REALTIME</code> .....	18, 22, 403
126	.....	19, 401	<code>_XOPEN_REALTIME_THREADS</code> ...	18, 23-24, 403
127	<code>_POSIX_THREAD_CPUTIME</code> .....	17, 24, 401	<code>_XOPEN_SHM</code> .....	403
128	<code>_POSIX_THREAD_DESTRUCTOR</code> .....		<code>_XOPEN_STREAMS</code> .....	25, 404
129	<code>ITERATIONS</code> .....	247, 253	<code>_XOPEN_UNIX</code> .....	18-19
130	<code>_POSIX_THREAD_KEYS_MAX</code> .....	247, 253	<code>_XOPEN_VERSION</code> .....	398
131	<code>_POSIX_THREAD_PRIORITY_SCHEDULING</code> ...		<code>ABDAY</code> .....	242
132	.....	17, 23-24, 401	<code>ABMON</code> .....	242
133	<code>_POSIX_THREAD_PRIO_INHERIT</code> .....	17	abortive release .....	33
134	.....	23-24, 401	absolute pathname .....	33, 98
135	<code>_POSIX_THREAD_PRIO_PROTECT</code> .....	17	access mode .....	33
136	.....	23-24, 401	additional file access control mechanism.....	33
137	<code>_POSIX_THREAD_PROCESS_SHARED</code> .....	17	address space.....	33
138	.....	19, 401	<code>ADV</code> .....	<b>6</b>
139	<code>_POSIX_THREAD_SAFE_FUNCTIONS</code> .....	17	advanced realtime .....	23
140	.....	19, 24, 401	<code>ADVANCED_REALTIME</code> .....	307
141	<code>_POSIX_THREAD_SPORADIC_SERVER</code> .....	17	advanced realtime threads .....	24
142	.....	24, 401	advisory information .....	33
143	<code>_POSIX_THREAD_THREADS_MAX</code> .....	247, 253	affirmative response .....	33
144	<code>_POSIX_TIMEOUTS</code> .....	18, 23, 401	<code>AF_INET</code> .....	353
145	<code>_POSIX_TIMERS</code> .....	18, 22-24	<code>AF_INET6</code> .....	353
146	<code>_POSIX_TIMER_MAX</code> .....	248, 253	<code>AIO</code> .....	<b>6</b>
147	<code>_POSIX_TRACE</code> .....	18, 25, 401	<code>AIO_ALLDONE</code> .....	202
148	<code>_POSIX_TRACE_EVENT_FILTER</code> .....	18, 25, 401	<code>AIO_CANCELED</code> .....	202
149	<code>_POSIX_TRACE_EVENT_NAME_MAX</code> .....	248, 253	<code>AIO_LISTIO_MAX</code> .....	246

150	AIO_MAX	246	basename	38
151	AIO_NOTCANCELED	202	basic regular expression	38, 167
152	AIO_PRIO_DELTA_MAX	246	batch access list	38
153	AI_CANONNAME	275	batch administrator	38
154	AI_NUMERICHOST	275	batch client	38
155	AI_PASSIVE	275	batch destination	39
156	alert	34	batch destination identifier	39
157	alert character	34	batch directive	39
158	alias name	34	batch job	39
159	alignment	34	batch job attribute	39
160	alternate file access control mechanism	34	batch job identifier	39
161	alternate signal stack	34	batch job name	39
162	ALT_DIGITS	242	batch job owner	40
163	AM_STR	242	batch job priority	40
164	anchoring	171	batch job state	40
165	ancillary data	35	batch name service	40
166	angle brackets	35	batch name space	40
167	ANYMARK	334	batch node	40
168	API	35	batch operator	40
169	application	35	batch queue	40
170	application address	35	batch queue attribute	41
171	application conformance	28	batch queue position	41
172	application program interface	35	batch queue priority	41
173	appropriate privileges	35	batch rerunability	41
174	AREGTYPE	376	batch restart	41
175	argument	35	batch server	41
176	ARG_MAX	246	batch server name	41
177	arm (a timer)	36	batch service	42
178	asterisk	36	batch service request	42
179	async-cancel-safe function	36	batch submission	42
180	async-signal-safe function	36	batch system	42
181	asynchronous events	36	batch target user	42
182	asynchronous I/O completion	37	batch user	42
183	asynchronous I/O operation	36	baud rate selection	380
184	asynchronous input and output	36	BC_BASE_MAX	250
185	asynchronously-generated signal	36	BC_DIM_MAX	250
186	ATEXIT_MAX	246	BC_SCALE_MAX	250
187	attribute selection	381	BC_STRING_MAX	250
188	authentication	37	BE	7
189	authorization	37	bind	42
190	background job	37	blank character	42
191	background process	37	blank line	43
192	background process group	37	blkcnt_t	366
193	backquote	37	blksize_t	366
194	BACKREF	175	BLKTYPE	376
195	backslash	37	block special file	43
196	backspace character	37	block-mode terminal	43
197	bandinfo	331	blocked process (or thread)	43
198	BAR	7	blocking	43
199	barrier	38	BOOT_TIME	419
200	base character	38	braces	43

## Index

201	brackets.....	43	CLOCK_PROCESS_CPUTIME_ID.....	388
202	BRE (ERE) matching a single character .....	166	CLOCK_REALTIME.....	251, 388
203	BRE (ERE) matching multiple characters.....	166	clock_t.....	366
204	BRKINT .....	379	CLOCK_THREAD_CPUTIME_ID.....	388
205	broadcast .....	43	CMMSG_DATA .....	352
206	BSD .....	212	CMMSG_FIRSTHDR.....	352
207	BSDLY .....	380	CMMSG_NXTHDR .....	352
208	BSn.....	380	coded character set.....	46
209	BUFSIZ.....	320	codeset .....	46
210	built-in.....	44	CODESET .....	242
211	built-in utility .....	44	collating element.....	46
212	BUS_ADRALN.....	304	collation .....	46
213	BUS_ADRERR.....	304	collation sequence .....	46
214	BUS_OBJERR.....	304	COLL_ELEM_MULTI.....	175
215	byte .....	44	COLL_ELEM_SINGLE.....	175
216	byte input/output functions .....	44	COLL_WEIGHTS_MAX .....	250
217	can.....	5	column position .....	47
218	canonical mode input processing .....	185	COLUMNS.....	161
219	carriage-return character.....	44	command.....	47
220	CD .....	7	command language interpreter .....	47
221	character .....	44	complex .....	206
222	character array.....	45	composite graphic symbol.....	47
223	character class .....	45	concurrent execution .....	95
224	character encoding.....	114	condition variable.....	47
225	state-dependent .....	118	conformance .....	15, 28
226	character set.....	45	POSIX.....	15
227	character special file.....	45	POSIX system interfaces .....	16
228	character string.....	45	XSI.....	15
229	CHARCLASS_NAME_MAX .....	250, 256	XSI system interfaces.....	19
230	charmap		conformance document.....	15
231	description .....	115	conforming application.....	15
232	CHAR_BIT .....	255	conforming implementation options .....	20
233	CHAR_MAX.....	255	connection .....	48
234	CHAR_MIN.....	256	connection mode.....	48
235	child process .....	45	connectionless mode.....	48
236	CHILD_MAX.....	246	control character .....	48
237	CHRTYPE .....	376	control modes.....	381
238	circumflex.....	45	control operator .....	48
239	CLD_CONTINUED .....	304	controlling process .....	48
240	CLD_DUMPED.....	304	controlling terminal .....	48, 184
241	CLD_EXITED .....	304	CONTTYTYPE .....	376
242	CLD_KILLED .....	304	conversion descriptor .....	49
243	CLD_STOPPED.....	304	core file.....	49
244	CLD_TRAPPED .....	304	CPT .....	7
245	CLOCAL.....	381	CPU .....	365
246	clock.....	45	CPU time .....	49
247	clock jump.....	46	clock.....	49
248	clock tick.....	46	timer .....	49
249	clockid_t .....	366	CRDLY .....	379
250	CLOCKS_PER_SEC .....	366, 388-389	CREAD .....	381
251	CLOCK_MONOTONIC .....	389	CRn.....	379

252	CRNCYSTR .....	242	mbstate_t.....	421
253	CS .....	7	msglen_t.....	341
254	CSIZE .....	381	msgqnum_t.....	341
255	CSn.....	381	nl_catd .....	283
256	CSTOPB.....	381	nl_item.....	283
257	current job .....	49	pid_t.....	300
258	current working directory .....	49, 92	ptrdiff_t .....	312
259	cursor position .....	49	regex_t .....	292
260	CX.....	7	regmatch_t.....	292
261	C_ constants in <cpio.h>.....	<b>209</b>	regoff_t .....	292
262	C_IRGRP .....	209	shmatt_t.....	349
263	C_IROTH .....	209	sigset_t.....	300
264	C_IRUSR.....	209	sig_atomic_t .....	300
265	C_ISBLK .....	209	size_t .....	312
266	C_ISCHR .....	209	speed_t.....	378
267	C_ISCTG.....	209	tflag_t.....	378
268	C_ISDIR .....	209	VISIT .....	296
269	C_ISFIFO .....	209	wchar_t.....	312
270	C_ISGID.....	209	wctrans_t.....	425
271	C_ISLNK .....	209	wctype_t.....	421
272	C_ISREG .....	209	wint_t.....	421
273	C_ISSOCK.....	209	data types	
274	C_ISUID.....	209	defined in <fenv.h>.....	<b>222</b>
275	C_ISVTX.....	209	defined in <sys/types.h>.....	366
276	C_IWGRP .....	209	DATEMSK.....	<b>161</b>
277	C_IWOTH.....	209	DAY_ .....	242
278	C_IWUSR .....	209	DBL_ constants	
279	C_IXGRP .....	209	defined in <float.h>.....	<b>226</b>
280	C_IXOTH .....	209	DBL_DIG.....	226, 255
281	C_IXUSR.....	209	DBL_EPSILON.....	227
282	data segment.....	50	DBL_MANT_DIG.....	226
283	data structure		DBL_MAX.....	227, 255
284	dirent .....	212	DBL_MAX_10_EXP.....	227
285	entry .....	296	DBL_MAX_EXP .....	227
286	group.....	236	DBL_MIN .....	228
287	lconv.....	260	DBL_MIN_10_EXP.....	227
288	msqid_ds.....	341	DBL_MIN_EXP .....	227
289	stat .....	356	DBM.....	272
290	tms.....	365	DBM_INSERT .....	272
291	utimbuf.....	418	DBM_REPLACE .....	272
292	data type		DEAD_PROCESS .....	419
293	ACTION .....	296	DECIMAL_DIG.....	226
294	cc_t.....	378	deferred batch service.....	50
295	DIR.....	212	DELAYTIMER_MAX .....	246
296	div_t .....	324	device .....	50
297	ENTRY.....	296	output .....	181
298	FILE .....	321	device ID.....	50
299	fpos_t .....	321	dev_t.....	366
300	glob_t .....	234	DIR.....	212
301	ldiv_t .....	324	directory .....	50
302	lldiv_t.....	324	directory entry (or link) .....	50

## Index

303	directory protection .....	95	EFAULT .....	215
304	directory stream .....	50	EFBIG .....	215
305	dirent structure .....	212	effective group ID .....	52
306	DIRTYPE .....	376	effective user ID .....	52
307	disarm (a timer) .....	50	EHOSTUNREACH .....	215
308	display .....	50	EIDRM .....	215
309	display line .....	51	eight-bit transparency .....	52
310	documentation .....	15	EILSEQ .....	215
311	dollar sign .....	51	EINPROGRESS .....	216
312	domain error .....	103	EINTR .....	216
313	dot .....	51	EINVAL .....	216
314	dot-dot .....	51	EIO .....	216
315	double-quote .....	51	EISCONN .....	216
316	downshifting .....	51	EISDIR .....	216
317	driver .....	51	ELOOP .....	216
318	DUP_COUNT .....	175	EMFILE .....	216
319	D_FMT .....	242	EMLINK .....	216
320	D_T_FMT .....	242	EMPTY .....	419
321	E2BIG .....	215	empty directory .....	52
322	EACCES .....	215	empty line .....	52
323	EADDRINUSE .....	215	empty string (or null string) .....	52
324	EADDRNOTAVAIL .....	215	empty wide-character string .....	52
325	EAFNOSUPPORT .....	215	EMSGSIZE .....	216
326	EAGAIN .....	215	EMULTIHOP .....	216
327	EAI_AGAIN .....	276	ENAMETOOLONG .....	216
328	EAI_BADFLAGS .....	276	encoding	
329	EAI_FAIL .....	276	character .....	114
330	EAI_FAMILY .....	276	encoding rule .....	52
331	EAI_MEMORY .....	276	encryption .....	21
332	EAI_NONAME .....	276	ENETDOWN .....	216
333	EAI_SERVICE .....	276	ENETUNREACH .....	216
334	EAI_SOCKTYPE .....	276	ENFILE .....	216
335	EAI_SYSTEM .....	276	ENOBUFS .....	216
336	EALREADY .....	215	ENODATA .....	216
337	EBADF .....	215	ENODEV .....	216
338	EBADMSG .....	215	ENOENT .....	216
339	EBUSY .....	215	ENOEXEC .....	216
340	ECANCELED .....	215	ENOLCK .....	216
341	ECHILD .....	215	ENOLINK .....	216
342	ECHO .....	381	ENOMEM .....	216
343	ECHOE .....	381	ENOMSG .....	216
344	ECHOK .....	381	ENOPROTOPT .....	216
345	ECHONL .....	381	ENOSPC .....	216
346	ECONNABORTED .....	215	ENOSR .....	216
347	ECONNREFUSED .....	215	ENOSTR .....	216
348	ECONNRESET .....	215	ENOSYS .....	216
349	EDEADLK .....	215	ENOTCONN .....	216
350	EDESTADDRREQ .....	215	ENOTDIR .....	216
351	EDOM .....	215	ENOTEMPTY .....	216
352	EDQUOT .....	215	ENOTSOCK .....	216
353	EEXIST .....	215	ENOTSUP .....	216

354	ENOTTY.....	217	FD_CLR.....	345
355	entire regular expression.....	52, 165	FD_ISSET.....	345
356	environment variables		FD_SET.....	345
357	internationalization.....	158	fd_set.....	<b>345</b> , 362
358	ENXIO.....	217	FD_SETSIZE.....	345
359	EOF.....	320	FD_ZERO.....	345
360	EOPNOTSUPP.....	217	feature test macro.....	54
361	EOVERFLOW.....	217	fenv_t.....	222
362	EPERM.....	217	fexcept_t.....	222
363	EPIPE.....	217	FE_constants	
364	epoch.....	53	defined in <fenv.h>.....	<b>222</b>
365	EPROTO.....	217	FE_ALL_EXCEPT.....	222
366	EPROTONOSUPPORT.....	217	FE_DFL_ENV.....	223
367	EPROTOTYPE.....	217	FE_DIVBYZERO.....	222
368	equivalence class.....	53	FE_DOWNWARD.....	222
369	era.....	53	FE_INEXACT.....	222
370	ERA.....	242	FE_INVALID.....	222
371	ERANGE.....	217	FE_OVERFLOW.....	222
372	ERA_D_FMT.....	242	FE_TONEAREST.....	222
373	ERA_D_T_FMT.....	242	FE_TOWARDZERO.....	222
374	ERA_T_FMT.....	242	FE_UNDERFLOW.....	222
375	EROFS.....	217	FE_UPWARD.....	222
376	error conditions		FFDLY.....	380
377	mathematical functions.....	103	FFn.....	380
378	ESPIPE.....	217	field.....	54
379	ESRCH.....	217	FIFO.....	55
380	ESTALE.....	217	FIFO special file.....	55
381	ETIME.....	217	FIFOTYPE.....	376
382	ETIMEDOUT.....	217	file.....	55
383	ETXTBSY.....	217	FILE.....	320, 421
384	event management.....	53	file access permissions.....	95
385	EWOLDBLOCK.....	217	file characteristics	
386	EXDEV.....	217	data structure.....	358
387	executable file.....	53	header.....	358
388	execute.....	53	file description.....	55
389	execution time.....	49, 54	file descriptor.....	55
390	measurement.....	97	file group class.....	55
391	monitoring.....	54	file hierarchy.....	96
392	EXIT_FAILURE.....	324	file mode.....	55
393	EXIT_SUCCESS.....	324	file mode bits.....	56
394	expand.....	54	file offset.....	56
395	EXPR_NEST_MAX.....	250	file other class.....	56
396	extended regular expression.....	54, 171	file owner class.....	56
397	extended security controls.....	54, 95	file permission bits.....	56
398	extension		file serial number.....	57
399	CX.....	7	file system.....	57
400	OH.....	9	file times update.....	96
401	XSI.....	13	file type.....	57
402	F-LOCK.....	<b>408</b>	filename.....	56
403	FD.....	7	filename portability.....	56
404	FD_CLOEXEC.....	219	FILENAME_MAX.....	320

## Index

405	FILESIZEBITS.....	248	FTW_constants	
406	filter .....	57	in <ftw.h>.....	<b>232</b>
407	FIPS.....	16	FTW_CHDIR.....	232
408	first open (of a file) .....	57	FTW_D.....	232
409	flow control.....	57	FTW_DEPTH.....	232
410	FLT_constants		FTW_DNR.....	232
411	defined in <float.h>.....	<b>226</b>	FTW_DP.....	232
412	FLT_DIG.....	226, 255	FTW_F.....	232
413	FLT_EPSILON.....	227	FTW_MOUNT.....	232
414	FLT_EVAL_METHOD.....	225	FTW_NS.....	232
415	FLT_MANT_DIG.....	226	FTW_PHYS.....	232
416	FLT_MAX.....	227, 255	FTW_SL.....	232
417	FLT_MAX_10_EXP.....	227	FTW_SLN.....	232
418	FLT_MAX_EXP.....	227	F_DUPFD.....	219
419	FLT_MIN.....	228	F_GETFD.....	219
420	FLT_MIN_10_EXP.....	227	F_GETFL.....	219
421	FLT_MIN_EXP.....	227	F_GETLK.....	219
422	FLT_RADIX.....	226	F_GETOWN.....	219
423	FLT_ROUNDS.....	225	F_OK.....	404
424	FLUSHR.....	333	F_RDLCK.....	219
425	FLUSHRW.....	333	F_SETFD.....	219
426	FLUSHW.....	333	F_SETFL.....	219
427	FMNAMESZ.....	332-333	F_SETLK.....	219
428	FNM_constants		F_SETLKW.....	219
429	in <fnmatch.h>.....	<b>231</b>	F_SETOWN.....	219
430	FNM_NOESCAPE.....	231	F_TEST.....	<b>408</b>
431	FNM_NOMATCH.....	231	F_TLOCK.....	<b>408</b>
432	FNM_NOSYS.....	231	F_ULOCK.....	<b>408</b>
433	FNM_PATHNAME.....	231	F_UNLCK.....	219
434	FNM_PERIOD.....	231	F_WRLCK.....	219
435	FOPEN_MAX.....	248, 320	GETALL.....	347
436	foreground job.....	57	GETNCNT.....	347
437	foreground process.....	57	GETPID.....	347
438	foreground process group.....	57	GETVAL.....	347
439	foreground process group ID.....	58	GETZCNT.....	347
440	form-feed character.....	58	gid_t.....	366
441	format of entries.....	<b>201</b>	GLOB_constants	
442	FPE_FLTDIV.....	304	defined in <glob.h>.....	<b>234</b>
443	FPE_FLTINV.....	304	GLOB_ABORTED.....	234
444	FPE_FLTOVF.....	304	GLOB_APPEND.....	234
445	FPE_FLTRES.....	304	GLOB_DOOFFS.....	234
446	FPE_FLTSUB.....	304	GLOB_ERR.....	234
447	FPE_FLTUND.....	304	GLOB_MARK.....	234
448	FPE_INTDIV.....	304	GLOB_NOCHECK.....	234
449	FPE_INTOVF.....	304	GLOB_NOESCAPE.....	234
450	FR.....	7	GLOB_NOMATCH.....	234
451	fsblkcnt_t.....	366	GLOB_NOSORT.....	234
452	FSC.....	<b>8</b>	GLOB_NOSPACE.....	234
453	fsfilcnt_t.....	366	GLOB_NOSYS.....	234
454	FTW.....	232	grammar	
455			locale.....	149

456	regular expression.....	175	Inf.....	59
457	graphic character.....	58	INFINITY.....	263
458	group database.....	58	INIT_PROCESS.....	419
459	group ID.....	58	INLCR.....	379
460	group name.....	58	ino_t.....	366
461	hard limit.....	59	INPCK.....	379
462	hard link.....	59	instrumented application.....	59
463	headers.....	<b>201</b>	interactive shell.....	59
464	HOME.....	<b>161</b>	internationalization.....	60
465	home directory.....	59	interprocess communication.....	60
466	host byte order.....	59	INTMAX_MAX.....	317
467	HUGE_VAL.....	263	INTMAX_MIN.....	317
468	HUGE_VALF.....	263	INTN_MAX.....	316
469	HUGE_VALL.....	263	INTN_MIN.....	316
470	HUPCL.....	381	INTPTR_MAX.....	316
471	I.....	206	INTPTR_MIN.....	316
472	ICANON.....	381	INT_FASTN_MAX.....	316
473	ICRNL.....	379	INT_FASTN_MIN.....	316
474	idtype_t.....	<b>372</b>	INT_LEASTN_MAX.....	316
475	id_t.....	366	INT_LEASTN_MIN.....	316
476	IEXTEN.....	381	INT_MAX.....	255
477	IGNBRK.....	379	INT_MIN.....	256
478	IGNCR.....	379	invalid.....	166
479	IGNPAR.....	379	invariant values.....	256
480	ILL_BADSTK.....	304	invoke.....	60
481	ILL_COPROC.....	304	iovec.....	<b>369</b>
482	ILL_ILLADR.....	304	IOV_MAX.....	246, 369
483	ILL_ILLOPC.....	304	IP6.....	<b>8</b>
484	ILL_ILLOPN.....	304	IPC.....	336
485	ILL_ILLTRP.....	304	IPC_constants	
486	ILL_PRVOPC.....	304	defined in <sys/ipc.h>.....	<b>336</b>
487	ILL_PRVREG.....	304	IPC_CREAT.....	336
488	imaginary.....	206	IPC_EXCL.....	336
489	implementation-defined.....	5	IPC_NOWAIT.....	336
490	IN6_IS_ADDR_LINKLOCAL.....	280	IPC_PRIVATE.....	336
491	IN6_IS_ADDR_LOOPBACK.....	280	IPC_RMID.....	336
492	IN6_IS_ADDR_MC_GLOBAL.....	280	IPC_SET.....	336
493	IN6_IS_ADDR_MC_LINKLOCAL.....	280	IPC_STAT.....	336
494	IN6_IS_ADDR_MC_NODELOCAL.....	280	IPPROTO_ICMP.....	279
495	IN6_IS_ADDR_MC_ORGLOCAL.....	280	IPPROTO_IP.....	279
496	IN6_IS_ADDR_MC_SITELOCAL.....	280	IPPROTO_IPV6.....	279
497	IN6_IS_ADDR_MULTICAST.....	280	IPPROTO_RAW.....	279
498	IN6_IS_ADDR_SITELOCAL.....	280	IPPROTO_TCP.....	279
499	IN6_IS_ADDR_UNSPECIFIED.....	280	IPPROTO_UDP.....	279
500	IN6_IS_ADDR_V4COMPAT.....	280	IPV6_JOIN_GROUP.....	279
501	IN6_IS_ADDR_V4MAPPED.....	280	IPV6_LEAVE_GROUP.....	279
502	INADDR_ANY.....	279	IPV6_MULTICAST_HOPS.....	279
503	INADDR_BROADCAST.....	279	IPV6_MULTICAST_IF.....	280
504	incomplete line.....	59	IPV6_MULTICAST_LOOP.....	280
505	INET6_ADDRSTRLEN.....	279	IPV6_UNICAST_HOPS.....	280
506	INET_ADDRSTRLEN.....	279	ISIG.....	381

## Index

507	ISOC standard.....	206	link.....	61
508	ISTRIP.....	379	link count.....	61
509	itimerval.....	<b>362</b>	LINK_MAX.....	249
510	ITIMER_PROF.....	362	LIO_NOP.....	202
511	ITIMER_REAL.....	362	LIO_NOWAIT.....	202
512	ITIMER_VIRTUAL.....	362	LIO_READ.....	202
513	IXANY.....	379	LIO_WAIT.....	202
514	IXOFF.....	379	LIO_WRITE.....	202
515	IXON.....	379	LLONG_MAX.....	256
516	I_LOOK.....	332	LLONG_MIN.....	256
517	I_POP.....	332	LNKTYPE.....	376
518	I_PUSH.....	332	local customs.....	61
519	job.....	60	local IPC.....	61
520	job control.....	60	local modes.....	381
521	job control job ID.....	60	locale.....	61, 119
522	key_t.....	366	grammar.....	149
523	LANG.....	<b>158</b>	POSIX.....	120
524	last close (of a file).....	61	locale definition.....	120
525	LASTMARK.....	334	localization.....	62
526	LC_ALL.....	<b>159, 260</b>	login.....	62
527	LC_COLLATE.....	<b>159, 250, 260</b>	login name.....	62
528	description.....	130	LOGIN_NAME_MAX.....	246
529	LC_CTYPE.....	<b>159, 242, 260, 425</b>	LOGIN_PROCESS.....	419
530	description.....	122	LOGNAME.....	<b>162</b>
531	LC_MESSAGES.....	<b>159, 242, 260, 283</b>	LOG_ALERT.....	375
532	description.....	148	LOG_AUTH.....	374
533	LC_MONETARY.....	<b>159, 242, 260</b>	LOG_CONS.....	374
534	description.....	138	LOG_CRIT.....	375
535	LC_NUMERIC.....	<b>159, 242, 260</b>	LOG_CRON.....	374
536	description.....	141	LOG_DAEMON.....	374
537	LC_TIME.....	<b>159, 242, 260</b>	LOG_DEBUG.....	375
538	description.....	142	LOG_EMERG.....	375
539	LDBL_constants		LOG_ERR.....	375
540	defined in <float.h>.....	<b>226</b>	LOG_INFO.....	375
541	LDBL_DIG.....	227	LOG_KERN.....	374
542	LDBL_EPSILON.....	228	LOG_LOCAL.....	374
543	LDBL_MANT_DIG.....	226	LOG_LPR.....	374
544	LDBL_MAX.....	227	LOG_MAIL.....	374
545	LDBL_MAX_10_EXP.....	227	LOG_MASK.....	374
546	LDBL_MAX_EXP.....	227	LOG_NDELAY.....	374
547	LDBL_MIN.....	228	LOG_NEWS.....	374
548	LDBL_MIN_10_EXP.....	227	LOG_NOTICE.....	375
549	LDBL_MIN_EXP.....	227	LOG_NOWAIT.....	374
550	legacy.....	5, 25	LOG_ODELAY.....	374
551	limit		LOG_PID.....	374
552	numerical.....	255	LOG_USER.....	374
553	line.....	61	LOG_UUCP.....	374
554	line control.....	382	LOG_WARNING.....	375
555	LINES.....	<b>161</b>	LONG_BIT.....	255
556	LINE_MAX.....	250	LONG_MAX.....	255
557	linger.....	61	LONG_MIN.....	256

558	lower multiplexing.....	82	MM_HALT.....	229
559	L_ANCHOR.....	175	MM_HARD.....	229
560	L_ctermid.....	320	MM_INFO.....	229
561	L_tmpnam.....	320	MM_NOCON.....	230
562	MAGIC.....	209	MM_NOMSG.....	229
563	map.....	62	MM_NOSEV.....	229
564	MAP_FIXED.....	338	MM_NOTOK.....	229
565	MAP_PRIVATE.....	338	MM_NRECOV.....	229
566	MAP_SHARED.....	338	MM_NULLACT.....	229
567	margin codes		MM_NULLLBL.....	229
568	notation.....	14	MM_NULLMC.....	229
569	marked message.....	62	MM_NULLSEV.....	229
570	matched.....	62, 165	MM_NULLTAG.....	229
571	mathematical functions		MM_NULLTXT.....	229
572	domain error.....	103	MM_OK.....	229
573	error conditions.....	103	MM_OPSYS.....	229
574	NaN arguments.....	104	MM_PRINT.....	229
575	pole error.....	104	MM_RECOVER.....	229
576	range error.....	104	MM_SOFT.....	229
577	MAXARGS.....	310	MM_UTIL.....	229
578	MAXFLOAT.....	263	MM_WARNING.....	229
579	maximum values.....	251	mode.....	63
580	MAX_CANON.....	249	mode_t.....	366
581	MAX_INPUT.....	249	MON.....	<b>9</b>
582	may.....	5	monotonic clock.....	64
583	MB_CUR_MAX.....	324	MON.....	242
584	MB_LEN_MAX.....	255	MORECTL.....	334
585	MC1.....	<b>8</b>	MOREDATA.....	334
586	MC2.....	<b>8</b>	mount point.....	64
587	MCL_CURRENT.....	338	MPR.....	<b>9</b>
588	MCL_FUTURE.....	338	MQ_OPEN_MAX.....	246
589	mcontext_t.....	<b>396</b>	MQ_PRIO_MAX.....	247
590	memory mapped files.....	62	MSG.....	<b>9</b>
591	memory object.....	63	MSGVERB.....	<b>162</b>
592	memory synchronization.....	98	MSG_ANY.....	334
593	memory-resident.....	63	MSG_BAND.....	334
594	message.....	63	MSG_CTRUNC.....	353
595	message catalog.....	63	MSG_DONTRROUTE.....	353
596	message catalog descriptor.....	63	MSG_EOR.....	353
597	message queue.....	63	MSG_HIPRI.....	334
598	META_CHAR.....	175	MSG_NOERROR.....	341
599	MF.....	<b>8</b>	MSG_OOB.....	353
600	minimum values.....	251	MSG_PEEK.....	353
601	MINSIGSTKSZ.....	302	MSG_TRUNC.....	353
602	ML.....	<b>8</b>	MSG_WAITALL.....	353
603	MLR.....	<b>9</b>	MS_ASYNC.....	338
604	MM_macros.....	<b>229</b>	MS_INVALIDATE.....	338
605	MM_APPL.....	229	MS_SYNC.....	338
606	MM_CONSOLE.....	229	multi-character collating element.....	64
607	MM_ERROR.....	229	mutex.....	64
608	MM_FIRM.....	229	MUXID_ALL.....	334

## Index

609	MX	9	NUL	66
610	M_	262	NULL	312, 320, 324, 328, 388, 404
611	M_E	262	null byte	66
612	M_LN	262	null pointer	66
613	M_LOG10E	262	null string	66
614	M_LOG2E	262	null wide-character code	66
615	M_PI	262	number sign	66
616	M_SQRT1_2	263	numerical limits	255
617	M_SQRT2	263	NZERO	257
618	name	64	OB	9
619	named STREAM	64	object file	66
620	NAME_MAX	98, 212, 249	OCRNL	379
621	NaN	225	octet	67
622	NAN	263	OF	9
623	NaN (Not a Number)	64	offset maximum	67
624	NaN arguments		off_t	366
625	mathematical functions	104	OFILL	379
626	native language	64	OH	9
627	NCCS	378	OLD_TIME	419
628	NDEBUG	205	ONLCR	379
629	negative response	65	ONLRET	379
630	network	65	ONOCR	379
631	network address	65	opaque address	67
632	network byte order	65	open file	67
633	newline character	65	open file description	67
634	NEW_TIME	419	OPEN_MAX	247, 298
635	NGROUPS_MAX	250	operand	67
636	nice value	65	operator	67
637	NI_DGRAM	276	OPOST	379
638	NI_NAMEREQD	276	option	67
639	NI_NOFQDN	275	ADV	6
640	NI_NUMERICHOST	275	AIO	6
641	NI_NUMERICSERV	276	BAR	7
642	NLDLY	379	BE	7
643	nlink_t	366	CD	7
644	NLn	379	CPT	7
645	NLSPATH	159	CS	7
646	NL_ARGMAX	256	FD	7
647	NL_CAT_LOCALE	283	FR	7
648	NL_LANGMAX	256	FSC	8
649	NL_MSGMAX	257	IP6	8
650	NL_NMAX	257	MC1	8
651	NL_SETD	283	MC2	8
652	NL_SETMAX	257	MF	8
653	NL_TEXTMAX	257	ML	8
654	NOEXPR	242	MLR	9
655	NOFLSH	381	MON	9
656	non-blocking	65	MPR	9
657	non-canonical mode input processing	186	MSG	9
658	non-spacing characters	66	MX	9
659	NOSTR	242	PIO	10

660	PS	10	O_WRONLY	220
661	RS	10	page	68
662	RTS	10	page size	68
663	SD	10	PAGESIZE	247
664	SEM	10	PAGE_SIZE	247
665	SHM	10	parameter	68
666	SIO	10	PARENB	381
667	SPI	11	parent directory	68
668	SPN	11	parent process	69
669	SS	11	parent process ID	69
670	TCT	11	PARMRK	379
671	TEF	11	PARODD	381
672	THR	11	PATH	162
673	TMO	11	path prefix	69
674	TMR	12	pathname	69
675	TPI	12	pathname component	69
676	TPP	12	pathname resolution	98
677	TPS	12	PATH_MAX	249, 257
678	TRC	12	pattern	69
679	TRI	12	period	69
680	TRL	12	permissions	70
681	TSA	12	persistence	70
682	TSF	13	pid_t	366
683	TSH	13	PIO	10
684	TSP	13	pipe	70
685	TSS	13	PIPE_BUF	249
686	TYM	13	PM_STR	242
687	UP	13	pole error	104
688	XSR	14	POLLERR	284
689	option-argument	67	pollfd	284
690	options		POLLHUP	284
691	shell and utilities	26	POLLIN	284
692	system interfaces	26	polling	70
693	ORD_CHAR	175	POLLNVAL	284
694	orientation	68	POLLOUT	284
695	orphaned process group	68	POLLPRI	284
696	output devices	181	POLLRDBAND	284
697	O_constants		POLLRDNORM	284
698	defined in <fcntl.h>	219-220	POLLWRBAND	284
699	O_ACCMODE	220	POLLWRNORM	284
700	O_APPEND	219	POLL_ERR	304
701	O_CREAT	219	POLL_HUP	304
702	O_DSYNC	219	POLL_IN	304
703	O_EXCL	219	POLL_MSG	304
704	O_NOCTTY	219	POLL_OUT	304
705	O_NONBLOCK	219	POLL_PRI	304
706	O_RDONLY	220	portable character set	70, 111
707	O_RDWR	220	portable filename character set	70, 96
708	O_RSYNC	220	positional parameter	70
709	O_SYNC	220	POSIX	
710	O_TRUNC	219	conformance	15

## Index

711	POSIX locale .....	120	PRIO_PROCESS.....	343
712	POSIX shell and utilities.....	18	PRIO_USER .....	343
713	POSIX system interfaces		privilege.....	72
714	conformance.....	16	process .....	72
715	POSIX2_CHAR_TERM .....	18, 27	process group .....	72
716	POSIX2_C_DEV.....	18, 26	process group ID.....	72
717	POSIX2_FORT_DEV.....	18, 27	process group leader.....	72
718	POSIX2_FORT_RUN .....	18, 27	process group lifetime .....	73
719	POSIX2_LOCALEDEF.....	18, 27	process groups	
720	POSIX2_PBS .....	18, 27	termios.....	183
721	POSIX2_PBS_ACCOUNTING .....	18, 27	process ID.....	73
722	POSIX2_PBS_CHECKPOINT .....	27	process ID reuse.....	99
723	POSIX2_PBS_LOCATE.....	18, 27	process lifetime .....	73
724	POSIX2_PBS_MESSAGE.....	18, 27	process memory locking.....	73
725	POSIX2_PBS_TRACK.....	18, 27	process termination.....	73
726	POSIX2_SW_DEV .....	18, 27	process virtual time.....	74
727	POSIX2_UPE .....	18, 28	process-to-process communication.....	74
728	POSIX_ALLOC_SIZE_MIN .....	249	program .....	74
729	POSIX_FADV_DONTNEED.....	220	protocol.....	74
730	POSIX_FADV_NOREUSE.....	220	PROT_EXEC .....	338
731	POSIX_FADV_NORMAL.....	220	PROT_NONE.....	338
732	POSIX_FADV_RANDOM .....	220	PROT_READ .....	338
733	POSIX_FADV_SEQUENTIAL .....	220	PROT_READ constants	
734	POSIX_FADV_WILLNEED.....	220	in <sys/mman.h>.....	<b>338</b>
735	POSIX_MADV_DONTNEED.....	339	PROT_WRITE .....	338
736	POSIX_MADV_NORMAL.....	338	PS.....	<b>10</b>
737	POSIX_MADV_RANDOM.....	339	pseudo-terminal.....	74
738	POSIX_MADV_SEQUENTIAL.....	339	PTHREAD_BARRIER_SERIAL_THREAD .....	286
739	POSIX_MADV_WILLNEED.....	339	PTHREAD_CANCELED .....	286
740	POSIX_REC_INCR_XFER_SIZE .....	249	PTHREAD_CANCEL_ASYNCHRONOUS .....	286
741	POSIX_REC_MAX_XFER_SIZE.....	249	PTHREAD_CANCEL_DEFERRED .....	286
742	POSIX_REC_MIN_XFER_SIZE.....	249	PTHREAD_CANCEL_DISABLE .....	286
743	POSIX_REC_XFER_ALIGN .....	249	PTHREAD_CANCEL_ENABLE.....	286
744	POSIX_TYPED_MEM_ALLOCATE .....	339	PTHREAD_COND_INITIALIZER .....	286
745	POSIX_TYPED_MEM_ALLOCATE_CONTIG.....		PTHREAD_CREATE_DETACHED.....	286
746	.....	339	PTHREAD_CREATE_JOINABLE .....	286
747	POSIX_TYPED_MEM_MAP_ALLOCATABLE.....		PTHREAD_DESTRUCTOR_ITERATIONS .....	247
748	.....	339	PTHREAD_EXPLICIT_SCHED.....	286
749	preallocation.....	71	PTHREAD_INHERIT_SCHED.....	286
750	preempted process (or thread) .....	71	PTHREAD_KEYS_MAX .....	247
751	previous job .....	71	PTHREAD_MUTEX_DEFAULT.....	286
752	printable character.....	71	PTHREAD_MUTEX_ERRORCHECK .....	286
753	printable file.....	71	PTHREAD_MUTEX_INITIALIZER .....	286
754	priority .....	71	PTHREAD_MUTEX_NORMAL .....	286
755	priority band.....	71	PTHREAD_MUTEX_RECURSIVE .....	286
756	priority inversion.....	72	PTHREAD_ONCE_INIT .....	286
757	priority scheduling.....	72	PTHREAD_PRIO_INHERIT .....	286
758	priority-based scheduling.....	72	PTHREAD_PRIO_NONE.....	286
759	PRIO_ constants		PTHREAD_PRIO_PROTECT .....	286
760	defined in <sys/resource.h> .....	<b>343</b>	PTHREAD_PROCESS_PRIVATE.....	286
761	PRIO_PGRP .....	343	PTHREAD_PROCESS_SHARED.....	286

762	PTHREAD_RWLOCK_INITIALIZER	286	REG_ECTYPE	292
763	PTHREAD_SCOPE_PROCESS	286	REG_EESCAPE	292
764	PTHREAD_SCOPE_SYSTEM	286	REG_ENOSYS	293
765	PTHREAD_STACK_MIN	247	REG_EPAREN	292
766	PTHREAD_THREADS_MAX	247	REG_ERANGE	293
767	PTRDIFF_MAX	317	REG_ESPACE	293
768	PTRDIFF_MIN	317	REG_ESUBREG	292
769	PWD	162	REG_EXTENDED	292
770	P_ALL	372	REG_ICASE	292
771	P_GID	372	REG_NEWLINE	292
772	P_PID	372	REG_NOMATCH	292
773	P_tmpdir	320	REG_NOSUB	292
774	quiet NaN	225	REG_NOTBOL	292
775	QUOTED_CHAR	175	REG_NOTEOL	292
776	radix character	74	relative pathname	76, 98
777	RADIXCHAR	242	relocatable file	76
778	RAND_MAX	324	relocation	76
779	range error	104	requested batch service	76
780	result overflows	104	requirements	15
781	result underflows	104	result overflows	104
782	read-only file system	74	result underflows	104
783	read-write lock	74	RE_DUP_MAX	247, 250
784	real group ID	75	rlimit	343
785	real time	75	RLIMIT_AS	344
786	real user ID	75	RLIMIT_CORE	343
787	realtime	22	RLIMIT_CPU	343
788	REALTIME	202, 270, 294, 298	RLIMIT_DATA	343
789	realtime signal extension	75	RLIMIT_FSIZE	343
790	realtime threads	23	RLIMIT_NOFILE	344
791	REALTIME_THREADS	24	RLIMIT_STACK	344
792	record	75	RLIM_INFINITY	343
793	redirection	75	RLIM_SAVED_CUR	343
794	redirection operator	75	RLIM_SAVED_MAX	343
795	reentrant function	75	RMSGD	333
796	referenced shared memory object	76	RMSGN	333
797	refresh	76	RNORM	333
798	region	76	root directory	77
799	REGTYPE	376	RPROTDAT	333
800	regular expression	76	RPROTDIS	333
801	basic	167	RPROTNORM	333
802	extended	171	RS	10
803	grammar	175	RS_HIPRI	333
804	regular file	76	RTLD_GLOBAL	214
805	REG_constants		RTLD_LAZY	214
806	defined in <regex.h>	292	RTLD_LOCAL	214
807	REG_BADBR	292	RTLD_NOW	214
808	REG_BADPAT	292	RTS	10
809	REG_BADRPT	293	RTSIG_MAX	247, 301
810	REG_EBRACE	292	runnable process (or thread)	77
811	REG_EBRACK	292	running process (or thread)	77
812	REG_ECOLLATE	292		

## Index

813	runtime values		
814	increasable .....	250	
815	invariant .....	246	
816	rusage .....	<b>343</b>	
817	RUSAGE_CHILDREN .....	343	
818	RUSAGE_SELF .....	343	
819	R_ANCHOR .....	175	
820	R_OK .....	404	
821	saved resource limits .....	77	
822	saved set-group-ID .....	77	
823	saved set-user-ID .....	77	
824	SA_ constants		
825	declared in <signal.h> .....	302	
826	SA_NOCLDSTOP .....	302	
827	SA_NOCLDWAIT .....	302	
828	SA_NODEFER .....	302	
829	SA_ONSTACK .....	302	
830	SA_RESETHAND .....	302	
831	SA_RESTART .....	302	
832	SA_SIGINFO .....	302	
833	SCHAR_MAX .....	255	
834	SCHAR_MIN .....	256	
835	scheduling .....	77	
836	scheduling allocation domain .....	78	
837	scheduling contention scope .....	78	
838	scheduling policy .....	78, 99	
839	SCHED_FIFO .....	294	
840	SCHED_OTHER .....	294	
841	SCHED_RR .....	294	
842	SCHED_SPORADIC .....	294	
843	SCM_RIGHTS .....	352	
844	screen .....	78	
845	scroll .....	78	
846	SD .....	<b>10</b>	
847	seconds since the Epoch .....	100	
848	SEEK_CUR .....	219, 320, 408	
849	SEEK_END .....	219, 320, 408	
850	SEEK_SET .....	219, 320, 408	
851	SEGV_ACCERR .....	304	
852	SEGV_MAPERR .....	304	
853	SEM .....	<b>10</b>	
854	semaphore .....	78, 100	
855	semaphore lock operation .....	100	
856	semaphore unlock operation .....	101	
857	SEM_NSEMS_MAX .....	247	
858	SEM_UNDO .....	347	
859	SEM_VALUE_MAX .....	247	
860	session .....	78	
861	session leader .....	79	
862	session lifetime .....	79	
863	SETALL .....	347	
	SETVAL .....	347	
	shall .....	5	
	shared memory object .....	79	
	shell .....	79	
	SHELL .....	<b>162</b>	
	shell script .....	79	
	shell, the .....	79	
	SHM .....	<b>10</b>	
	SHMLBA .....	349	
	SHM_RDONLY .....	349	
	SHM_RND .....	349	
	should .....	5	
	SHRT_MAX .....	255	
	SHRT_MIN .....	256	
	SHUT_RD .....	353	
	SHUT_RDWR .....	354	
	SIGABRT .....	301	
	SIGALRM .....	301	
	SIGBUS .....	301, 304	
	SIGCHLD .....	301, 304	
	SIGCONT .....	301	
	SIGEV_NONE .....	300	
	SIGEV_SIGNAL .....	300	
	SIGEV_THREAD .....	300	
	SIGFPE .....	301, 304	
	SIGHUP .....	301	
	SIGILL .....	301, 304	
	siginfo_t .....	<b>303</b>	
	SIGINT .....	301	
	SIGKILL .....	301	
	signal .....	79	
	signal stack .....	80	
	signaling NaN .....	225	
	SIGPIPE .....	301	
	SIGPOLL .....	301, 304	
	SIGPROF .....	301	
	SIGQUEUE_MAX .....	247	
	SIGQUIT .....	301	
	SIGRTMAX .....	300	
	SIGRTMIN .....	300	
	SIGSEGV .....	301, 304	
	SIGSTKSZ .....	302	
	SIGSTOP .....	301	
	SIGSYS .....	301	
	SIGTERM .....	301	
	SIGTRAP .....	301, 304	
	SIGTSTP .....	301	
	SIGTTIN .....	301	
	SIGTTOU .....	301	
	SIGURG .....	301	
	SIGUSR1 .....	301	

864	SIGUSR2	301	special parameter	81
865	SIGVTALRM	301	SPEC_CHAR	175
866	SIGXCPU	301	SPI	11
867	SIGXFSZ	301	spin lock	81
868	SIG_ATOMIC_MAX	317	SPN	11
869	SIG_ATOMIC_MIN	317	sporadic server	81
870	SIG_BLOCK	302	SS	11
871	SIG_DFL	300	SSIZE_MAX	255, 367
872	SIG_ERR	300	ssize_t	366
873	SIG_HOLD	300	SS_DISABLE	302
874	SIG_IGN	300	SS_ONSTACK	302
875	SIG_SETMASK	302	SS_REPL_MAX	247
876	SIG_UNBLOCK	302	stack_t	303
877	single-quote	80	standard error	81
878	SIO	10	standard input	81
879	SIZE_MAX	317	standard output	81
880	size_t	328, 366	standard utilities	82
881	SI_ASYNCIO	304	stat data structure	356
882	SI_MESGQ	304	stderr	320
883	SI_QUEUE	304	STDERR_FILENO	411
884	SI_TIMER	304	stdin	320
885	SI_USER	304	STDIN_FILENO	411
886	slash	80	stdout	321
887	SNDZERO	334	STDOUT_FILENO	411
888	socket	80	strbuf	331
889	socket address	80	STREAM	82
890	SOCK_DGRAM	352	stream	82
891	SOCK_RAW	352	STREAM	216
892	soft limit	80	STREAM end	82
893	SOL_SOCKET	352	STREAM head	82
894	SOMAXCONN	353	STREAMS	25, 331
895	source code	80	STREAMS multiplexor	82
896	SO_ACCEPTCONN	352	STREAM_MAX	248
897	SO_BROADCAST	353	strfdinsert	331
898	SO_DEBUG	353	string	82
899	SO_DONTROUTE	353	strioctl	331
900	SO_ERROR	353	strpeek	331
901	SO_KEEPALIVE	353	strecvfd	331
902	SO_LINGER	353	str_list	331
903	SO_OOINLINE	353	str_mlist	332
904	SO_RCVBUF	353	ST_NOSUID	360
905	SO_RCVLOWAT	353	ST_RDONLY	360
906	SO_RCVTIMEO	353	subshell	83
907	SO_REUSEADDR	353	successfully transferred	83
908	SO_SNDBUF	353	supplementary group ID	83
909	SO_SNDLOWAT	353	suseconds_t	366
910	SO_SNDTIMEO	353	suspended job	83
911	SO_TYPE	353	symbolic link	83
912	space character	81	SYMLINK_MAX	249, 257
913	spawn	81	SYMLOOP_MAX	248
914	special built-in	81	SYMTYPE	376

## Index

915	synchronized I/O completion .....	83	S_IWUSR .....	357
916	synchronized I/O data integrity completion .....	84	S_IXGRP .....	357
917	synchronized I/O file integrity completion.....	84	S_IXOTH .....	357
918	synchronized I/O operation .....	84	S_IXUSR .....	357
919	synchronized input and output.....	83	S_MSG.....	333
920	synchronous I/O operation.....	84	S_OUTPUT .....	333
921	synchronously-generated signal .....	84	S_RDBAND .....	333
922	system .....	84	S_RDNORM .....	333
923	system console .....	85	S_TYPEISMQ.....	358
924	system crash .....	85	S_TYPEISSEM .....	358
925	system databases .....	85	S_TYPEISSHM .....	358
926	system documentation .....	85	S_TYPEISTMO .....	358
927	system process .....	85	S_WRBAND .....	333
928	system reboot .....	86	S_WRNORM .....	333
929	system trace event .....	86	tab character .....	86
930	system-wide.....	86	TABDLY .....	379
931	S_ constants		TABn.....	379
932	defined in <sys/stat.h> .....	<b>356-357</b>	TCIFLUSH .....	382
933	S_ macros		TCIOFF .....	382
934	defined in <sys/stat.h> .....	<b>357</b>	TCIOFLUSH .....	382
935	S_BANDURG .....	333	TCION .....	382
936	S_ERROR.....	333	TCOFLUSH.....	382
937	S_HANGUP.....	333	TCOOFF .....	382
938	S_HIPRI .....	333	TCOON .....	382
939	S_IFBLK .....	356	TCP_NODELAY .....	282
940	S_IFCHR.....	357	TCSADRAIN .....	381
941	S_IFDIR.....	357	TCSAFLUSH .....	381
942	S_IFIFO .....	357	TCSANOW .....	381
943	S_IFLNK.....	357	TCT .....	<b>11</b>
944	S_IFMT.....	356	TEF.....	<b>11</b>
945	S_IFREG.....	357	TERM .....	<b>162</b>
946	S_IFSOCK.....	357	terminal	
947	S_INPUT.....	333	controlling.....	184
948	S_IRGRP .....	357	terminal (or terminal device) .....	86
949	S_IROTH .....	357	terminal types.....	181
950	S_IRUSR .....	357	termios .....	183
951	S_IRWXG .....	357	canonical mode input processing .....	185
952	S_IRWXO .....	357	control modes.....	192
953	S_IRWXU .....	357	controlling terminal .....	184
954	S_ISBLK .....	357	input modes.....	189
955	S_ISCHR.....	357	local modes.....	193
956	S_ISDIR.....	358	non-canonical mode input processing.....	186
957	S_ISFIFO.....	358	output modes .....	190
958	S_ISGID.....	357-358	process groups .....	183
959	S_ISLNK .....	358	special control characters .....	194
960	S_ISREG.....	358	text column .....	86
961	S_ISSOCK.....	358	text file .....	86
962	S_ISUID.....	357-358	TGEXEC.....	376
963	S_ISVTX.....	357	TGREAD.....	376
964	S_IWGRP.....	357	TGWRITE.....	376
965	S_IWOTH.....	357	THOUSEP .....	242

966	THR	11	TRAP_TRACE	304
967	thread	87	TRC	12
968	thread ID	87	TRI	12
969	thread list	87	TRL	12
970	thread-safe	87	TSA	12
971	thread-safety	101	TSF	13
972	thread-specific data key	87	TSGID	376
973	tilde	87	TSH	13
974	timeb	364	TSP	13
975	timeouts	88	TSS	13
976	timer	88	TSUID	376
977	timer overrun	88	TSVTX	376
978	TIMER_ABSTIME	388	TTY_NAME_MAX	248
979	TIMER_MAX	248	TUEXEC	376
980	timer_t	366	TUREAD	376
981	timeval	345, 362	TUWRITE	376
982	time_t	366	TVERSION	376
983	TMAGIC	376	TVERSLEN	376
984	TMAGLEN	376	TYM	13
985	TMO	11	typed memory name space	90
986	TMPDIR	162	typed memory object	90
987	TMP_MAX	320	typed memory pool	90
988	TMR	12	typed memory port	90
989	TOEXEC	376	TZ	162
990	token	88	TZNAME_MAX	248
991	TOREAD	376	T_FMT	242
992	TOSTOP	381	T_FMT_AMPM	242
993	TOWRITE	376	t_uscalar_t	331
994	TPI	12	UCHAR_MAX	255
995	TPP	12	ucontext_t	396
996	TPS	12	uid_t	366
997	trace analyzer process	88	UINTMAX_MAX	317
998	trace controller process	88	UINTN_MAX	316
999	trace event	88	UINTPTR_MAX	317
1000	trace event type	88	UINT_FASTN_MAX	316
1001	trace event type mapping	88	UINT_LEASTN_MAX	316
1002	trace filter	89	UINT_MAX	255
1003	trace generation version	89	ULLONG_MAX	256
1004	trace log	89	ULONG_MAX	256
1005	trace point	89	UL_GETFSIZE	397
1006	trace stream	89	UL_SETFSIZE	397
1007	trace stream identifier	89	unbind	90
1008	trace system	89	undefined	6
1009	traced process	89	unspecified	6
1010	TRACE_EVENT_NAME_MAX	248	UP	13
1011	TRACE_NAME_MAX	248	upper multiplexing	82
1012	TRACE_SYS_MAX	248	upshifting	90
1013	TRACE_USER_EVENT_MAX	248	user database	90
1014	tracing	25, 101	user ID	91
1015	tracing status of a trace stream	89	user name	91
1016	TRAP_BRKPT	304	user trace event	91

## Index

1017	USER_PROCESS.....	419	WRDE_NOCMD .....	427
1018	USHRT_MAX.....	256	WRDE_NOSPACE .....	427
1019	utility.....	91, 105	WRDE_NOSYS .....	427
1020	Utility Syntax Guidelines.....	199	WRDE_REUSE.....	427
1021	utmpx.....	419	WRDE_SHOWERR.....	427
1022	variable .....	91	WRDE_SYNTAX.....	427
1023	variable assignment.....	105	WRDE_UNDEF.....	427
1024	VEOF.....	378	write.....	93
1025	VEOL.....	378	WSTOPPED .....	372
1026	VERASE.....	378	WSTOPSIG .....	324, 372
1027	vertical-tab character .....	91	WTERMSIG .....	324, 372
1028	VFS.....	360	WUNTRACED.....	324, 372
1029	VINTR.....	378	W_OK.....	404
1030	VKILL.....	378	XSI.....	13, 93
1031	VQUIT.....	378	conformance .....	15
1032	VSTART .....	378	XSI conformance.....	19
1033	VSTOP.....	378	XSI options groups.....	21
1034	VSUSP .....	378	XSI STREAMS .....	25
1035	VTDLY .....	380	XSI system interfaces	
1036	VTn .....	380	conformance .....	19
1037	warning		XSI-conformant.....	93
1038	OB .....	9	XSR .....	14
1039	OF.....	9	X_OK.....	404
1040	WCHAR_MAX .....	317, 423	YESEXPR.....	242
1041	WCHAR_MIN .....	317, 423	YESSTR.....	242
1042	WCONTINUED.....	372	zombie process.....	93
1043	WEOF .....	421, 423, 425		
1044	WEXITED.....	372		
1045	WEXITSTATUS.....	324, 372		
1046	white space .....	92		
1047	wide characters .....	115		
1048	wide-character code (C language) .....	92		
1049	wide-character input/output functions.....	92		
1050	wide-character string.....	92		
1051	WIFCONTINUED .....	372		
1052	WIFEXITED .....	324, 372		
1053	WIFSIGNALED .....	324, 372		
1054	WIFSTOPPED .....	324, 372		
1055	WINT_MAX .....	317		
1056	WINT_MIN.....	317		
1057	WNOHANG .....	324, 372		
1058	WNOWAIT .....	372		
1059	word .....	92		
1060	WORD_BIT .....	255-256		
1061	working directory.....	92		
1062	worldwide portability interface .....	92		
1063	WRDE_APPEND.....	427		
1064	WRDE_BADCHAR.....	427		
1065	WRDE_BADVAL.....	427		
1066	WRDE_CMDSUB .....	427		
1067	WRDE_DOOFFS.....	427		



## 1.1 Scope

The scope of IEEE Std 1003.1-200x is described in the Base Definitions volume of IEEE Std 1003.1-200x.

## 1.2 Conformance

Conformance requirements for IEEE Std 1003.1-200x are defined in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 2, Conformance.

## 1.3 Normative References

Normative references for IEEE Std 1003.1-200x are defined in the Base Definitions volume of IEEE Std 1003.1-200x.

## 1.4 Change History

Change history is described in the Rationale (Informative) volume of IEEE Std 1003.1-200x, and in the CHANGE HISTORY section of reference pages.

## 1.5 Terminology

This section appears in the Base Definitions volume of IEEE Std 1003.1-200x, but is repeated here for convenience:

For the purposes of IEEE Std 1003.1-200x, the following terminology definitions apply:

### **can**

Describes a permissible optional feature or behavior available to the user or application. The feature or behavior is mandatory for an implementation that conforms to IEEE Std 1003.1-200x. An application can rely on the existence of the feature or behavior.

### **implementation-defined**

Describes a value or behavior that is not defined by IEEE Std 1003.1-200x but is selected by an implementor. The value or behavior may vary among implementations that conform to IEEE Std 1003.1-200x. An application should not rely on the existence of the value or behavior. An application that relies on such a value or behavior cannot be assured to be portable across conforming implementations.

The implementor shall document such a value or behavior so that it can be used correctly by an application.

### **legacy**

Describes a feature or behavior that is being retained for compatibility with older applications, but which has limitations which make it inappropriate for developing portable

33 applications. New applications should use alternative means of obtaining equivalent  
34 functionality.

35 **may**

36 Describes a feature or behavior that is optional for an implementation that conforms to  
37 IEEE Std 1003.1-200x. An application should not rely on the existence of the feature or  
38 behavior. An application that relies on such a feature or behavior cannot be assured to be  
39 portable across conforming implementations.

40 To avoid ambiguity, the opposite of *may* is expressed as *need not*, instead of *may not*.

41 **shall**

42 For an implementation that conforms to IEEE Std 1003.1-200x, describes a feature or  
43 behavior that is mandatory. An application can rely on the existence of the feature or  
44 behavior.

45 For an application or user, describes a behavior that is mandatory.

46 **should**

47 For an implementation that conforms to IEEE Std 1003.1-200x, describes a feature or  
48 behavior that is recommended but not mandatory. An application should not rely on the  
49 existence of the feature or behavior. An application that relies on such a feature or behavior  
50 cannot be assured to be portable across conforming implementations.

51 For an application, describes a feature or behavior that is recommended programming  
52 practice for optimum portability.

53 **undefined**

54 Describes the nature of a value or behavior not defined by IEEE Std 1003.1-200x which  
55 results from use of an invalid program construct or invalid data input.

56 The value or behavior may vary among implementations that conform to  
57 IEEE Std 1003.1-200x. An application should not rely on the existence or validity of the  
58 value or behavior. An application that relies on any particular value or behavior cannot be  
59 assured to be portable across conforming implementations.

60 **unspecified**

61 Describes the nature of a value or behavior not specified by IEEE Std 1003.1-200x which  
62 results from use of a valid program construct or valid data input.

63 The value or behavior may vary among implementations that conform to  
64 IEEE Std 1003.1-200x. An application should not rely on the existence or validity of the  
65 value or behavior. An application that relies on any particular value or behavior cannot be  
66 assured to be portable across conforming implementations.

## 67 1.6 Definitions

68 Concepts and definitions are defined in the Base Definitions volume of IEEE Std 1003.1-200x.

## 69 1.7 Relationship to Other Formal Standards

70 Great care has been taken to ensure that this volume of IEEE Std 1003.1-200x is fully aligned  
71 with the following standards:

72 ISO C (1999)

73 ISO/IEC 9899: 1999, Programming Languages — C.

74 Parts of the ISO/IEC 9899: 1999 standard (hereinafter referred to as the ISO C standard) are  
75 referenced to describe requirements also mandated by this volume of IEEE Std 1003.1-200x.  
76 Some functions and headers included within this volume of IEEE Std 1003.1-200x have a version  
77 in the ISO C standard; in this case CX markings are added as appropriate to show where the  
78 ISO C standard has been extended (see Section 1.8.1). Any conflict between this volume of  
79 IEEE Std 1003.1-200x and the ISO C standard is unintentional.

80 This volume of IEEE Std 1003.1-200x also allows, but does not require, mathematics functions to  
81 support IEEE Std 754-1985 and IEEE Std 854-1987.

## 82 1.8 Portability

83 Some of the utilities in the Shell and Utilities volume of IEEE Std 1003.1-200x and functions in  
84 the System Interfaces volume of IEEE Std 1003.1-200x describe functionality that might not be  
85 fully portable to systems meeting the requirements for POSIX conformance (see the Base  
86 Definitions volume of IEEE Std 1003.1-200x, Chapter 2, Conformance).

87 Where optional, enhanced, or reduced functionality is specified, the text is shaded and a code in  
88 the margin identifies the nature of the option, extension, or warning (see Section 1.8.1). For  
89 maximum portability, an application should avoid such functionality.

### 90 1.8.1 Codes

91 Margin codes and their meanings are listed in the Base Definitions volume of  
92 IEEE Std 1003.1-200x, but are repeated here for convenience:

93 ADV **Advisory Information**

94 The functionality described is optional. The functionality described is also an extension to the  
95 ISO C standard.

96 Where applicable, functions are marked with the ADV margin legend in the SYNOPSIS section.  
97 Where additional semantics apply to a function, the material is identified by use of the ADV  
98 margin legend.

99 AIO **Asynchronous Input and Output**

100 The functionality described is optional. The functionality described is also an extension to the  
101 ISO C standard.

102 Where applicable, functions are marked with the AIO margin legend in the SYNOPSIS section.  
103 Where additional semantics apply to a function, the material is identified by use of the AIO  
104 margin legend.

105 BAR **Barriers**

106 The functionality described is optional. The functionality described is also an extension to the

107 ISO C standard.

108 Where applicable, functions are marked with the BAR margin legend in the SYNOPSIS section.  
109 Where additional semantics apply to a function, the material is identified by use of the BAR  
110 margin legend.

111 BE **Batch Environment Services and Utilities**  
112 The functionality described is optional.

113 Where applicable, utilities are marked with the BE margin legend in the SYNOPSIS section.  
114 Where additional semantics apply to a utility, the material is identified by use of the BE margin  
115 legend.

116 CD **C-Language Development Utilities**  
117 The functionality described is optional.

118 Where applicable, utilities are marked with the CD margin legend in the SYNOPSIS section.  
119 Where additional semantics apply to a utility, the material is identified by use of the CD margin  
120 legend.

121 CPT **Process CPU-Time Clocks**  
122 The functionality described is optional. The functionality described is also an extension to the  
123 ISO C standard.

124 Where applicable, functions are marked with the CPT margin legend in the SYNOPSIS section.  
125 Where additional semantics apply to a function, the material is identified by use of the CPT  
126 margin legend.

127 CS **Clock Selection**  
128 The functionality described is optional. The functionality described is also an extension to the  
129 ISO C standard.

130 Where applicable, functions are marked with the CS margin legend in the SYNOPSIS section.  
131 Where additional semantics apply to a function, the material is identified by use of the CS  
132 margin legend.

133 CX **Extension to the ISO C standard**  
134 The functionality described is an extension to the ISO C standard. Application writers may make  
135 use of an extension as it is supported on all IEEE Std 1003.1-200x-conforming systems.

136 With each function or header from the ISO C standard, a statement to the effect that “any  
137 conflict is unintentional” is included. That is intended to refer to a direct conflict.  
138 IEEE Std 1003.1-200x acts in part as a profile of the ISO C standard, and it may choose to further  
139 constrain behaviors allowed to vary by the ISO C standard. Such limitations are not considered  
140 conflicts.

141 FD **FORTRAN Development Utilities**  
142 The functionality described is optional.

143 Where applicable, utilities are marked with the FD margin legend in the SYNOPSIS section.  
144 Where additional semantics apply to a utility, the material is identified by use of the FD margin  
145 legend.

146 FR **FORTRAN Runtime Utilities**  
147 The functionality described is optional.

148 Where applicable, utilities are marked with the FR margin legend in the SYNOPSIS section.  
149 Where additional semantics apply to a utility, the material is identified by use of the FR margin  
150 legend.

- 151 FSC **File Synchronization**  
152 The functionality described is optional. The functionality described is also an extension to the  
153 ISO C standard.
- 154 Where applicable, functions are marked with the FSC margin legend in the SYNOPSIS section.  
155 Where additional semantics apply to a function, the material is identified by use of the FSC  
156 margin legend.
- 157 IP6 **IPV6**  
158 The functionality described is optional. The functionality described is also an extension to the  
159 ISO C standard.
- 160 Where applicable, functions are marked with the IP6 margin legend in the SYNOPSIS section.  
161 Where additional semantics apply to a function, the material is identified by use of the IP6  
162 margin legend.
- 163 MC1 **Advisory Information and either Memory Mapped Files or Shared Memory Objects**  
164 The functionality described is optional. The functionality described is also an extension to the  
165 ISO C standard.
- 166 This is a shorthand notation for combinations of multiple option codes.
- 167 Where applicable, functions are marked with the MC1 margin legend in the SYNOPSIS section.  
168 Where additional semantics apply to a function, the material is identified by use of the MC1  
169 margin legend.
- 170 Refer to the Base Definitions volume of IEEE Std 1003.1-200x, Section 1.5.2, Margin Code  
171 Notation.
- 172 MC2 **Memory Mapped Files, Shared Memory Objects, or Memory Protection**  
173 The functionality described is optional. The functionality described is also an extension to the  
174 ISO C standard.
- 175 This is a shorthand notation for combinations of multiple option codes.
- 176 Where applicable, functions are marked with the MC2 margin legend in the SYNOPSIS section.  
177 Where additional semantics apply to a function, the material is identified by use of the MC2  
178 margin legend.
- 179 Refer to the Base Definitions volume of IEEE Std 1003.1-200x, Section 1.5.2, Margin Code  
180 Notation.
- 181 MF **Memory Mapped Files**  
182 The functionality described is optional. The functionality described is also an extension to the  
183 ISO C standard.
- 184 Where applicable, functions are marked with the MF margin legend in the SYNOPSIS section.  
185 Where additional semantics apply to a function, the material is identified by use of the MF  
186 margin legend.
- 187 ML **Process Memory Locking**  
188 The functionality described is optional. The functionality described is also an extension to the  
189 ISO C standard.
- 190 Where applicable, functions are marked with the ML margin legend in the SYNOPSIS section.  
191 Where additional semantics apply to a function, the material is identified by use of the ML  
192 margin legend.
- 193 MLR **Range Memory Locking**  
194 The functionality described is optional. The functionality described is also an extension to the

- 195 ISO C standard.
- 196 Where applicable, functions are marked with the MLR margin legend in the SYNOPSIS section.  
197 Where additional semantics apply to a function, the material is identified by use of the MLR  
198 margin legend.
- 199 MON **Monotonic Clock**  
200 The functionality described is optional. The functionality described is also an extension to the  
201 ISO C standard.
- 202 Where applicable, functions are marked with the MON margin legend in the SYNOPSIS section.  
203 Where additional semantics apply to a function, the material is identified by use of the MON  
204 margin legend.
- 205 MPR **Memory Protection**  
206 The functionality described is optional. The functionality described is also an extension to the  
207 ISO C standard.
- 208 Where applicable, functions are marked with the MPR margin legend in the SYNOPSIS section.  
209 Where additional semantics apply to a function, the material is identified by use of the MPR  
210 margin legend.
- 211 MSG **Message Passing**  
212 The functionality described is optional. The functionality described is also an extension to the  
213 ISO C standard.
- 214 Where applicable, functions are marked with the MSG margin legend in the SYNOPSIS section.  
215 Where additional semantics apply to a function, the material is identified by use of the MSG  
216 margin legend.
- 217 MX **IEC 60559 Floating-Point Option**  
218 The functionality described is optional. The functionality described is also an extension to the  
219 ISO C standard.
- 220 Where applicable, functions are marked with the MX margin legend in the SYNOPSIS section.  
221 Where additional semantics apply to a function, the material is identified by use of the MX  
222 margin legend.
- 223 OB **Obsolescent**  
224 The functionality described may be withdrawn in a future version of this volume of  
225 IEEE Std 1003.1-200x. Strictly Conforming POSIX Applications and Strictly Conforming XSI  
226 Applications shall not use obsolescent features.
- 227 OF **Output Format Incompletely Specified**  
228 The functionality described is an XSI extension. The format of the output produced by the utility  
229 is not fully specified. It is therefore not possible to post-process this output in a consistent  
230 fashion. Typical problems include unknown length of strings and unspecified field delimiters.
- 231 OH **Optional Header**  
232 In the SYNOPSIS section of some interfaces in the System Interfaces volume of  
233 IEEE Std 1003.1-200x an included header is marked as in the following example:
- 234 OH `#include <sys/types.h>`  
235 `#include <grp.h>`  
236 `struct group *getgrnam(const char *name);`
- 237 This indicates that the marked header is not required on XSI-conformant systems.

238	PIO	<b>Prioritized Input and Output</b>
239		The functionality described is optional. The functionality described is also an extension to the
240		ISO C standard.
241		Where applicable, functions are marked with the PIO margin legend in the SYNOPSIS section.
242		Where additional semantics apply to a function, the material is identified by use of the PIO
243		margin legend.
244	PS	<b>Process Scheduling</b>
245		The functionality described is optional. The functionality described is also an extension to the
246		ISO C standard.
247		Where applicable, functions are marked with the PS margin legend in the SYNOPSIS section.
248		Where additional semantics apply to a function, the material is identified by use of the PS
249		margin legend.
250	RS	<b>Raw Sockets</b>
251		The functionality described is optional. The functionality described is also an extension to the
252		ISO C standard.
253		Where applicable, functions are marked with the RS margin legend in the SYNOPSIS section.
254		Where additional semantics apply to a function, the material is identified by use of the RS
255		margin legend.
256	RTS	<b>Realtime Signals Extension</b>
257		The functionality described is optional. The functionality described is also an extension to the
258		ISO C standard.
259		Where applicable, functions are marked with the RTS margin legend in the SYNOPSIS section.
260		Where additional semantics apply to a function, the material is identified by use of the RTS
261		margin legend.
262	SD	<b>Software Development Utilities</b>
263		The functionality described is optional.
264		Where applicable, utilities are marked with the SD margin legend in the SYNOPSIS section.
265		Where additional semantics apply to a utility, the material is identified by use of the SD
266		margin legend.
267	SEM	<b>Semaphores</b>
268		The functionality described is optional. The functionality described is also an extension to the
269		ISO C standard.
270		Where applicable, functions are marked with the SEM margin legend in the SYNOPSIS section.
271		Where additional semantics apply to a function, the material is identified by use of the SEM
272		margin legend.
273	SHM	<b>Shared Memory Objects</b>
274		The functionality described is optional. The functionality described is also an extension to the
275		ISO C standard.
276		Where applicable, functions are marked with the SHM margin legend in the SYNOPSIS section.
277		Where additional semantics apply to a function, the material is identified by use of the SHM
278		margin legend.
279	SIO	<b>Synchronized Input and Output</b>
280		The functionality described is optional. The functionality described is also an extension to the
281		ISO C standard.

282 Where applicable, functions are marked with the SIO margin legend in the SYNOPSIS section.  
283 Where additional semantics apply to a function, the material is identified by use of the SIO  
284 margin legend.

285 SPI **Spin Locks**

286 The functionality described is optional. The functionality described is also an extension to the  
287 ISO C standard.

288 Where applicable, functions are marked with the SPI margin legend in the SYNOPSIS section.  
289 Where additional semantics apply to a function, the material is identified by use of the SPI  
290 margin legend.

291 SPN **Spawn**

292 The functionality described is optional. The functionality described is also an extension to the  
293 ISO C standard.

294 Where applicable, functions are marked with the SPN margin legend in the SYNOPSIS section.  
295 Where additional semantics apply to a function, the material is identified by use of the SPN  
296 margin legend.

297 SS **Process Sporadic Server**

298 The functionality described is optional. The functionality described is also an extension to the  
299 ISO C standard.

300 Where applicable, functions are marked with the SS margin legend in the SYNOPSIS section.  
301 Where additional semantics apply to a function, the material is identified by use of the SS  
302 margin legend.

303 TCT **Thread CPU-Time Clocks**

304 The functionality described is optional. The functionality described is also an extension to the  
305 ISO C standard.

306 Where applicable, functions are marked with the TCT margin legend in the SYNOPSIS section.  
307 Where additional semantics apply to a function, the material is identified by use of the TCT  
308 margin legend.

309 TEF **Trace Event Filter**

310 The functionality described is optional. The functionality described is also an extension to the  
311 ISO C standard.

312 Where applicable, functions are marked with the TEF margin legend in the SYNOPSIS section.  
313 Where additional semantics apply to a function, the material is identified by use of the TEF  
314 margin legend.

315 THR **Threads**

316 The functionality described is optional. The functionality described is also an extension to the  
317 ISO C standard.

318 Where applicable, functions are marked with the THR margin legend in the SYNOPSIS section.  
319 Where additional semantics apply to a function, the material is identified by use of the THR  
320 margin legend.

321 TMO **Timeouts**

322 The functionality described is optional. The functionality described is also an extension to the  
323 ISO C standard.

324 Where applicable, functions are marked with the TMO margin legend in the SYNOPSIS section.  
325 Where additional semantics apply to a function, the material is identified by use of the TMO  
326 margin legend.

327	TMR	<b>Timers</b>
328		The functionality described is optional. The functionality described is also an extension to the
329		ISO C standard.
330		Where applicable, functions are marked with the TMR margin legend in the SYNOPSIS section.
331		Where additional semantics apply to a function, the material is identified by use of the TMR
332		margin legend.
333	TPI	<b>Thread Priority Inheritance</b>
334		The functionality described is optional. The functionality described is also an extension to the
335		ISO C standard.
336		Where applicable, functions are marked with the TPI margin legend in the SYNOPSIS section.
337		Where additional semantics apply to a function, the material is identified by use of the TPI
338		margin legend.
339	TPP	<b>Thread Priority Protection</b>
340		The functionality described is optional. The functionality described is also an extension to the
341		ISO C standard.
342		Where applicable, functions are marked with the TPP margin legend in the SYNOPSIS section.
343		Where additional semantics apply to a function, the material is identified by use of the TPP
344		margin legend.
345	TPS	<b>Thread Execution Scheduling</b>
346		The functionality described is optional. The functionality described is also an extension to the
347		ISO C standard.
348		Where applicable, functions are marked with the TPS margin legend for the SYNOPSIS section.
349		Where additional semantics apply to a function, the material is identified by use of the TPS
350		margin legend.
351	TRC	<b>Trace</b>
352		The functionality described is optional. The functionality described is also an extension to the
353		ISO C standard.
354		Where applicable, functions are marked with the TRC margin legend in the SYNOPSIS section.
355		Where additional semantics apply to a function, the material is identified by use of the TRC
356		margin legend.
357	TRI	<b>Trace Inherit</b>
358		The functionality described is optional. The functionality described is also an extension to the
359		ISO C standard.
360		Where applicable, functions are marked with the TRI margin legend in the SYNOPSIS section.
361		Where additional semantics apply to a function, the material is identified by use of the TRI
362		margin legend.
363	TRL	<b>Trace Log</b>
364		The functionality described is optional. The functionality described is also an extension to the
365		ISO C standard.
366		Where applicable, functions are marked with the TRL margin legend in the SYNOPSIS section.
367		Where additional semantics apply to a function, the material is identified by use of the TRL
368		margin legend.
369	TSA	<b>Thread Stack Address Attribute</b>
370		The functionality described is optional. The functionality described is also an extension to the
371		ISO C standard.

372 Where applicable, functions are marked with the TSA margin legend for the SYNOPSIS section.  
373 Where additional semantics apply to a function, the material is identified by use of the TSA  
374 margin legend.

375 TSF **Thread-Safe Functions**

376 The functionality described is optional. The functionality described is also an extension to the  
377 ISO C standard.

378 Where applicable, functions are marked with the TSF margin legend in the SYNOPSIS section.  
379 Where additional semantics apply to a function, the material is identified by use of the TSF  
380 margin legend.

381 TSH **Thread Process-Shared Synchronization**

382 The functionality described is optional. The functionality described is also an extension to the  
383 ISO C standard.

384 Where applicable, functions are marked with the TSH margin legend in the SYNOPSIS section.  
385 Where additional semantics apply to a function, the material is identified by use of the TSH  
386 margin legend.

387 TSP **Thread Sporadic Server**

388 The functionality described is optional. The functionality described is also an extension to the  
389 ISO C standard.

390 Where applicable, functions are marked with the TSP margin legend in the SYNOPSIS section.  
391 Where additional semantics apply to a function, the material is identified by use of the TSP  
392 margin legend.

393 TSS **Thread Stack Address Size**

394 The functionality described is optional. The functionality described is also an extension to the  
395 ISO C standard.

396 Where applicable, functions are marked with the TSS margin legend in the SYNOPSIS section.  
397 Where additional semantics apply to a function, the material is identified by use of the TSS  
398 margin legend.

399 TYM **Typed Memory Objects**

400 The functionality described is optional. The functionality described is also an extension to the  
401 ISO C standard.

402 Where applicable, functions are marked with the TYM margin legend in the SYNOPSIS section.  
403 Where additional semantics apply to a function, the material is identified by use of the TYM  
404 margin legend.

405 UP **User Portability Utilities**

406 The functionality described is optional.

407 Where applicable, utilities are marked with the UP margin legend in the SYNOPSIS section.  
408 Where additional semantics apply to a utility, the material is identified by use of the UP margin  
409 legend.

410 XSI **Extension**

411 The functionality described is an XSI extension. Functionality marked XSI is also an extension to  
412 the ISO C standard. Application writers may confidently make use of an extension on all  
413 systems supporting the X/Open System Interfaces Extension.

414 If an entire SYNOPSIS section is shaded and marked XSI, all the functionality described in that  
415 reference page is an extension. See the Base Definitions volume of IEEE Std 1003.1-200x, Section  
416 3.439, XSI.

417 XSR **XSI STREAMS**  
 418 The functionality described is optional. The functionality described is also an extension to the  
 419 ISO C standard.

420 Where applicable, functions are marked with the XSR margin legend in the SYNOPSIS section.  
 421 Where additional semantics apply to a function, the material is identified by use of the XSR  
 422 margin legend.

## 423 1.9 Format of Entries

424 The entries in Chapter 3 are based on a common format as follows. The only sections relating to  
 425 conformance are the SYNOPSIS, DESCRIPTION, RETURN VALUE, and ERRORS sections.

### 426 NAME

427 This section gives the name or names of the entry and briefly states its purpose.

### 428 SYNOPSIS

429 This section summarizes the use of the entry being described. If it is necessary to  
 430 include a header to use this function, the names of such headers are shown, for  
 431 example:

```
432 #include <stdio.h>
```

### 433 DESCRIPTION

434 This section describes the functionality of the function or header.

### 435 RETURN VALUE

436 This section indicates the possible return values, if any.

437 If the implementation can detect errors, “successful completion” means that no error  
 438 has been detected during execution of the function. If the implementation does detect  
 439 an error, the error is indicated.

440 For functions where no errors are defined, “successful completion” means that if the  
 441 implementation checks for errors, no error has been detected. If the implementation can  
 442 detect errors, and an error is detected, the indicated return value is returned and *errno*  
 443 may be set.

### 444 ERRORS

445 This section gives the symbolic names of the error values returned by a function or  
 446 stored into a variable accessed through the symbol *errno* if an error occurs.

447 “No errors are defined” means that error values returned by a function or stored into a  
 448 variable accessed through the symbol *errno*, if any, depend on the implementation.

### 449 EXAMPLES

450 This section is non-normative.

451 This section gives examples of usage, where appropriate. In the event of conflict  
 452 between an example and a normative part of this volume of IEEE Std 1003.1-200x, the  
 453 normative material is to be taken as correct.

### 454 APPLICATION USAGE

455 This section is non-normative.

456 This section gives warnings and advice to application writers about the entry. In the  
 457 event of conflict between warnings and advice and a normative part of this volume of  
 458 IEEE Std 1003.1-200x, the normative material is to be taken as correct.

459 **RATIONALE**

460 This section is non-normative.

461 This section contains historical information concerning the contents of this volume of  
462 IEEE Std 1003.1-200x and why features were included or discarded by the standard  
463 developers.

464 **FUTURE DIRECTIONS**

465 This section is non-normative.

466 This section provides comments which should be used as a guide to current thinking;  
467 there is not necessarily a commitment to adopt these future directions.

468 **SEE ALSO**

469 This section is non-normative.

470 This section gives references to related information.

471 **CHANGE HISTORY**

472 This section is non-normative.

473 This section shows the derivation of the entry and any significant changes that have  
474 been made to it. |

# General Information

475

476 This chapter covers information that is relevant to all the functions specified in Chapter 3 and  
 477 the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 13, Headers.

## 478 2.1 Use and Implementation of Functions

479 Each of the following statements shall apply unless explicitly stated otherwise in the detailed  
 480 descriptions that follow:

- 481 1. If an argument to a function has an invalid value (such as a value outside the domain of  
 482 the function, or a pointer outside the address space of the program, or a null pointer), the  
 483 behavior is undefined.
- 484 2. Any function declared in a header may also be implemented as a macro defined in the  
 485 header, so a function should not be declared explicitly if its header is included. Any macro  
 486 definition of a function can be suppressed locally by enclosing the name of the function in  
 487 parentheses, because the name is then not followed by the left parenthesis that indicates  
 488 expansion of a macro function name. For the same syntactic reason, it is permitted to take  
 489 the address of a function even if it is also defined as a macro. The use of the C-language  
 490 `#undef` construct to remove any such macro definition shall also ensure that an actual  
 491 function is referred to.
- 492 3. Any invocation of a function that is implemented as a macro shall expand to code that  
 493 evaluates each of its arguments exactly once, fully protected by parentheses where  
 494 necessary, so it is generally safe to use arbitrary expressions as arguments. Likewise, those  
 495 function-like macros described in the following sections may be invoked in an expression  
 496 anywhere a function with a compatible return type could be called.
- 497 4. Provided that a function can be declared without reference to any type defined in a header,  
 498 it is also permissible to declare the function, either explicitly or implicitly, and use it  
 499 without including its associated header.
- 500 5. If a function that accepts a variable number of arguments is not declared (explicitly or by  
 501 including its associated header), the behavior is undefined.

## 502 2.2 The Compilation Environment

### 503 2.2.1 POSIX.1 Symbols

504 Certain symbols in this volume of IEEE Std 1003.1-200x are defined in headers (see the Base  
 505 Definitions volume of IEEE Std 1003.1-200x, Chapter 13, Headers). Some of those headers could  
 506 also define symbols other than those defined by IEEE Std 1003.1-200x, potentially conflicting |  
 507 with symbols used by the application. Also, IEEE Std 1003.1-200x defines symbols that are not |  
 508 permitted by other standards to appear in those headers without some control on the visibility |  
 509 of those symbols.

510 Symbols called “feature test macros” are used to control the visibility of symbols that might be  
 511 included in a header. Implementations, future versions of IEEE Std 1003.1-200x, and other |  
 512 standards may define additional feature test macros. |

513 In the compilation of an application that **#defines** a feature test macro specified by  
 514 IEEE Std 1003.1-200x, no header defined by IEEE Std 1003.1-200x shall be included prior to the  
 515 definition of the feature test macro. This restriction also applies to any implementation-  
 516 provided header in which these feature test macros are used. If the definition of the macro does  
 517 not precede the **#include**, the result is undefined.

518 Feature test macros shall begin with the underscore character ('\_').

#### 519 2.2.1.1 *The `_POSIX_C_SOURCE` Feature Test Macro*

520 A POSIX-conforming application should ensure that the feature test macro `_POSIX_C_SOURCE`  
 521 is defined before inclusion of any header.

522 When an application includes a header described by IEEE Std 1003.1-200x, and when this feature  
 523 test macro is defined to have the value `200xxxL`:

- 524 1. All symbols required by IEEE Std 1003.1-200x to appear when the header is included shall  
 525 be made visible.
- 526 2. Symbols that are explicitly permitted, but not required, by IEEE Std 1003.1-200x to appear  
 527 in that header (including those in reserved name spaces) may be made visible.
- 528 3. Additional symbols not required or explicitly permitted by IEEE Std 1003.1-200x to be in  
 529 that header shall not be made visible, except when enabled by another feature test macro.

530 Identifiers in IEEE Std 1003.1-200x may only be undefined using the **#undef** directive as  
 531 described in Section 2.1 (on page 463) or Section 2.2.2. These **#undef** directives shall follow all  
 532 **#include** directives of any header in IEEE Std 1003.1-200x.

533 **Note:** The POSIX.1-1990 standard specified a macro called `_POSIX_SOURCE`. This has been  
 534 superseded by `_POSIX_C_SOURCE`.

#### 535 2.2.1.2 *The `_XOPEN_SOURCE` Feature Test Macro*

536 XSI An XSI-conforming application should ensure that the feature test macro `_XOPEN_SOURCE` is  
 537 defined with the value `600` before inclusion of any header. This is needed to enable the  
 538 functionality described in Section 2.2.1.1 and in addition to enable the X/Open System Interfaces  
 539 Extension.

540 Since this volume of IEEE Std 1003.1-200x is aligned with the ISO C standard, and since all  
 541 functionality enabled by `_POSIX_C_SOURCE` set equal to `200xxxL` is enabled by  
 542 `_XOPEN_SOURCE` set equal to `600`, there should be no need to define `_POSIX_C_SOURCE` if  
 543 `_XOPEN_SOURCE` is so defined. Therefore, if `_XOPEN_SOURCE` is set equal to `600` and  
 544 `_POSIX_C_SOURCE` is set equal to `200xxxL`, the behavior is the same as if only  
 545 `_XOPEN_SOURCE` is defined and set equal to `600`. However, should `_POSIX_C_SOURCE` be set  
 546 to a value greater than `200xxxL`, the behavior is unspecified.

### 547 2.2.2 The Name Space

548 All identifiers in this volume of IEEE Std 1003.1-200x, except *environ*, are defined in at least one  
 549 of the headers, as shown in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 13,  
 550 XSI Headers. When `_XOPEN_SOURCE` or `_POSIX_C_SOURCE` is defined, each header defines or  
 551 declares some identifiers, potentially conflicting with identifiers used by the application. The set  
 552 of identifiers visible to the application consists of precisely those identifiers from the header  
 553 pages of the included headers, as well as additional identifiers reserved for the implementation.  
 554 In addition, some headers may make visible identifiers from other headers as indicated on the  
 555 relevant header pages.

556 Implementations may also add members to a structure or union without controlling the  
557 visibility of those members with a feature test macro, as long as a user-defined macro with the  
558 same name cannot interfere with the correct interpretation of the program. The identifiers  
559 reserved for use by the implementation are described below:

- 560 1. Each identifier with external linkage described in the header section is reserved for use as  
561 an identifier with external linkage if the header is included.
- 562 2. Each macro described in the header section is reserved for any use if the header is  
563 included.
- 564 3. Each identifier with file scope described in the header section is reserved for use as an  
565 identifier with file scope in the same name space if the header is included.

566 The prefixes `posix_`, `POSIX_`, and `_POSIX_` are reserved for use by IEEE Std 1003.1-200x and  
567 other POSIX standards. Implementations may add symbols to the headers shown in the  
568 following table, provided the identifiers for those symbols begin with the corresponding  
569 reserved prefixes in the following table, and do not use the reserved prefixes `posix_`, `POSIX_`, or  
570 `_POSIX_`.



619  
620  
621  
622  
623  
624  
625  
626  
627  
628  
629  
630  
631  
632  
633  
634  
635  
636  
637  
638

XSI  
  
  
XSI  
  
TMR  
TMR  
XSI  
XSI  
  
XSI

Header	Prefix	Suffix	Complete Name
<sys/uio.h>	iov_		UIO_MAXIOV
<sys/un.h>	sun_		
<sys/utsname.h>	uts_		
<sys/wait.h>	si_, W[A-Z], P_		
<termios.h>	c_		
<time.h>	tm_		
	clock_, timer_, it_, tv_, CLOCK_, TIMER_		
<ucontext.h>	uc_, ss_		
<ulimit.h>	UL_		
<utime.h>	utim_		
<utmpx.h>	ut_	_LVL, _TIME, _PROCESS	
<wchar.h>	wcs[a-z]		
<wctype.h>	is[a-z], to[a-z]		
<wordexp.h>	we_		
ANY header	POSIX_, _POSIX_, posix_	_t	

639  
640  
641  
642

**Note:** The notation [A–Z] indicates any uppercase letter in the portable character set. The notation [a–z] indicates any lowercase letter in the portable character set. Commas and spaces in the lists of prefixes and complete names in the above table are not part of any prefix or complete name.

643  
644  
645  
646

If any header in the following table is included, macros with the prefixes shown may be defined. After the last inclusion of a given header, an application may use identifiers with the corresponding prefixes for its own purpose, provided their use is preceded by a **#undef** of the corresponding macro.

647  
648  
649  
650  
651  
652  
653  
654  
655  
656  
657  
658  
659  
660  
661  
662  
663  
664  
665  
666  
667  
668  
669  
670  
671  
672  
673  
674  
675  
676  
677  
678  
679  
680  
681  
682  
683

Header	Prefix
<dlfcn.h>	RTLD_
<fcntl.h>	F_, O_, S_
<fmtmsg.h>	MM_
<fnmatch.h>	FNM_
<ftw.h>	FTW
<glob.h>	GLOB_
<inttypes.h>	PRI[a-z], SCN[a-z]
<ndbm.h>	DBM_
<net/if.h>	IF_
<netinet/in.h>	IMPLINK_, IN_, INADDR_, IP_, IPPORT_, IPPROTO_, SOCK_
<netinet/tcp.h>	IPV6_, IN6_
<nl_types.h>	TCP_
<poll.h>	NL_
<regex.h>	POLL
<signal.h>	REG_
stdint.h	SA_, SIG_[0-9a-z_], BUS_, CLD_, FPE_, ILL_, POLL_, SEGV_, SI_, SS_, SV_, TRAP_
<stropts.h>	INT[0-9A-Z_]_MIN, INT[0-9A-Z_]_MAX, INT[0-9A-Z_]_C UINT[0-9A-Z_]_MIN, UINT[0-9A-Z_]_MAX, UINT[0-9A-Z_]_C
<syslog.h>	FLUSH[A-Z], I_, M_, MUXID_R[A-Z], S_, SND[A-Z], STR
<sys/ipc.h>	LOG_
<sys/mman.h>	IPC_
<sys/msg.h>	PROT_, MAP_, MS_
<sys/resource.h>	MSG[A-Z]
<sys/sem.h>	PRIO_, RLIM_, RLIMIT_, RUSAGE_
<sys/shm.h>	SEM_
<sys/socket.h>	SHM[A-Z], SHM_[A-Z]
<sys/stat.h>	AF_, CMSG_, MSG_, PF_, SCM_, SHUT_, SO
<sys/statvfs.h>	S_
<sys/time.h>	ST_
<sys/uid.h>	FD_, ITIMER_
<sys/wait.h>	IOV_
<termios.h>	BUS_, CLD_, FPE_, ILL_, POLL_, SEGV_, SI_, TRAP_
<wordexp.h>	V, I, O, TC, B[0-9] (See below.)
	WRDE_

684  
685  
686

**Note:** The notation [0-9] indicates any digit. The notation [A-Z] indicates any uppercase letter in the portable character set. The notation [0-9a-z\_] indicates any digit, any lowercase letter in the portable character set, or underscore.

687

The following reserved names are used as exact matches for <termios.h>:

688  
689  
690  
691  
692  
693

CBAUD	EXTB	VDSUSP
DEFECHO	FLUSHO	VLNEXT
ECHOCTL	LOBLK	VREPRINT
ECHOKE	PENDIN	VSTATUS
ECHOPRT	SWTCH	VWERASE
EXTA	VDISCARD	

694 The following identifiers are reserved regardless of the inclusion of headers:

- 695 1. All identifiers that begin with an underscore and either an uppercase letter or another  
696 underscore are always reserved for any use by the implementation.
- 697 2. All identifiers that begin with an underscore are always reserved for use as identifiers with  
698 file scope in both the ordinary identifier and tag name spaces.
- 699 3. All identifiers in the table below are reserved for use as identifiers with external linkage.  
700 Some of these identifiers do not appear in this volume of IEEE Std 1003.1-200x, but are  
701 reserved for future use by the ISO C standard.

702	_Exit	cexp	fesetexceptflag	localtime	scalbn
703	abort	cexpf	fesetround	log	scalbnf
704	abs	cexpl	fetestexcept	log10	scalbnl
705	acos	cimag	feupdateenv	log10f	scanf
706	acosf	cimagf	fflush	log10l	setbuf
707	acosh	cimagl	fgetc	log1p	setjmp
708	acoshf	clearerr	fgetpos	log1pf	setlocale
709	acoshl	clock	fgets	log1pl	setvbuf
710	acosl	clog	fgetwc	log2	signal
711	acosl	clogf	fgetws	log2f	sin
712	asctime	clogl	floor	log2l	sinf
713	asin	conj	floorf	logb	sinh
714	asinf	conjf	floorl	logbf	sinhf
715	asinh	conjl	fma	logbl	sinhl
716	asinhf	copysign	fmaf	logf	sinl
717	asinhl	copysignf	fmal	logl	sprintf
718	asinl	copysignl	fmax	longjmp	sqrt
719	asinl	cos	fmaxf	lrint	sqrtf
720	atan	cosf	fmaxl	lrintf	sqrtl
721	atan2	cosh	fmin	lrintl	srand
722	atan2f	coshf	fminf	lround	sscanf
723	atan2l	coshl	fminl	lroundf	str[a-z]*
724	atanf	cosl	fmod	lroundl	strtof
725	atanf	cpow	fmodf	malloc	strtoimax
726	atanh	cpowf	fmodl	mblen	strtold
727	atanh	cpowl	fopen	mbrlen	strtoll
728	atanhf	cproj	fprintf	mbrtowc	strtoull
729	atanhl	cprojf	fputc	mbsinit	strtoumax
730	atanl	cprojl	fputs	mbsrtowcs	swprintf
731	atanl	creal	fputwc	mbstowcs	swscanf
732	atexit	crealf	fputws	mbtowc	system
733	atof	creall	fread	mem[a-z]*	tan
734	atoi	csin	free	mktime	tanf
735	atol	csinf	freopen	modf	tanh
736	atoll	csinh	frexp	modff	tanhf
737	bsearch	csinhf	frexpf	modfl	tanhl
738	cabs	csinhl	frexpl	nan	tanl
739	cabsf	csinl	fscanf	nanf	tgamma
740	cabsl	csqrt	fseek	nanl	tgammaf
741	cacos	csqrtf	fsetpos	nearbyint	tgammal

742	cacosf	csqrtl	ftell	nearbyintf	time
743	cacosh	ctan	fwide	nearbyintl	tmpfile
744	cacoshf	ctanf	fwprintf	nextafterf	tmpnam
745	cacoshl	ctanl	fwwrite	nextafterl	to[a-z]*
746	cacosl	ctime	fwscanf	nexttoward	trunc
747	calloc	difftime	getc	nexttowardf	truncf
748	carg	div	getchar	nexttowardl	truncl
749	cargf	erfcf	getenv	perror	ungetc
750	cargl	erfcl	gets	pow	ungetwc
751	casin	erff	getwc	powf	va_end
752	casinf	erfl	getwchar	powl	vfprintf
753	casinh	errno	gmtime	printf	vfscanf
754	casinhf	exit	hypotf	putc	vfwprintf
755	casinhl	exp	hypotl	putchar	vfwscanf
756	casinl	exp2	ilogb	puts	vprintf
757	catan	exp2f	ilogbf	putwc	vscanf
758	catanf	exp2l	ilogbl	putwchar	vsprintf
759	catanh	expf	imaxabs	qsort	vsscanf
760	catanh	expl	imaxdiv	raise	vswprintf
761	catanhf	expm1	is[a-z]*	rand	vswscanf
762	catanhf	expm1f	isblank	realloc	vwprintf
763	catanhl	expm1l	iswblank	remainderf	vwscanf
764	catanhl	fabs	labs	remainderl	wcrtomb
765	catanl	fabsf	ldexp	remove	wcs[a-z]*
766	cbrt	fabsl	ldexpf	remquo	wcstof
767	cbrtf	fclose	ldexpl	remquof	wcstoimax
768	cbrtl	fdim	ldiv	remquol	wcstold
769	ccos	fdimf	ldiv	rename	wcstoll
770	ccosf	fdiml	lgammaf	rewind	wcstoull
771	ccosh	feclearexcept	lgammal	rint	wcstoumax
772	ccoshf	fegetenv	llabs	rintf	wctob
773	ccoshl	fegetexceptflag	llrint	rintl	wctomb
774	ccosl	fegetround	llrintf	round	wctrans
775	ceil	feholdexcept	llrintl	roundf	wctype
776	ceilf	feof	llround	roundl	wcwidth
777	ceilf	feraiseexcept	llroundf	scalbln	wmem[a-z]*
778	ceil	ferror	llroundl	scalblnf	wprintf
779	ceil	fesetenv	localeconv	scalblnl	wscanf

780           **Note:**    The notation [a-z] indicates any lowercase letter in the portable character set. The  
781                           notation ' \* ' indicates any combination of digits, letters in the portable character set, or  
782                           underscore.

783           4. All functions and external identifiers defined in the Base Definitions volume of  
784           IEEE Std 1003.1-200x, Chapter 13, Headers are reserved for use as identifiers with external  
785           linkage.

786           5. All the identifiers defined in this volume of IEEE Std 1003.1-200x that have external linkage  
787           are always reserved for use as identifiers with external linkage.

788           No other identifiers are reserved.

789           Applications shall not declare or define identifiers with the same name as an identifier reserved  
790           in the same context. Since macro names are replaced whenever found, independent of scope and  
791           name space, macro names matching any of the reserved identifier names shall not be defined by

792 an application if any associated header is included.  
793 Except that the effect of each inclusion of `<assert.h>` depends on the definition of `NDEBUG`,  
794 headers may be included in any order, and each may be included more than once in a given  
795 scope, with no difference in effect from that of being included only once.  
796 If used, the application shall ensure that a header is included outside of any external declaration  
797 or definition, and it shall be first included before the first reference to any type or macro it  
798 defines, or to any function or object it declares. However, if an identifier is declared or defined in  
799 more than one header, the second and subsequent associated headers may be included after the  
800 initial reference to the identifier. Prior to the inclusion of a header, the application shall not  
801 define any macros with names lexically identical to symbols defined by that header.

## 802 2.3 Error Numbers

803 Most functions can provide an error number. The means by which each function provides its  
804 error numbers is specified in its description.

805 Some functions provide the error number in a variable accessed through the symbol *errno*. The  
806 symbol *errno*, defined by including the `<errno.h>` header, expands to a modifiable lvalue of type  
807 `int`. It is unspecified whether *errno* is a macro or an identifier declared with external linkage. If a  
808 macro definition is suppressed in order to access an actual object, or a program defines an  
809 identifier with the name *errno*, the behavior is undefined.

810 The value of *errno* should only be examined when it is indicated to be valid by a function's return  
811 value. No function in this volume of IEEE Std 1003.1-200x shall set *errno* to zero. For each thread  
812 of a process, the value of *errno* shall not be affected by function calls or assignments to *errno* by  
813 other threads.

814 Some functions return an error number directly as the function value. These functions return a  
815 value of zero to indicate success.

816 If more than one error occurs in processing a function call, any one of the possible errors may be  
817 returned, as the order of detection is undefined.

818 Implementations may support additional errors not included in this list, may generate errors  
819 included in this list under circumstances other than those described here, or may contain  
820 extensions or limitations that prevent some errors from occurring. The ERRORS section on each  
821 reference page specifies whether an error shall be returned, or whether it may be returned.  
822 Implementations shall not generate a different error number from the ones described here for  
823 error conditions described in this volume of IEEE Std 1003.1-200x, but may generate additional  
824 errors unless explicitly disallowed for a particular function.

825 Each implementation shall document, in the conformance document, situations in which each of  
826 the optional conditions defined in IEEE Std 1003.1-200x is detected. The conformance document  
827 may also contain statements that one or more of the optional error conditions are not detected.

828 For functions under the Threads option for which `[EINTR]` is not listed as a possible error  
829 condition in this volume of IEEE Std 1003.1-200x, an implementation shall not return an error  
830 code of `[EINTR]`.

831 The following symbolic names identify the possible error numbers, in the context of the  
832 functions specifically defined in this volume of IEEE Std 1003.1-200x; these general descriptions  
833 are more precisely defined in the ERRORS sections of the functions that return them. Only these  
834 symbolic names should be used in programs, since the actual value of the error number is  
835 unspecified. All values listed in this section shall be unique integer constant expressions with

836 type **int** suitable for use in **#if** preprocessing directives, except as noted below. The values for all  
 837 these names shall be found in the **<errno.h>** header defined in the Base Definitions volume of  
 838 IEEE Std 1003.1-200x. The actual values are unspecified by this volume of IEEE Std 1003.1-200x.

839 [E2BIG]

840 Argument list too long. The sum of the number of bytes used by the new process image's  
 841 argument list and environment list is greater than the system-imposed limit of {ARG\_MAX}  
 842 bytes.

843 or:

844 Lack of space in an output buffer.

845 or:

846 Argument is greater than the system-imposed maximum.

847 [EACCES]

848 Permission denied. An attempt was made to access a file in a way forbidden by its file  
 849 access permissions.

850 [EADDRINUSE]

851 Address in use. The specified address is in use.

852 [EADDRNOTAVAIL]

853 Address not available. The specified address is not available from the local system.

854 [EAFNOSUPPORT]

855 Address family not supported. The implementation does not support the specified address  
 856 family, or the specified address is not a valid address for the address family of the specified  
 857 socket.

858 [EAGAIN]

859 Resource temporarily unavailable. This is a temporary condition and later calls to the same  
 860 routine may complete normally.

861 [EALREADY]

862 Connection already in progress. A connection request is already in progress for the specified  
 863 socket.

864 [EBADF]

865 Bad file descriptor. A file descriptor argument is out of range, refers to no open file, or a  
 866 read (write) request is made to a file that is only open for writing (reading).

867 [EBADMSG]

868 XSR Bad message. During a *read()*, *getmsg()*, *getpmsg()*, or *ioctl()* I\_RECVFD request to a |  
 869 STREAMS device, a message arrived at the head of the STREAM that is inappropriate for |  
 870 the function receiving the message. |

871 *read()* Message waiting to be read on a STREAM is not a data message. |

872 *getmsg()* or *getpmsg()* |

873 A file descriptor was received instead of a control message. |

874 *ioctl()* Control or data information was received instead of a file descriptor when |  
 875 I\_RECVFD was specified. |

876 or:

877 Bad Message. The implementation has detected a corrupted message.

878	[EBUSY]
879	Resource busy. An attempt was made to make use of a system resource that is not currently
880	available, as it is being used by another process in a manner that would have conflicted with
881	the request being made by this process.
882	[ECANCELED]
883	Operation canceled. The associated asynchronous operation was canceled before
884	completion.
885	[ECHILD]
886	No child process. A <i>wait()</i> or <i>waitpid()</i> function was executed by a process that had no
887	existing or unwaited-for child process.
888	[ECONNABORTED]
889	Connection aborted. The connection has been aborted.
890	[ECONNREFUSED]
891	Connection refused. An attempt to connect to a socket was refused because there was no
892	process listening or because the queue of connection requests was full and the underlying
893	protocol does not support retransmissions.
894	[ECONNRESET]
895	Connection reset. The connection was forcibly closed by the peer.
896	[EDEADLK]
897	Resource deadlock would occur. An attempt was made to lock a system resource that
898	would have resulted in a deadlock situation.
899	[EDESTADDRREQ]
900	Destination address required. No bind address was established.
901	[EDOM]
902	Domain error. An input argument is outside the defined domain of the mathematical
903	function (defined in the ISO C standard).
904	[EDQUOT]
905	Reserved.
906	[EEXIST]
907	File exists. An existing file was mentioned in an inappropriate context; for example, as a
908	new link name in the <i>link()</i> function.
909	[EFAULT]
910	Bad address. The system detected an invalid address in attempting to use an argument of a
911	call. The reliable detection of this error cannot be guaranteed, and when not detected may
912	result in the generation of a signal, indicating an address violation, which is sent to the
913	process.
914	[EFBIG]
915	File too large. The size of a file would exceed the maximum file size of an implementation or
916	offset maximum established in the corresponding file description.
917	[EHOSTUNREACH]
918	Host is unreachable. The destination host cannot be reached (probably because the host is
919	down or a remote router cannot reach it).
920	[EIDRM]
921	Identifier removed. Returned during XSI interprocess communication if an identifier has
922	been removed from the system.

923	[EILSEQ]
924	Illegal byte sequence. A wide-character code has been detected that does not correspond to
925	a valid character, or a byte sequence does not form a valid wide-character code (defined in
926	the ISO C standard).
927	[EINPROGRESS]
928	Operation in progress. This code is used to indicate that an asynchronous operation has not
929	yet completed.
930	or:
931	O_NONBLOCK is set for the socket file descriptor and the connection cannot be
932	immediately established.
933	[EINTR]
934	Interrupted function call. An asynchronous signal was caught by the process during the
935	execution of an interruptible function. If the signal handler performs a normal return, the
936	interrupted function call may return this condition (see the Base Definitions volume of
937	IEEE Std 1003.1-200x, <signal.h>).
938	[EINVAL]
939	Invalid argument. Some invalid argument was supplied; for example, specifying an
940	undefined signal in a <i>signal()</i> function or a <i>kill()</i> function.
941	[EIO]
942	Input/output error. Some physical input or output error has occurred. This error may be
943	reported on a subsequent operation on the same file descriptor. Any other error-causing
944	operation on the same file descriptor may cause the [EIO] error indication to be lost.
945	[EISCONN]
946	Socket is connected. The specified socket is already connected.
947	[EISDIR]
948	Is a directory. An attempt was made to open a directory with write mode specified.
949	[ELOOP]
950	Symbolic link loop. A loop exists in symbolic links encountered during pathname
951	resolution. This error may also be returned if more than {SYMLOOP_MAX} symbolic links
952	are encountered during pathname resolution.
953	[EMFILE]
954	Too many open files. An attempt was made to open more than the maximum number of
955	{OPEN_MAX} file descriptors allowed in this process.
956	[EMLINK]
957	Too many links. An attempt was made to have the link count of a single file exceed
958	{LINK_MAX}.
959	[EMSGSIZE]
960	Message too large. A message sent on a transport provider was larger than an internal
961	message buffer or some other network limit.
962	or:
963	Inappropriate message buffer length.
964	[EMULTIHOP]
965	Reserved.

966	[ENAMETOOLONG]	
967	Filename too long. The length of a pathname exceeds {PATH_MAX}, or a pathname	
968	component is longer than {NAME_MAX}. This error may also occur when pathname	
969	substitution, as a result of encountering a symbolic link during pathname resolution, results	
970	in a pathname string the size of which exceeds {PATH_MAX}.	
971	[ENETDOWN]	
972	Network is down. The local network interface used to reach the destination is down.	
973	[ENETRESET]	
974	The connection was aborted by the network.	
975	[ENETUNREACH]	
976	Network unreachable. No route to the network is present.	
977	[ENFILE]	
978	Too many files open in system. Too many files are currently open in the system. The system	
979	has reached its predefined limit for simultaneously open files and temporarily cannot accept	
980	requests to open another one.	
981	[ENOBUFS]	
982	No buffer space available. Insufficient buffer resources were available in the system to	
983	perform the socket operation.	
984	XSR [ENODATA]	
985	No message available. No message is available on the STREAM head read queue.	
986	[ENODEV]	
987	No such device. An attempt was made to apply an inappropriate function to a device; for	
988	example, trying to read a write-only device such as a printer.	
989	[ENOENT]	
990	No such file or directory. A component of a specified pathname does not exist, or the	
991	pathname is an empty string.	
992	[ENOEXEC]	
993	Executable file format error. A request is made to execute a file that, although it has the	
994	appropriate permissions, is not in the format required by the implementation for executable	
995	files.	
996	[ENOLCK]	
997	No locks available. A system-imposed limit on the number of simultaneous file and record	
998	locks has been reached and no more are currently available.	
999	[ENOLINK]	
1000	Reserved.	
1001	[ENOMEM]	
1002	Not enough space. The new process image requires more memory than is allowed by the	
1003	hardware or system-imposed memory management constraints.	
1004	[ENOMSG]	
1005	No message of the desired type. The message queue does not contain a message of the	
1006	required type during XSI interprocess communication.	
1007	[ENOPROTOPT]	
1008	Protocol not available. The protocol option specified to <i>setsockopt()</i> is not supported by the	
1009	implementation.	

1010	[ENOSPC]	
1011		No space left on a device. During the <i>write()</i> function on a regular file or when extending a
1012		directory, there is no free space left on the device.
1013	XSR [ENOSR]	
1014		No STREAM resources. Insufficient STREAMS memory resources are available to perform a
1015		STREAMS-related function. This is a temporary condition; it may be recovered from if other
1016		processes release resources.
1017	XSR [ENOSTR]	
1018		Not a STREAM. A STREAM function was attempted on a file descriptor that was not
1019		associated with a STREAMS device.
1020	[ENOSYS]	
1021		Function not implemented. An attempt was made to use a function that is not available in
1022		this implementation.
1023	[ENOTCONN]	
1024		Socket not connected. The socket is not connected.
1025	[ENOTDIR]	
1026		Not a directory. A component of the specified pathname exists, but it is not a directory,
1027		when a directory was expected.
1028	[ENOTEMPTY]	
1029		Directory not empty. A directory other than an empty directory was supplied when an
1030		empty directory was expected.
1031	[ENOTSOCK]	
1032		Not a socket. The file descriptor does not refer to a socket.
1033	[ENOTSUP]	
1034		Not supported. The implementation does not support this feature of the Realtime Option
1035		Group.
1036	[ENOTTY]	
1037		Inappropriate I/O control operation. A control function has been attempted for a file or
1038		special file for which the operation is inappropriate.
1039	[ENXIO]	
1040		No such device or address. Input or output on a special file refers to a device that does not
1041		exist, or makes a request beyond the capabilities of the device. It may also occur when, for
1042		example, a tape drive is not on-line.
1043	[EOPNOTSUPP]	
1044		Operation not supported on socket. The type of socket (address family or protocol) does not
1045		support the requested operation.
1046	[EOVERFLOW]	
1047		Value too large to be stored in data type. An operation was attempted which would
1048		generate a value that is outside the range of values that can be represented in the relevant
1049		data type or that are allowed for a given data item.
1050	[EPERM]	
1051		Operation not permitted. An attempt was made to perform an operation limited to
1052		processes with appropriate privileges or to the owner of a file or other resource.
1053	[EPIPE]	
1054		Broken pipe. A write was attempted on a socket, pipe, or FIFO for which there is no process

1055		to read the data.
1056		[EPROTO]
1057		Protocol error. Some protocol error occurred. This error is device-specific, but is generally
1058		not related to a hardware failure.
1059		[EPROTONOSUPPORT]
1060		Protocol not supported. The protocol is not supported by the address family, or the protocol
1061		is not supported by the implementation.
1062		[EPROTOTYPE]
1063		Protocol wrong type for socket. The socket type is not supported by the protocol.
1064		[ERANGE]
1065		Result too large or too small. The result of the function is too large (overflow) or too small
1066		(underflow) to be represented in the available space (defined in the ISO C standard).
1067		[EROFS]
1068		Read-only file system. An attempt was made to modify a file or directory on a file system
1069		that is read-only.
1070		[ESPIPE]
1071		Invalid seek. An attempt was made to access the file offset associated with a pipe or FIFO.
1072		[ESRCH]
1073		No such process. No process can be found corresponding to that specified by the given
1074		process ID.
1075		[ESTALE]
1076		Reserved.
1077	XSR	[ETIME]
1078		STREAM <i>ioctl()</i> timeout. The timer set for a STREAMS <i>ioctl()</i> call has expired. The cause of
1079		this error is device-specific and could indicate either a hardware or software failure, or a
1080		timeout value that is too short for the specific operation. The status of the <i>ioctl()</i> operation
1081		is unspecified.
1082		[ETIMEDOUT]
1083		Connection timed out. The connection to a remote machine has timed out. If the connection
1084		timed out during execution of the function that reported this error (as opposed to timing
1085		out prior to the function being called), it is unspecified whether the function has completed
1086		some or all of the documented behavior associated with a successful completion of the
1087		function.
1088		or:
1089		Operation timed out. The time limit associated with the operation was exceeded before the
1090		operation completed.
1091		[ETXTBSY]
1092		Text file busy. An attempt was made to execute a pure-procedure program that is currently
1093		open for writing, or an attempt has been made to open for writing a pure-procedure
1094		program that is being executed.
1095		[EWOULDBLOCK]
1096		Operation would block. An operation on a socket marked as non-blocking has encountered
1097		a situation such as no data available that otherwise would have caused the function to
1098		suspend execution.

1099 A conforming implementation may assign the same values for [EWOULDBLOCK] and  
 1100 [EAGAIN].  
 1101 [EXDEV]  
 1102 Improper link. A link to a file on another file system was attempted.

### 1103 2.3.1 Additional Error Numbers

1104 Additional implementation-defined error numbers may be defined in <errno.h>.

## 1105 2.4 Signal Concepts

### 1106 2.4.1 Signal Generation and Delivery

1107 A signal is said to be *generated* for (or sent to) a process or thread when the event that causes the  
 1108 signal first occurs. Examples of such events include detection of hardware faults, timer  
 1109 RTS expiration, signals generated via the `sigevent` structure and terminal activity, as well as  
 1110 RTS invocations of the `kill()` and `sigqueue()` functions. In some circumstances, the same event  
 1111 generates signals for multiple processes.

1112 At the time of generation, a determination shall be made whether the signal has been generated  
 1113 for the process or for a specific thread within the process. Signals which are generated by some  
 1114 action attributable to a particular thread, such as a hardware fault, shall be generated for the  
 1115 thread that caused the signal to be generated. Signals that are generated in association with a  
 1116 process ID or process group ID or an asynchronous event, such as terminal activity, shall be  
 1117 generated for the process.

1118 Each process has an action to be taken in response to each signal defined by the system (see  
 1119 Section 2.4.3 (on page 480)). A signal is said to be *delivered* to a process when the appropriate  
 1120 action for the process and signal is taken. A signal is said to be *accepted* by a process when the  
 1121 signal is selected and returned by one of the `sigwait()` functions.

1122 During the time between the generation of a signal and its delivery or acceptance, the signal is  
 1123 said to be *pending*. Ordinarily, this interval cannot be detected by an application. However, a  
 1124 signal can be *blocked* from delivery to a thread. If the action associated with a blocked signal is  
 1125 anything other than to ignore the signal, and if that signal is generated for the thread, the signal  
 1126 shall remain pending until it is unblocked, it is accepted when it is selected and returned by a  
 1127 call to the `sigwait()` function, or the action associated with it is set to ignore the signal. Signals  
 1128 generated for the process shall be delivered to exactly one of those threads within the process  
 1129 which is in a call to a `sigwait()` function selecting that signal or has not blocked delivery of the  
 1130 signal. If there are no threads in a call to a `sigwait()` function selecting that signal, and if all  
 1131 threads within the process block delivery of the signal, the signal shall remain pending on the  
 1132 process until a thread calls a `sigwait()` function selecting that signal, a thread unblocks delivery  
 1133 of the signal, or the action associated with the signal is set to ignore the signal. If the action  
 1134 associated with a blocked signal is to ignore the signal and if that signal is generated for the  
 1135 process, it is unspecified whether the signal is discarded immediately upon generation or  
 1136 remains pending.

1137 Each thread has a *signal mask* that defines the set of signals currently blocked from delivery to it. |  
 1138 The signal mask for a thread shall be initialized from that of its parent or creating thread, or from |  
 1139 the corresponding thread in the parent process if the thread was created as the result of a call to |  
 1140 `fork()`. The `sigaction()`, `sigprocmask()`, and `sigsuspend()` functions control the manipulation of the |  
 1141 signal mask.

1142 The determination of which action is taken in response to a signal is made at the time the signal  
 1143 is delivered, allowing for any changes since the time of generation. This determination is  
 1144 independent of the means by which the signal was originally generated. If a subsequent  
 1145 occurrence of a pending signal is generated, it is implementation-defined as to whether the  
 1146 RTS signal is delivered or accepted more than once in circumstances other than those in which  
 1147 queuing is required under the Realtime Signals Extension option. The order in which multiple,  
 1148 simultaneously pending signals outside the range SIGRTMIN to SIGRTMAX are delivered to or  
 1149 accepted by a process is unspecified.

1150 When any stop signal (SIGSTOP, SIGTSTP, SIGTTIN, SIGTTOU) is generated for a process, any  
 1151 pending SIGCONT signals for that process shall be discarded. Conversely, when SIGCONT is  
 1152 generated for a process, all pending stop signals for that process shall be discarded. When  
 1153 SIGCONT is generated for a process that is stopped, the process shall be continued, even if the  
 1154 SIGCONT signal is blocked or ignored. If SIGCONT is blocked and not ignored, it shall remain  
 1155 pending until it is either unblocked or a stop signal is generated for the process.

1156 An implementation shall document any condition not specified by this volume of  
 1157 IEEE Std 1003.1-200x under which the implementation generates signals.

## 1158 2.4.2 Realtime Signal Generation and Delivery

1159 RTS This section describes extensions to support realtime signal generation and delivery. This  
 1160 functionality is dependent on support of the Realtime Signals Extension option (and the rest of  
 1161 this section is not further shaded for this option).

1162 Some signal-generating functions, such as high-resolution timer expiration, asynchronous I/O  
 1163 completion, interprocess message arrival, and the *sigqueue()* function, support the specification  
 1164 of an application-defined value, either explicitly as a parameter to the function or in a **sigevent**  
 1165 structure parameter. The **sigevent** structure is defined in <**signal.h**> and contains at least the  
 1166 following members:

1167

1168

1169

1170

1171

1172

1173

Member Type	Member Name	Description
<b>int</b>	<i>sigev_notify</i>	Notification type.
<b>int</b>	<i>sigev_signo</i>	Signal number.
<b>union signal</b>	<i>sigev_value</i>	Signal value.
<b>void*(unsigned signal)</b>	<i>sigev_notify_function</i>	Notification function.
<b>(pthread_attr_t*)</b>	<i>sigev_notify_attributes</i>	Notification attributes.

1174 The *sigev\_notify* member specifies the notification mechanism to use when an asynchronous  
 1175 event occurs. This volume of IEEE Std 1003.1-200x defines the following values for the  
 1176 *sigev\_notify* member:

1177 SIGEV\_NONE No asynchronous notification shall be delivered when the event of  
 1178 interest occurs.

1179 SIGEV\_SIGNAL The signal specified in *sigev\_signo* shall be generated for the process when  
 1180 the event of interest occurs. If the implementation supports the Realtime  
 1181 Signals Extension option and if the SA\_SIGINFO flag is set for that signal  
 1182 number, then the signal shall be queued to the process and the value  
 1183 specified in *sigev\_value* shall be the *si\_value* component of the generated  
 1184 signal. If SA\_SIGINFO is not set for that signal number, it is unspecified  
 1185 whether the signal is queued and what value, if any, is sent.

1186 SIGEV\_THREAD A notification function shall be called to perform notification.

1187 An implementation may define additional notification mechanisms.

1188 The *sigev\_signo* member specifies the signal to be generated. The *sigev\_value* member is the  
1189 application-defined value to be passed to the signal-catching function at the time of the signal  
1190 delivery or to be returned at signal acceptance as the *si\_value* member of the **siginfo\_t** structure.

1191 The **signal** union is defined in `<signal.h>` and contains at least the following members:

1192

1193

1194

1195

Member Type	Member Name	Description
<b>int</b>	<i>sival_int</i>	Integer signal value.
<b>void*</b>	<i>sival_ptr</i>	Pointer signal value.

1196 The *sival\_int* member shall be used when the application-defined value is of type **int**; the  
1197 *sival\_ptr* member shall be used when the application-defined value is a pointer.

1198 When a signal is generated by the *sigqueue()* function or any signal-generating function that  
1199 supports the specification of an application-defined value, the signal shall be marked pending  
1200 and, if the SA\_SIGINFO flag is set for that signal, the signal shall be queued to the process along  
1201 with the application-specified signal value. Multiple occurrences of signals so generated are  
1202 queued in FIFO order. It is unspecified whether signals so generated are queued when the  
1203 SA\_SIGINFO flag is not set for that signal.

1204 Signals generated by the *kill()* function or other events that cause signals to occur, such as  
1205 detection of hardware faults, *alarm()* timer expiration, or terminal activity, and for which the  
1206 implementation does not support queuing, shall have no effect on signals already queued for the  
1207 same signal number.

1208 When multiple unblocked signals, all in the range SIGRTMIN to SIGRTMAX, are pending, the  
1209 behavior shall be as if the implementation delivers the pending unblocked signal with the lowest  
1210 signal number within that range. No other ordering of signal delivery is specified.

1211 If, when a pending signal is delivered, there are additional signals queued to that signal number,  
1212 the signal shall remain pending. Otherwise, the pending indication shall be reset.

1213 Multi-threaded programs can use an alternate event notification mechanism. When a  
1214 notification is processed, and the *sigev\_notify* member of the **sigevent** structure has the value  
1215 SIGEV\_THREAD, the function *sigev\_notify\_function* is called with parameter *sigev\_value*.

1216 The function shall be executed in an environment as if it were the *start\_routine* for a newly  
1217 created thread with thread attributes specified by *sigev\_notify\_attributes*. If *sigev\_notify\_attributes*  
1218 is NULL, the behavior shall be as if the thread were created with the *detachstate* attribute set to  
1219 PTHREAD\_CREATE\_DETACHED. Supplying an attributes structure with a *detachstate* attribute  
1220 of PTHREAD\_CREATE\_JOINABLE results in undefined behavior. The signal mask of this  
1221 thread is implementation-defined.

### 1222 2.4.3 Signal Actions

1223 There are three types of action that can be associated with a signal: SIG\_DFL, SIG\_IGN, or a  
1224 pointer to a function. Initially, all signals shall be set to SIG\_DFL or SIG\_IGN prior to entry of  
1225 the *main()* routine (see the *exec* functions). The actions prescribed by these values are as follows:

1226 SIG\_DFL Signal-specific default action.

1227 The default actions for the signals defined in this volume of IEEE Std 1003.1-200x  
1228 RTS are specified under `<signal.h>`. If the Realtime Signals Extension option is  
1229 supported, the default actions for the realtime signals in the range SIGRTMIN to  
1230 SIGRTMAX shall be to terminate the process abnormally.

1231		If the default action is to stop the process, the execution of that process is temporarily suspended. When a process stops, a SIGCHLD signal shall be
1232		generated for its parent process, unless the parent process has set the
1233		SA_NOCLDSTOP flag. While a process is stopped, any additional signals that are
1234		sent to the process shall not be delivered until the process is continued, except
1235		SIGKILL which always terminates the receiving process. A process that is a
1236		member of an orphaned process group shall not be allowed to stop in response to
1237		the SIGTSTP, SIGTTIN, or SIGTTOU signals. In cases where delivery of one of
1238		these signals would stop such a process, the signal shall be discarded.
1239		
1240		Setting a signal action to SIG_DFL for a signal that is pending, and whose default
1241		action is to ignore the signal (for example, SIGCHLD), shall cause the pending
1242		signal to be discarded, whether or not it is blocked.
1243		The default action for SIGCONT is to resume execution at the point where the
1244	RTS	process was stopped, after first handling any pending unblocked signals. If the
1245		Realtime Signals Extension option is supported, any queued values pending shall
1246		be discarded and the resources used to queue them shall be released and returned
1247		to the system for other use. When a stopped process is continued, a SIGCHLD
1248		signal may be generated for its parent process, unless the parent process has set
1249		the SA_NOCLDSTOP flag.
1250	SIG_IGN	Ignore signal.
1251		Delivery of the signal shall have no effect on the process. The behavior of a process
1252	RTS	is undefined after it ignores a SIGFPE, SIGILL, SIGSEGV, or SIGBUS signal that
1253	RTS	was not generated by <i>kill()</i> , <i>sigqueue()</i> , or <i>raise()</i> .
1254		The system shall not allow the action for the signals SIGKILL or SIGSTOP to be set
1255		to SIG_IGN.
1256		Setting a signal action to SIG_IGN for a signal that is pending shall cause the
1257		pending signal to be discarded, whether or not it is blocked.
1258		If a process sets the action for the SIGCHLD signal to SIG_IGN, the behavior is
1259	XSI	unspecified, except as specified below.
1260		If the action for the SIGCHLD signal is set to SIG_IGN, child processes of the
1261		calling processes shall not be transformed into zombie processes when they
1262		terminate. If the calling process subsequently waits for its children, and the process
1263		has no unwaited-for children that were transformed into zombie processes, it shall
1264		block until all of its children terminate, and <i>wait()</i> , <i>waitid()</i> , and <i>waitpid()</i> shall fail
1265		and set <i>errno</i> to [ECHILD].
1266	RTS	If the Realtime Signals Extension option is supported, any queued values pending
1267		shall be discarded and the resources used to queue them shall be released and
1268		made available to queue other signals.
1269		<i>pointer to a function</i>
1270		Catch signal.
1271		On delivery of the signal, the receiving process is to execute the signal-catching
1272		function at the specified address. After returning from the signal-catching function,
1273		the receiving process shall resume execution at the point at which it was
1274		interrupted.
1275		If the SA_SIGINFO flag for the signal is cleared, the signal-catching function shall
1276		be entered as a C-language function call as follows:

1277 `void func(int signo);`

1278 XSI|RTS If the SA\_SIGINFO flag for the signal is set, the signal-catching function shall be  
1279 entered as a C-language function call as follows:

1280 `void func(int signo, siginfo_t *info, void *context);`

1281 where *func* is the specified signal-catching function, *signo* is the signal number of  
1282 the signal being delivered, and *info* is a pointer to a **siginfo\_t** structure defined in  
1283 **<signal.h>** containing at least the following members:

Member Type	Member Name	Description
int	<i>si_signo</i>	Signal number.
int	<i>si_code</i>	Cause of the signal.
union signal	<i>si_value</i>	Signal value.

1289 The *si\_signo* member shall contain the signal number. This shall be the same as the  
1290 *signo* parameter. The *si\_code* member shall contain a code identifying the cause of  
1291 the signal. The following values are defined for *si\_code*:

1292 SI\_USER The signal was sent by the *kill()* function. The implementation  
1293 may set *si\_code* to SI\_USER if the signal was sent by the *raise()* or  
1294 *abort()* functions or any similar functions provided as  
1295 implementation extensions.

1296 RTS SI\_QUEUE The signal was sent by the *sigqueue()* function.

1297 RTS SI\_TIMER The signal was generated by the expiration of a timer set by  
1298 *timer\_settime()*.

1299 RTS SI\_ASYNCIO The signal was generated by the completion of an asynchronous  
1300 I/O request.

1301 RTS SI\_MESGQ The signal was generated by the arrival of a message on an  
1302 empty message queue.

1303 If the signal was not generated by one of the functions or events listed above, the  
1304 *si\_code* shall be set to an implementation-defined value that is not equal to any of  
1305 the values defined above.

1306 RTS If the Realtime Signals Extension is supported, and *si\_code* is one of SI\_QUEUE,  
1307 SI\_TIMER, SI\_ASYNCIO, or SI\_MESGQ, then *si\_value* shall contain the  
1308 application-specified signal value. Otherwise, the contents of *si\_value* are  
1309 undefined.

1310 The behavior of a process is undefined after it returns normally from a signal-  
1311 XSI catching function for a SIGBUS, SIGFPE, SIGILL, or SIGSEGV signal that was not  
1312 RTS generated by *kill()*, *sigqueue()*, or *raise()*.

1313 The system shall not allow a process to catch the signals SIGKILL and SIGSTOP.

1314 If a process establishes a signal-catching function for the SIGCHLD signal while it  
1315 has a terminated child process for which it has not waited, it is unspecified  
1316 whether a SIGCHLD signal is generated to indicate that child process.

1317 When signal-catching functions are invoked asynchronously with process  
1318 execution, the behavior of some of the functions defined by this volume of  
1319 IEEE Std 1003.1-200x is unspecified if they are called from a signal-catching  
1320 function.

1321 The following table defines a set of functions that shall be either reentrant or non-  
 1322 interruptible by signals and shall be async-signal-safe. Therefore applications may  
 1323 invoke them, without restriction, from signal-catching functions:

### 1324 **Notes to Reviewers**

1325 *This section with side shading will not appear in the final copy. - Ed.*

1326 The contents of the following tables need to be reviewed.

1327	<code>_Exit()</code>	<code>fdatasync()</code>	<code>posix_trace_event()</code>	<code>sigsuspend()</code>
1328	<code>_exit()</code>	<code>fork()</code>	<code>raise()</code>	<code>stat()</code>
1329	<code>access()</code>	<code>fpathconf()</code>	<code>read()</code>	<code>symlink()</code>
1330	<code>aio_error()</code>	<code>fstat()</code>	<code>readlink()</code>	<code>sysconf()</code>
1331	<code>aio_return()</code>	<code>fsync()</code>	<code>rename()</code>	<code>tcdrain()</code>
1332	<code>aio_suspend()</code>	<code>ftruncate()</code>	<code>rmdir()</code>	<code>tcflow()</code>
1333	<code>alarm()</code>	<code>getegid()</code>	<code>sem_post()</code>	<code>tcflush()</code>
1334	<code>cfgetispeed()</code>	<code>geteuid()</code>	<code>setgid()</code>	<code>tcgetattr()</code>
1335	<code>cfgetospeed()</code>	<code>getgid()</code>	<code>setpgid()</code>	<code>tcgetpgrp()</code>
1336	<code>cfsetispeed()</code>	<code>getgroups()</code>	<code>setsid()</code>	<code>tcsendbreak()</code>
1337	<code>cfsetospeed()</code>	<code>getpgrp()</code>	<code>setuid()</code>	<code>tcsetattr()</code>
1338	<code>chdir()</code>	<code>getpid()</code>	<code>sigaction()</code>	<code>tcsetpgrp()</code>
1339	<code>chmod()</code>	<code>getppid()</code>	<code>sigaddset()</code>	<code>time()</code>
1340	<code>chown()</code>	<code>getuid()</code>	<code>sigdelset()</code>	<code>timer_getoverrun()</code>
1341	<code>clock_gettime()</code>	<code>kill()</code>	<code>sigemptyset()</code>	<code>timer_gettime()</code>
1342	<code>close()</code>	<code>link()</code>	<code>sigfillset()</code>	<code>timer_settime()</code>
1343	<code>creat()</code>	<code>lseek()</code>	<code>sigismember()</code>	<code>times()</code>
1344	<code>dup()</code>	<code>lstat()</code>	<code>sleep()</code>	<code>umask()</code>
1345	<code>dup2()</code>	<code>mkdir()</code>	<code>signal()</code>	<code>uname()</code>
1346	<code>execle()</code>	<code>mkfifo()</code>	<code>sigpause()</code>	<code>unlink()</code>
1347	<code>execve()</code>	<code>open()</code>	<code>sigpending()</code>	<code>utime()</code>
1348	<code>fchmod()</code>	<code>pathconf()</code>	<code>sigprocmask()</code>	<code>wait()</code>
1349	<code>fchown()</code>	<code>pause()</code>	<code>sigqueue()</code>	<code>waitpid()</code>
1350	<code>fcntl()</code>	<code>pipe()</code>	<code>sigset()</code>	<code>write()</code>

1351 All functions not in the above table are considered to be unsafe with respect to  
 1352 signals. In the presence of signals, all functions defined by this volume of  
 1353 IEEE Std 1003.1-200x shall behave as defined when called from or interrupted by a  
 1354 signal-catching function, with a single exception: when a signal interrupts an  
 1355 unsafe function and the signal-catching function calls an unsafe function, the  
 1356 behavior is undefined.

1357 When a signal is delivered to a thread, if the action of that signal specifies termination, stop, or  
 1358 continue, the entire process shall be terminated, stopped, or continued, respectively.

#### 1359 2.4.4 Signal Effects on Other Functions

1360 Signals affect the behavior of certain functions defined by this volume of IEEE Std 1003.1-200x if  
1361 delivered to a process while it is executing such a function. If the action of the signal is to  
1362 terminate the process, the process shall be terminated and the function shall not return. If the  
1363 action of the signal is to stop the process, the process shall stop until continued or terminated.  
1364 Generation of a SIGCONT signal for the process shall cause the process to be continued, and the  
1365 original function shall continue at the point the process was stopped. If the action of the signal is  
1366 to invoke a signal-catching function, the signal-catching function shall be invoked; in this case  
1367 the original function is said to be *interrupted* by the signal. If the signal-catching function  
1368 executes a **return** statement, the behavior of the interrupted function shall be as described  
1369 individually for that function, except as noted for unsafe functions. Signals that are ignored shall  
1370 not affect the behavior of any function; signals that are blocked shall not affect the behavior of  
1371 any function until they are unblocked and then delivered, except as specified for the *sigpending()*  
1372 and *sigwait()* functions.

#### 1373 2.5 Standard I/O Streams

1374 A stream is associated with an external file (which may be a physical device) by *opening* a file,  
1375 which may involve *creating* a new file. Creating an existing file causes its former contents to be  
1376 discarded if necessary. If a file can support positioning requests, (such as a disk file, as opposed  
1377 to a terminal), then a *file position indicator* associated with the stream is positioned at the start  
1378 (byte number 0) of the file, unless the file is opened with append mode, in which case it is  
1379 implementation-defined whether the file position indicator is initially positioned at the  
1380 beginning or end of the file. The file position indicator is maintained by subsequent reads,  
1381 writes, and positioning requests, to facilitate an orderly progression through the file. All input  
1382 takes place as if bytes were read by successive calls to *fgetc()*; all output takes place as if bytes  
1383 were written by successive calls to *fputc()*.

1384 When a stream is *unbuffered*, bytes are intended to appear from the source or at the destination  
1385 as soon as possible; otherwise, bytes may be accumulated and transmitted as a block. When a  
1386 stream is *fully buffered*, bytes are intended to be transmitted as a block when a buffer is filled.  
1387 When a stream is *line buffered*, bytes are intended to be transmitted as a block when a newline  
1388 byte is encountered. Furthermore, bytes are intended to be transmitted as a block when a buffer  
1389 is filled, when input is requested on an unbuffered stream, or when input is requested on a line-  
1390 buffered stream that requires the transmission of bytes. Support for these characteristics is  
1391 implementation-defined, and may be affected via *setbuf()* and *setvbuf()*.

1392 A file may be disassociated from a controlling stream by *closing* the file. Output streams are  
1393 flushed (any unwritten buffer contents are transmitted) before the stream is disassociated from  
1394 the file. The value of a pointer to a **FILE** object is unspecified after the associated file is closed  
1395 (including the standard streams).

1396 A file may be subsequently reopened, by the same or another program execution, and its  
1397 contents reclaimed or modified (if it can be repositioned at its start). If the *main()* function  
1398 returns to its original caller, or if the *exit()* function is called, all open files are closed (hence all  
1399 output streams are flushed) before program termination. Other paths to program termination,  
1400 such as calling *abort()*, need not close all files properly.

1401 The address of the **FILE** object used to control a stream may be significant; a copy of a **FILE**  
1402 object need not necessarily serve in place of the original.

1403 At program start-up, three streams are predefined and need not be opened explicitly: *standard*  
1404 *input* (for reading conventional input), *standard output* (for writing conventional output), and

1405 *standard error* (for writing diagnostic output). When opened, the standard error stream is not  
 1406 fully buffered; the standard input and standard output streams are fully buffered if and only if  
 1407 the stream can be determined not to refer to an interactive device.

### 1408 2.5.1 Interaction of File Descriptors and Standard I/O Streams

1409 cx This section describes the interaction of file descriptors and standard I/O streams. This  
 1410 functionality is an extension to the ISO C standard (and the rest of this section is not further CX  
 1411 shaded).

1412 An open file description may be accessed through a file descriptor, which is created using  
 1413 functions such as *open()* or *pipe()*, or through a stream, which is created using functions such as  
 1414 *fopen()* or *popen()*. Either a file descriptor or a stream is called a *handle* on the open file  
 1415 description to which it refers; an open file description may have several handles.

1416 Handles can be created or destroyed by explicit user action, without affecting the underlying  
 1417 open file description. Some of the ways to create them include *fcntl()*, *dup()*, *fdopen()*, *fileno()*,  
 1418 and *fork()*. They can be destroyed by at least *fclose()*, *close()*, and the *exec* functions.

1419 A file descriptor that is never used in an operation that could affect the file offset (for example,  
 1420 *read()*, *write()*, or *lseek()*) is not considered a handle for this discussion, but could give rise to one  
 1421 (for example, as a consequence of *fdopen()*, *dup()*, or *fork()*). This exception does not include the  
 1422 file descriptor underlying a stream, whether created with *fopen()* or *fdopen()*, so long as it is not  
 1423 used directly by the application to affect the file offset. The *read()* and *write()* functions  
 1424 implicitly affect the file offset; *lseek()* explicitly affects it.

1425 The result of function calls involving any one handle (the *active handle*) is defined elsewhere in  
 1426 this volume of IEEE Std 1003.1-200x, but if two or more handles are used, and any one of them is  
 1427 a stream, the application shall ensure that their actions are coordinated as described below. If  
 1428 this is not done, the result is undefined.

1429 A handle which is a stream is considered to be closed when either an *fclose()* or *freopen()* is  
 1430 executed on it (the result of *freopen()* is a new stream, which cannot be a handle on the same  
 1431 open file description as its previous value), or when the process owning that stream terminates  
 1432 with *exit()* or *abort()*. A file descriptor is closed by *close()*, *\_exit()*, or the *exec* functions when  
 1433 FD\_CLOEXEC is set on that file descriptor.

1434 For a handle to become the active handle, the application shall ensure that the actions below are  
 1435 performed between the last use of the handle (the current active handle) and the first use of the  
 1436 second handle (the future active handle). The second handle then becomes the active handle. All  
 1437 activity by the application affecting the file offset on the first handle shall be suspended until it  
 1438 again becomes the active file handle. (If a stream function has as an underlying function one that  
 1439 affects the file offset, the stream function shall be considered to affect the file offset.)

1440 The handles need not be in the same process for these rules to apply.

1441 Note that after a *fork()*, two handles exist where one existed before. The application shall ensure  
 1442 that, if both handles can ever be accessed, they are both in a state where the other could become  
 1443 the active handle first. The application shall prepare for a *fork()* exactly as if it were a change of  
 1444 active handle. (If the only action performed by one of the processes is one of the *exec* functions or  
 1445 *\_exit()* (not *exit()*), the handle is never accessed in that process.)

1446 For the first handle, the first applicable condition below applies. After the actions required  
 1447 below are taken, if the handle is still open, the application can close it.

- 1448 • If it is a file descriptor, no action is required.

- 1449       • If the only further action to be performed on any handle to this open file descriptor is to close  
1450       it, no action need be taken.
- 1451       • If it is a stream which is unbuffered, no action need be taken.
- 1452       • If it is a stream which is line buffered, and the last byte written to the stream was a newline  
1453       (that is, as if a:
- 1454                 putc( '\n' )
- 1455       was the most recent operation on that stream), no action need be taken.
- 1456       • If it is a stream which is open for writing or appending (but not also open for reading), the  
1457       application shall either perform an *flush()*, or the stream shall be closed.
- 1458       • If the stream is open for reading and it is at the end of the file (*feof()* is true), no action need  
1459       be taken.
- 1460       • If the stream is open with a mode that allows reading and the underlying open file  
1461       description refers to a device that is capable of seeking, the application shall either perform  
1462       an *flush()*, or the stream shall be closed.
- 1463       Otherwise, the result is undefined.
- 1464       For the second handle:
- 1465       • If any previous active handle has been used by a function that explicitly changed the file  
1466       offset, except as required above for the first handle, the application shall perform an *lseek()* or  
1467       *fseek()* (as appropriate to the type of handle) to an appropriate location.
- 1468       If the active handle ceases to be accessible before the requirements on the first handle, above,  
1469       have been met, the state of the open file description becomes undefined. This might occur during  
1470       functions such as a *fork()* or *\_exit()*.
- 1471       The *exec* functions make inaccessible all streams that are open at the time they are called,  
1472       independent of which streams or file descriptors may be available to the new process image.
- 1473       When these rules are followed, regardless of the sequence of handles used, implementations  
1474       shall ensure that an application, even one consisting of several processes, shall yield correct  
1475       results: no data shall be lost or duplicated when writing, and all data shall be written in order,  
1476       except as requested by seeks. It is implementation-defined whether, and under what conditions,  
1477       all input is seen exactly once.
- 1478       If the rules above are not followed, the result is unspecified.
- 1479       Each function that operates on a stream is said to have zero or more *underlying functions*. This  
1480       means that the stream function shares certain traits with the underlying functions, but does not  
1481       require that there be any relation between the implementations of the stream function and its  
1482       underlying functions.

## 1483 2.5.2 Stream Orientation and Encoding Rules

1484       For conformance to the ISO/IEC 9899:1999 standard, the definition of a stream includes an  
1485       *orientation*. After a stream is associated with an external file, but before any operations are  
1486       performed on it, the stream is without orientation. Once a wide-character input/output function  
1487       has been applied to a stream without orientation, the stream shall become *wide-oriented*.  
1488       Similarly, once a byte input/output function has been applied to a stream without orientation,  
1489       the stream shall become *byte-oriented*. Only a call to the *freopen()* function or the *fwide()* function  
1490       can otherwise alter the orientation of a stream.

1491 A successful call to *freopen()* shall remove any orientation. The three predefined streams *standard*  
 1492 *input*, *standard output*, and *standard error* shall be unoriented at program start-up.

1493 Byte input/output functions cannot be applied to a wide-oriented stream, and wide-character  
 1494 input/output functions cannot be applied to a byte-oriented stream. The remaining stream  
 1495 operations shall not affect and shall not be affected by a stream's orientation, except for the  
 1496 following additional restrictions:

- 1497 • Binary wide-oriented streams have the file positioning restrictions ascribed to both text and  
 1498 binary streams.
- 1499 • For wide-oriented streams, after a successful call to a file-positioning function that leaves the  
 1500 file position indicator prior to the end-of-file, a wide-character output function can overwrite  
 1501 a partial character; any file contents beyond the byte(s) written are henceforth undefined.

1502 Each wide-oriented stream has an associated **mbstate\_t** object that stores the current parse state  
 1503 of the stream. A successful call to *fgetpos()* shall store a representation of the value of this  
 1504 **mbstate\_t** object as part of the value of the **fpos\_t** object. A later successful call to *fsetpos()* using  
 1505 the same stored **fpos\_t** value shall restore the value of the associated **mbstate\_t** object as well as  
 1506 the position within the controlled stream.

1507 Implementations that support multiple encoding rules associate an encoding rule with the  
 1508 stream. The encoding rule shall be determined by the setting of the *LC\_CTYPE* category in the  
 1509 current locale at the time when the stream becomes wide-oriented. If a wide-character  
 1510 input/output function is applied to a byte-oriented stream, the encoding rule used is undefined.  
 1511 As with the stream's orientation, the encoding rule associated with a stream cannot be changed  
 1512 once it has been set, except by a successful call to *freopen()* which clears the encoding rule and  
 1513 resets the orientation to unoriented.

1514 Although both text and binary wide-oriented streams are conceptually sequences of wide  
 1515 characters, the external file associated with a wide-oriented stream is a sequence of (possibly  
 1516 multi-byte) characters generalized as follows:

- 1517 • Multi-byte encodings within files may contain embedded null bytes (unlike multi-byte  
 1518 encodings valid for use internal to the program).
- 1519 • A file need not begin nor end in the initial shift state.

1520 Moreover, the encodings used for characters may differ among files. Both the nature and choice  
 1521 of such encodings are implementation-defined.

1522 The wide-character input functions read characters from the stream and convert them to wide  
 1523 characters as if they were read by successive calls to the *fgetwc()* function. Each conversion shall  
 1524 occur as if by a call to the *mbrtowc()* function, with the conversion state described by the stream's  
 1525 own **mbstate\_t** object, except the encoding rule associated with the stream is used instead of the  
 1526 encoding rule implied by the *LC\_CTYPE* category of the current locale.

1527 The wide-character output functions convert wide characters to (possibly multi-byte) characters  
 1528 and write them to the stream as if they were written by successive calls to the *fputwc()* function.  
 1529 Each conversion shall occur as if by a call to the *wcrtomb()* function, with the conversion state  
 1530 described by the stream's own **mbstate\_t** object, except the encoding rule associated with the  
 1531 stream is used instead of the encoding rule implied by the *LC\_CTYPE* category of the current  
 1532 locale.

1533 An *encoding error* shall occur if the character sequence presented to the underlying *mbrtowc()*  
 1534 function does not form a valid (generalized) character, or if the code value passed to the  
 1535 underlying *wcrtomb()* function does not correspond to a valid (generalized) character. The  
 1536 wide-character input/output functions and the byte input/output functions store the value of

1537 the macro [EILSEQ] in *errno* if and only if an encoding error occurs.

## 1538 **2.6 STREAMS**

1539 XSR STREAMS functionality is provided on implementations supporting the XSI STREAMS Option  
1540 Group. This functionality is dependent on support of the XSI STREAMS option (and the rest of  
1541 this section is not further shaded for this option).

1542 STREAMS provides a uniform mechanism for implementing networking services and other  
1543 character-based I/O. The STREAMS function provides direct access to protocol modules.  
1544 STREAMS modules are unspecified objects. Access to STREAMS modules is provided by  
1545 interfaces in IEEE Std 1003.1-200x. Creation of STREAMS modules is outside the scope of  
1546 IEEE Std 1003.1-200x.

1547 A STREAM is typically a full-duplex connection between a process and an open device or  
1548 pseudo-device. However, since pipes may be STREAMS-based, a STREAM can be a full-duplex  
1549 connection between two processes. The STREAM itself exists entirely within the implementation  
1550 and provides a general character I/O function for processes. It optionally includes one or more  
1551 intermediate processing modules that are interposed between the process end of the STREAM  
1552 (STREAM head) and a device driver at the end of the STREAM (STREAM end).

1553 STREAMS I/O is based on messages. There are three types of message:

- 1554 • *Data messages* containing actual data for input or output
- 1555 • *Control data* containing instructions for the STREAMS modules and underlying  
1556 implementation
- 1557 • Other messages, which include file descriptors

1558 The interface between the STREAM and the rest of the implementation is provided by a set of  
1559 functions at the STREAM head. When a process calls *write()*, *writew()*, *putmsg()*, *putpmsg()*, or  
1560 *ioctl()*, messages are sent down the STREAM, and *read()*, *readv()*, *getmsg()*, or *getpmsg()* accepts  
1561 data from the STREAM and passes it to a process. Data intended for the device at the  
1562 downstream end of the STREAM is packaged into messages and sent downstream, while data  
1563 and signals from the device are composed into messages by the device driver and sent upstream  
1564 to the STREAM head.

1565 When a STREAMS-based device is opened, a STREAM shall be created that contains the  
1566 STREAM head and the STREAM end (driver). If pipes are STREAMS-based in an  
1567 implementation, when a pipe is created, two STREAMS shall be created, each containing a  
1568 STREAM head. Other modules are added to the STREAM using *ioctl()*. New modules are  
1569 “pushed” onto the STREAM one at a time in last-in, first-out (LIFO) style, as though the  
1570 STREAM was a push-down stack.

### 1571 **Priority**

1572 Message types are classified according to their queuing priority and may be *normal* (non-  
1573 priority), *priority*, or *high-priority* messages. A message belongs to a particular priority band that  
1574 determines its ordering when placed on a queue. Normal messages have a priority band of 0 and  
1575 shall always be placed at the end of the queue following all other messages in the queue. High-  
1576 priority messages are always placed at the head of a queue, but shall be discarded if there is  
1577 already a high-priority message in the queue. Their priority band shall be ignored; they are  
1578 high-priority by virtue of their type. Priority messages have a priority band greater than 0.  
1579 Priority messages are always placed after any messages of the same or higher priority. High-  
1580 priority and priority messages are used to send control and data information outside the normal

1581 flow of control. By convention, high-priority messages shall not be affected by flow control. |  
 1582 Normal and priority messages have separate flow controls. |

### 1583 **Message Parts**

1584 A process may access STREAMS messages that contain a data part, control part, or both. The  
 1585 data part is that information which is transmitted over the communication medium and the  
 1586 control information is used by the local STREAMS modules. The other types of messages are  
 1587 used between modules and are not accessible to processes. Messages containing only a data part  
 1588 are accessible via *putmsg()*, *putpmsg()*, *getmsg()*, *getpmsg()*, *read()*, *readv()*, *write()*, or *writv()*. |  
 1589 Messages containing a control part with or without a data part are accessible via calls to  
 1590 *putmsg()*, *putpmsg()*, *getmsg()*, or *getpmsg()*.

### 1591 **2.6.1 Accessing STREAMS**

1592 A process accesses STREAMS-based files using the standard functions *close()*, *ioctl()*, *getmsg()*,  
 1593 *getpmsg()*, *open()*, *pipe()*, *poll()*, *putmsg()*, *putpmsg()*, *read()*, or *write()*. Refer to the applicable  
 1594 function definitions for general properties and errors.

1595 Calls to *ioctl()* shall perform control functions on the STREAM associated with the file descriptor |  
 1596 *fildev*. The control functions may be performed by the STREAM head, a STREAMS module, or |  
 1597 the STREAMS driver for the STREAM. |

1598 STREAMS modules and drivers can detect errors, sending an error message to the STREAM |  
 1599 head, thus causing subsequent functions to fail and set *errno* to the value specified in the |  
 1600 message. In addition, STREAMS modules and drivers can elect to fail a particular *ioctl()* request |  
 1601 alone by sending a negative acknowledgement message to the STREAM head. This shall cause |  
 1602 just the pending *ioctl()* request to fail and set *errno* to the value specified in the message. |

## 1603 **2.7 XSI Interprocess Communication**

1604 XSI This section describes extensions to support interprocess communication. This functionality is  
 1605 dependent on support of the XSI Extension (and the rest of this section is not further shaded for  
 1606 this option).

1607 The following message passing, semaphore, and shared memory services form an XSI  
 1608 interprocess communication facility. Certain aspects of their operation are common, and are  
 1609 described below.

1610

1611

1612

1613

1614

1615

IPC Functions		
<i>msgctl()</i>	<i>semctl()</i>	<i>shmctl()</i>
<i>msgget()</i>	<i>semget()</i>	<i>shmdt()</i>
<i>msgrcv()</i>	<i>semop()</i>	<i>shmget()</i>
<i>msgsnd()</i>	<i>shmat()</i>	

1616 Another interprocess communication facility is provided by functions in the Realtime Option  
 1617 Group; see Section 2.8 (on page 491).

1618 **2.7.1 IPC General Description**

1619 Each individual shared memory segment, message queue, and semaphore set shall be identified |  
 1620 by a unique positive integer, called, respectively, a shared memory identifier, *shmid*, a |  
 1621 semaphore identifier, *semid*, and a message queue identifier, *msgid*. The identifiers shall be |  
 1622 returned by calls to *shmget()*, *semget()*, and *msgget()*, respectively. |

1623 Associated with each identifier is a data structure which contains data related to the operations  
 1624 which may be or may have been performed; see the Base Definitions volume of  
 1625 IEEE Std 1003.1-200x, <*sys/shm.h*>, <*sys/sem.h*>, and <*sys/msg.h*> for their descriptions.

1626 Each of the data structures contains both ownership information and an **ipc\_perm** structure (see  
 1627 the Base Definitions volume of IEEE Std 1003.1-200x, <*sys/ipc.h*>) which are used in conjunction  
 1628 to determine whether or not read/write (read/alter for semaphores) permissions should be  
 1629 granted to processes using the IPC facilities. The *mode* member of the **ipc\_perm** structure acts as  
 1630 a bit field which determines the permissions.

1631 The values of the bits are given below in octal notation.

1632  
 1633

Bit	Meaning
0400	Read by user.
0200	Write by user.
0040	Read by group.
0020	Write by group.
0004	Read by others.
0002	Write by others.

1634  
 1635  
 1636  
 1637  
 1638  
 1639

1640 The name of the **ipc\_perm** structure is *shm\_perm*, *sem\_perm*, or *msg\_perm*, depending on which  
 1641 service is being used. In each case, read and write/alter permissions shall be granted to a process |  
 1642 if one or more of the following are true ("xxx" is replaced by *shm*, *sem*, or *msg*, as appropriate): |

- 1643 • The process has appropriate privileges.
- 1644 • The effective user ID of the process matches *xxx\_perm.cuid* or *xxx\_perm.uid* in the data  
 1645 structure associated with the IPC identifier, and the appropriate bit of the *user* field in  
 1646 *xxx\_perm.mode* is set.
- 1647 • The effective user ID of the process does not match *xxx\_perm.cuid* or *xxx\_perm.uid* but the  
 1648 effective group ID of the process matches *xxx\_perm.cgid* or *xxx\_perm.gid* in the data structure  
 1649 associated with the IPC identifier, and the appropriate bit of the *group* field in *xxx\_perm.mode*  
 1650 is set.
- 1651 • The effective user ID of the process does not match *xxx\_perm.cuid* or *xxx\_perm.uid* and the  
 1652 effective group ID of the process does not match *xxx\_perm.cgid* or *xxx\_perm.gid* in the data  
 1653 structure associated with the IPC identifier, but the appropriate bit of the *other* field in  
 1654 *xxx\_perm.mode* is set.

1655 Otherwise, the permission shall be denied. |

## 1656 2.8 Realtime

1657 This section defines functions to support the source portability of applications with realtime  
1658 requirements. The presence of many of these functions is dependent on support for  
1659 implementation options described in the text.

1660 The specific functional areas included in this section and their scope include the following. Full  
1661 definitions of these terms can be found in the Base Definitions volume of IEEE Std 1003.1-200x,  
1662 Chapter 3, Definitions.

- 1663 • Semaphores
- 1664 • Process Memory Locking
- 1665 • Memory Mapped Files and Shared Memory Objects
- 1666 • Priority Scheduling
- 1667 • Realtime Signal Extension
- 1668 • Timers
- 1669 • Interprocess Communication
- 1670 • Synchronized Input and Output
- 1671 • Asynchronous Input and Output

1672 All the realtime functions defined in this volume of IEEE Std 1003.1-200x are portable, although  
1673 some of the numeric parameters used by an implementation may have hardware dependencies.

### 1674 2.8.1 Realtime Signals

1675 RTS Realtime signal generation and delivery is dependent on support for the Realtime Signals  
1676 Extension option.

1677 See Section 2.4.2 (on page 479).

### 1678 2.8.2 Asynchronous I/O

1679 AIO The functionality described in this section is dependent on support of the Asynchronous Input  
1680 and Output option (and the rest of this section is not further shaded for this option).

1681 An asynchronous I/O control block structure **aiocb** is used in many asynchronous I/O  
1682 functions. It is defined in the Base Definitions volume of IEEE Std 1003.1-200x, <**aio.h**> and has  
1683 at least the following members:

1684	Member Type	Member Name	Description
1685	<b>int</b>	<i>aio_fildes</i>	File descriptor.
1686	<b>off_t</b>	<i>aio_offset</i>	File offset.
1687	<b>volatile void*</b>	<i>aio_buf</i>	Location of buffer.
1688	<b>size_t</b>	<i>aio_nbytes</i>	Length of transfer.
1689	<b>int</b>	<i>aio_reqprio</i>	Request priority offset.
1690	<b>struct sigevent</b>	<i>aio_sigevent</i>	Signal number and value.
1691	<b>int</b>	<i>aio_lio_opcode</i>	Operation to be performed.

1692 The *aio\_fildes* element is the file descriptor on which the asynchronous operation is performed.

1693 If **O\_APPEND** is not set for the file descriptor *aio\_fildes* and if *aio\_fildes* is associated with a  
1694 device that is capable of seeking, then the requested operation takes place at the absolute  
1695 position in the file as given by *aio\_offset*, as if *lseek()* were called immediately prior to the

1696 operation with an *offset* argument equal to *aio\_offset* and a *whence* argument equal to `SEEK_SET`.  
1697 If `O_APPEND` is set for the file descriptor, or if *aio\_fildes* is associated with a device that is  
1698 incapable of seeking, write operations append to the file in the same order as the calls were  
1699 made, with the following exception: under implementation-defined circumstances, such as  
1700 operation on a multi-processor or when requests of differing priorities are submitted at the same  
1701 time, the ordering restriction may be relaxed. Since there is no way for a strictly conforming  
1702 application to determine whether this relaxation applies, all strictly conforming applications  
1703 which rely on ordering of output shall be written in such a way that they will operate correctly if  
1704 the relaxation applies. After a successful call to enqueue an asynchronous I/O operation, the  
1705 value of the file offset for the file is unspecified. The *aio\_nbytes* and *aio\_buf* elements are the same  
1706 as the *nbyte* and *buf* arguments defined by `read()` and `write()`, respectively.

1707 If `_POSIX_PRIORITIZED_IO` and `_POSIX_PRIORITY_SCHEDULING` are defined, then  
1708 asynchronous I/O is queued in priority order, with the priority of each asynchronous operation  
1709 based on the current scheduling priority of the calling process. The *aio\_reqprio* member can be  
1710 used to lower (but not raise) the asynchronous I/O operation priority and is within the range  
1711 zero through `{AIO_PRIO_DELTA_MAX}`, inclusive. Unless both `_POSIX_PRIORITIZED_IO` and  
1712 `_POSIX_PRIORITY_SCHEDULING` are defined, the order of processing asynchronous I/O  
1713 requests is unspecified. When both `_POSIX_PRIORITIZED_IO` and  
1714 `_POSIX_PRIORITY_SCHEDULING` are defined, the order of processing of requests submitted  
1715 by processes whose schedulers are not `SCHED_FIFO`, `SCHED_RR`, or `SCHED_SPORADIC` is  
1716 unspecified. The priority of an asynchronous request is computed as (process scheduling  
1717 priority) minus *aio\_reqprio*. The priority assigned to each asynchronous I/O request is an  
1718 indication of the desired order of execution of the request relative to other asynchronous I/O  
1719 requests for this file. If `_POSIX_PRIORITIZED_IO` is defined, requests issued with the same  
1720 priority to a character special file are processed by the underlying device in FIFO order; the order  
1721 of processing of requests of the same priority issued to files that are not character special files is  
1722 unspecified. Numerically higher priority values indicate requests of higher priority. The value of  
1723 *aio\_reqprio* has no effect on process scheduling priority. When prioritized asynchronous I/O  
1724 requests to the same file are blocked waiting for a resource required for that I/O operation, the  
1725 higher-priority I/O requests shall be granted the resource before lower-priority I/O requests are  
1726 granted the resource. The relative priority of asynchronous I/O and synchronous I/O is  
1727 implementation-defined. If `_POSIX_PRIORITIZED_IO` is defined, the implementation shall  
1728 define for which files I/O prioritization is supported.

1729 The *aio\_sigevent* determines how the calling process shall be notified upon I/O completion, as  
1730 specified in Section 2.4.1 (on page 478). If *aio\_sigevent.sigev\_notify* is `SIGEV_NONE`, then no  
1731 signal shall be posted upon I/O completion, but the error status for the operation and the return  
1732 status for the operation shall be set appropriately.

1733 The *aio\_lio\_opcode* field is used only by the `lio_listio()` call. The `lio_listio()` call allows multiple  
1734 asynchronous I/O operations to be submitted at a single time. The function takes as an  
1735 argument an array of pointers to **aiocb** structures. Each **aiocb** structure indicates the operation to  
1736 be performed (read or write) via the *aio\_lio\_opcode* field.

1737 The address of the **aiocb** structure is used as a handle for retrieving the error status and return  
1738 status of the asynchronous operation while it is in progress.

1739 The **aiocb** structure and the data buffers associated with the asynchronous I/O operation are  
1740 being used by the system for asynchronous I/O while, and only while, the error status of the  
1741 asynchronous operation is equal to `[EINPROGRESS]`. Applications shall not modify the **aiocb**  
1742 structure while the structure is being used by the system for asynchronous I/O.

1743 The return status of the asynchronous operation is the number of bytes transferred by the I/O  
1744 operation. If the error status is set to indicate an error completion, then the return status is set to

1745 the return value that the corresponding *read()*, *write()*, or *fsync()* call would have returned.  
1746 When the error status is not equal to [EINPROGRESS], the return status shall reflect the return  
1747 status of the corresponding synchronous operation.

## 1748 **2.8.3 Memory Management**

### 1749 *2.8.3.1 Memory Locking*

1750 ML The functionality described in this section is dependent on support of the Process Memory  
1751 Locking option (and the rest of this section is not further shaded for this option).

1752 Range memory locking operations are defined in terms of pages. Implementations may restrict  
1753 the size and alignment of range lockings to be on page-size boundaries. The page size, in bytes,  
1754 is the value of the configurable system variable {PAGESIZE}. If an implementation has no  
1755 restrictions on size or alignment, it may specify a 1-byte page size.

1756 Memory locking guarantees the residence of portions of the address space. It is  
1757 implementation-defined whether locking memory guarantees fixed translation between virtual  
1758 addresses (as seen by the process) and physical addresses. Per-process memory locks are not  
1759 inherited across a *fork()*, and all memory locks owned by a process are unlocked upon *exec* or  
1760 process termination. Unmapping of an address range removes any memory locks established on  
1761 that address range by this process.

### 1762 *2.8.3.2 Memory Mapped Files*

1763 MF The functionality described in this section is dependent on support of the Memory Mapped Files  
1764 option (and the rest of this section is not further shaded for this option).

1765 Range memory mapping operations are defined in terms of pages. Implementations may  
1766 restrict the size and alignment of range mappings to be on page-size boundaries. The page size,  
1767 in bytes, is the value of the configurable system variable {PAGESIZE}. If an implementation has  
1768 no restrictions on size or alignment, it may specify a 1-byte page size.

1769 Memory mapped files provide a mechanism that allows a process to access files by directly  
1770 incorporating file data into its address space. Once a file is mapped into a process address space,  
1771 the data can be manipulated as memory. If more than one process maps a file, its contents are  
1772 shared among them. If the mappings allow shared write access, then data written into the  
1773 memory object through the address space of one process appears in the address spaces of all  
1774 processes that similarly map the same portion of the memory object.

1775 SHM Shared memory objects are named regions of storage that may be independent of the file system  
1776 and can be mapped into the address space of one or more processes to allow them to share the  
1777 associated memory.

1778 SHM An *unlink()* of a file or *shm\_unlink()* of a shared memory object, while causing the removal of the  
1779 name, does not unmap any mappings established for the object. Once the name has been  
1780 removed, the contents of the memory object are preserved as long as it is referenced. The  
1781 memory object remains referenced as long as a process has the memory object open or has some  
1782 area of the memory object mapped.

### 1783 2.8.3.3 *Memory Protection*

1784 MPR MF The functionality described in this section is dependent on support of the Memory Protection  
1785 and Memory Mapped Files option (and the rest of this section is not further shaded for these  
1786 options).

1787 When an object is mapped, various application accesses to the mapped region may result in  
1788 signals. In this context, SIGBUS is used to indicate an error using the mapped object, and  
1789 SIGSEGV is used to indicate a protection violation or misuse of an address:

- 1790 • A mapping may be restricted to disallow some types of access.
- 1791 • Write attempts to memory that was mapped without write access, or any access to memory  
1792 mapped PROT\_NONE, shall result in a SIGSEGV signal.
- 1793 • References to unmapped addresses shall result in a SIGSEGV signal.
- 1794 • Reference to whole pages within the mapping, but beyond the current length of the object,  
1795 shall result in a SIGBUS signal.
- 1796 • The size of the object is unaffected by access beyond the end of the object (even if a SIGBUS is  
1797 not generated).

### 1798 2.8.3.4 *Typed Memory Objects*

1799 TYM The functionality described in this section is dependent on support of the Typed Memory  
1800 Objects option (and the rest of this section is not further shaded for this option).

1801 Implementations may support the Typed Memory Objects option without supporting the  
1802 Memory Mapped Files option or the Shared Memory Objects option. Typed memory objects are  
1803 implementation-configurable named storage pools accessible from one or more processors in a  
1804 system, each via one or more ports, such as backplane buses, LANs, I/O channels, and so on.  
1805 Each valid combination of a storage pool and a port is identified through a name that is defined  
1806 at system configuration time, in an implementation-defined manner; the name may be  
1807 independent of the file system. Using this name, a typed memory object can be opened and  
1808 mapped into process address space. For a given storage pool and port, it is necessary to support  
1809 both dynamic allocation from the pool as well as mapping at an application-supplied offset  
1810 within the pool; when dynamic allocation has been performed, subsequent deallocation must be  
1811 supported. Lastly, accessing typed memory objects from different ports requires a method for  
1812 obtaining the offset and length of contiguous storage of a region of typed memory (dynamically  
1813 allocated or not); this allows typed memory to be shared among processes and/or processors  
1814 while being accessed from the desired port.

## 1815 2.8.4 **Process Scheduling**

1816 PS The functionality described in this section is dependent on support of the Process Scheduling  
1817 option (and the rest of this section is not further shaded for this option).

### 1818 **Scheduling Policies**

1819 The scheduling semantics described in this volume of IEEE Std 1003.1-200x are defined in terms  
1820 of a conceptual model that contains a set of thread lists. No implementation structures are  
1821 necessarily implied by the use of this conceptual model. It is assumed that no time elapses  
1822 during operations described using this model, and therefore no simultaneous operations are  
1823 possible. This model discusses only processor scheduling for runnable threads, but it should be  
1824 noted that greatly enhanced predictability of realtime applications results if the sequencing of  
1825 other resources takes processor scheduling policy into account.

1826 There is, conceptually, one thread list for each priority. A runnable thread will be on the thread |  
 1827 list for that thread's priority. Multiple scheduling policies shall be provided. Each non-empty |  
 1828 thread list is ordered, contains a head as one end of its order, and a tail as the other. The purpose  
 1829 of a scheduling policy is to define the allowable operations on this set of lists (for example,  
 1830 moving threads between and within lists).

1831 Each process shall be controlled by an associated scheduling policy and priority. These  
 1832 parameters may be specified by explicit application execution of the *sched\_setscheduler()* or  
 1833 *sched\_setparam()* functions.

1834 Each thread shall be controlled by an associated scheduling policy and priority. These  
 1835 parameters may be specified by explicit application execution of the *pthread\_setschedparam()*  
 1836 function.

1837 Associated with each policy is a priority range. Each policy definition shall specify the minimum  
 1838 priority range for that policy. The priority ranges for each policy may but need not overlap the  
 1839 priority ranges of other policies.

1840 A conforming implementation shall select the thread that is defined as being at the head of the  
 1841 highest priority non-empty thread list to become a running thread, regardless of its associated  
 1842 policy. This thread is then removed from its thread list.

1843 Four scheduling policies are specifically required. Other implementation-defined scheduling  
 1844 policies may be defined. The following symbols are defined in the Base Definitions volume of  
 1845 IEEE Std 1003.1-200x, <**sched.h**>:

1846 **SCHED\_FIFO** First in, first out (FIFO) scheduling policy.

1847 **SCHED\_RR** Round robin scheduling policy.

1848 ss **SCHED\_SPORADIC** Sporadic server scheduling policy.

1849 **SCHED\_OTHER** Another scheduling policy.

1850 The values of these symbols shall be distinct.

### 1851 **SCHED\_FIFO**

1852 Conforming implementations shall include a scheduling policy called the FIFO scheduling  
 1853 policy.

1854 Threads scheduled under this policy are chosen from a thread list that is ordered by the time its  
 1855 threads have been on the list without being executed; generally, the head of the list is the thread  
 1856 that has been on the list the longest time, and the tail is the thread that has been on the list the  
 1857 shortest time.

1858 Under the **SCHED\_FIFO** policy, the modification of the definitional thread lists is as follows:

- 1859 1. When a running thread becomes a preempted thread, it becomes the head of the thread list  
 1860 for its priority.
- 1861 2. When a blocked thread becomes a runnable thread, it becomes the tail of the thread list for  
 1862 its priority.
- 1863 3. When a running thread calls the *sched\_setscheduler()* function, the process specified in the  
 1864 function call is modified to the specified policy and the priority specified by the *param*  
 1865 argument.
- 1866 4. When a running thread calls the *sched\_setparam()* function, the priority of the process  
 1867 specified in the function call is modified to the priority specified by the *param* argument.

- 1868 5. When a running thread calls the *pthread\_setschedparam()* function, the thread specified in  
 1869 the function call is modified to the specified policy and the priority specified by the *param*  
 1870 argument.
- 1871 6. When a running thread calls the *pthread\_setschedprio()* function, the thread specified in the |  
 1872 function call is modified to the priority specified by the *prio* argument. |
- 1873 7. If a thread whose policy or priority has been modified other than by *pthread\_setschedprio()* |  
 1874 is a running thread or is runnable, it then becomes the tail of the thread list for its new |  
 1875 priority. |
- 1876 8. If a thread whose policy or priority has been modified by *pthread\_setschedprio()* is a |  
 1877 running thread or is runnable, the effect on its position in the thread list depends on the |  
 1878 direction of the modification, as follows: |
- 1879 a. If the priority is raised, the thread becomes the tail of the thread list. |
- 1880 b. If the priority is unchanged, the thread does not change position in the thread list. |
- 1881 c. If the priority is lowered, the thread becomes the head of the thread list. |
- 1882 9. When a running thread issues the *sched\_yield()* function, the thread becomes the tail of the |  
 1883 thread list for its priority. |
- 1884 10. At no other time is the position of a thread with this scheduling policy within the thread  
 1885 lists affected.

1886 For this policy, valid priorities shall be within the range returned by the *sched\_get\_priority\_max()*  
 1887 and *sched\_get\_priority\_min()* functions when SCHED\_FIFO is provided as the parameter.  
 1888 Conforming implementations shall provide a priority range of at least 32 priorities for this  
 1889 policy.

## 1890 SCHED\_RR

1891 Conforming implementations shall include a scheduling policy called the *round robin* scheduling |  
 1892 policy. This policy shall be identical to the SCHED\_FIFO policy with the additional condition |  
 1893 that when the implementation detects that a running thread has been executing as a running |  
 1894 thread for a time period of the length returned by the *sched\_rr\_get\_interval()* function or longer,  
 1895 the thread shall become the tail of its thread list and the head of that thread list shall be removed  
 1896 and made a running thread.

1897 The effect of this policy is to ensure that if there are multiple SCHED\_RR threads at the same  
 1898 priority, one of them does not monopolize the processor. An application should not rely only on  
 1899 the use of SCHED\_RR to ensure application progress among multiple threads if the application  
 1900 includes threads using the SCHED\_FIFO policy at the same or higher priority levels or  
 1901 SCHED\_RR threads at a higher priority level.

1902 A thread under this policy that is preempted and subsequently resumes execution as a running  
 1903 thread completes the unexpired portion of its round robin interval time period.

1904 For this policy, valid priorities shall be within the range returned by the *sched\_get\_priority\_max()*  
 1905 and *sched\_get\_priority\_min()* functions when SCHED\_RR is provided as the parameter.  
 1906 Conforming implementations shall provide a priority range of at least 32 priorities for this  
 1907 policy.

1908 **SCHED\_SPORADIC**

1909 SS|TSP The functionality described in this section is dependent on support of the Process Sporadic  
 1910 Server or Thread Sporadic Server options (and the rest of this section is not further shaded for  
 1911 these options).

1912 If `_POSIX_SPORADIC_SERVER` or `_POSIX_THREAD_SPORADIC_SERVER` is defined, the  
 1913 implementation shall include a scheduling policy identified by the value `SCHED_SPORADIC`.

1914 The sporadic server policy is based primarily on two parameters: the *replenishment period* and the  
 1915 *available execution capacity*. The replenishment period is given by the *sched\_ss\_repl\_period*  
 1916 member of the **sched\_param** structure. The available execution capacity is initialized to the  
 1917 value given by the *sched\_ss\_init\_budget* member of the same parameter. The sporadic server  
 1918 policy is identical to the `SCHED_FIFO` policy with some additional conditions that cause the  
 1919 thread's assigned priority to be switched between the values specified by the *sched\_priority* and  
 1920 *sched\_ss\_low\_priority* members of the **sched\_param** structure.

1921 The priority assigned to a thread using the sporadic server scheduling policy is determined in  
 1922 the following manner: if the available execution capacity is greater than zero and the number of  
 1923 pending replenishment operations is strictly less than *sched\_ss\_max\_repl*, the thread is assigned  
 1924 the priority specified by *sched\_priority*; otherwise, the assigned priority shall be  
 1925 *sched\_ss\_low\_priority*. If the value of *sched\_priority* is less than or equal to the value of  
 1926 *sched\_ss\_low\_priority*, the results are undefined. When active, the thread shall belong to the  
 1927 thread list corresponding to its assigned priority level, according to the mentioned priority  
 1928 assignment. The modification of the available execution capacity and, consequently of the  
 1929 assigned priority, is done as follows:

- 1930 1. When the thread at the head of the *sched\_priority* list becomes a running thread, its  
 1931 execution time shall be limited to at most its available execution capacity, plus the  
 1932 resolution of the execution time clock used for this scheduling policy. This resolution shall  
 1933 be implementation-defined.
- 1934 2. Each time the thread is inserted at the tail of the list associated with *sched\_priority*—  
 1935 because as a blocked thread it became runnable with priority *sched\_priority* or because a  
 1936 replenishment operation was performed—the time at which this operation is done is  
 1937 posted as the *activation\_time*.
- 1938 3. When the running thread with assigned priority equal to *sched\_priority* becomes a  
 1939 preempted thread, it becomes the head of the thread list for its priority, and the execution  
 1940 time consumed is subtracted from the available execution capacity. If the available  
 1941 execution capacity would become negative by this operation, it shall be set to zero.
- 1942 4. When the running thread with assigned priority equal to *sched\_priority* becomes a blocked  
 1943 thread, the execution time consumed is subtracted from the available execution capacity,  
 1944 and a replenishment operation is scheduled, as described in 6 and 7. If the available  
 1945 execution capacity would become negative by this operation, it shall be set to zero.
- 1946 5. When the running thread with assigned priority equal to *sched\_priority* reaches the limit  
 1947 imposed on its execution time, it becomes the tail of the thread list for  
 1948 *sched\_ss\_low\_priority*, the execution time consumed is subtracted from the available  
 1949 execution capacity (which becomes zero), and a replenishment operation is scheduled, as  
 1950 described in 6 and 7.
- 1951 6. Each time a replenishment operation is scheduled, the amount of execution capacity to be  
 1952 replenished, *replenish\_amount*, is set equal to the execution time consumed by the thread  
 1953 since the *activation\_time*. The replenishment is scheduled to occur at *activation\_time* plus  
 1954 *sched\_ss\_repl\_period*. If the scheduled time obtained is before the current time, the

1955 replenishment operation is carried out immediately. Several replenishment operations may  
 1956 be pending at the same time, each of which will be serviced at its respective scheduled  
 1957 time. With the above rules, the number of replenishment operations simultaneously  
 1958 pending for a given thread that is scheduled under the sporadic server policy shall not be  
 1959 greater than *sched\_ss\_max\_repl*.

1960 7. A replenishment operation consists of adding the corresponding *replenish\_amount* to the  
 1961 available execution capacity at the scheduled time. If, as a consequence of this operation,  
 1962 the execution capacity would become larger than *sched\_ss\_initial\_budget*, it shall be  
 1963 rounded down to a value equal to *sched\_ss\_initial\_budget*. Additionally, if the thread was  
 1964 runnable or running, and had assigned priority equal to *sched\_ss\_low\_priority*, then it  
 1965 becomes the tail of the thread list for *sched\_priority*.

1966 Execution time is defined in Section 2.2.2 (on page 464).

1967 For this policy, changing the value of a CPU-time clock via *clock\_settime()* shall have no effect on  
 1968 its behavior.

1969 For this policy, valid priorities shall be within the range returned by the *sched\_get\_priority\_min()*  
 1970 and *sched\_get\_priority\_max()* functions when SCHED\_SPORADIC is provided as the parameter.  
 1971 Conforming implementations shall provide a priority range of at least 32 distinct priorities for  
 1972 this policy.

1973 **SCHED\_OTHER**

1974 Conforming implementations shall include one scheduling policy identified as SCHED\_OTHER  
 1975 (which may execute identically with either the FIFO or round robin scheduling policy). The  
 1976 effect of scheduling threads with the SCHED\_OTHER policy in a system in which other threads  
 1977 ss are executing under SCHED\_FIFO, SCHED\_RR, or SCHED\_SPORADIC is implementation-  
 1978 defined.

1979 This policy is defined to allow strictly conforming applications to be able to indicate in a  
 1980 portable manner that they no longer need a realtime scheduling policy.

1981 For threads executing under this policy, the implementation shall use only priorities within the  
 1982 range returned by the *sched\_get\_priority\_max()* and *sched\_get\_priority\_min()* functions when  
 1983 SCHED\_OTHER is provided as the parameter.

1984 **2.8.5 Clocks and Timers**

1985 TMR The functionality described in this section is dependent on support of the Timers option (and the  
 1986 rest of this section is not further shaded for this option).

1987 The `<time.h>` header defines the types and manifest constants used by the timing facility.

1988 **Time Value Specification Structures**

1989 Many of the timing facility functions accept or return time value specifications. A time value  
 1990 structure **timespec** specifies a single time value and includes at least the following members:

1991

1992

1993

1994

Member Type	Member Name	Description
<b>time_t</b>	<i>tv_sec</i>	Seconds.
<b>long</b>	<i>tv_nsec</i>	Nanoseconds.

1995 The *tv\_nsec* member is only valid if greater than or equal to zero, and less than the number of  
 1996 nanoseconds in a second (1,000 million). The time interval described by this structure is (*tv\_sec* \*  
 1997 10<sup>9</sup> + *tv\_nsec*) nanoseconds.

1998 A time value structure **itimerspec** specifies an initial timer value and a repetition interval for use  
 1999 by the per-process timer functions. This structure includes at least the following members:

2000  
 2001  
 2002  
 2003

Member Type	Member Name	Description
<b>struct timespec</b>	<i>it_interval</i>	Timer period.
<b>struct timespec</b>	<i>it_value</i>	Timer expiration.

2004 If the value described by *it\_value* is non-zero, it indicates the time to or time of the next timer  
 2005 expiration (for relative and absolute timer values, respectively). If the value described by *it\_value*  
 2006 is zero, the timer shall be disarmed.

2007 If the value described by *it\_interval* is non-zero, it specifies an interval which shall be used in  
 2008 reloading the timer when it expires; that is, a periodic timer is specified. If the value described by  
 2009 *it\_interval* is zero, the timer is disarmed after its next expiration; that is, a one-shot timer is  
 2010 specified.

2011 **Timer Event Notification Control Block**

2012 RTS Per-process timers may be created that notify the process of timer expirations by queuing a  
 2013 realtime extended signal. The **sigevent** structure, defined in the Base Definitions volume of  
 2014 IEEE Std 1003.1-200x, <**signal.h**>, is used in creating such a timer. The **sigevent** structure  
 2015 contains the signal number and an application-specific data value which shall be used when  
 2016 notifying the calling process of timer expiration events.

2017 **Manifest Constants**

2018 The following constants are defined in the Base Definitions volume of IEEE Std 1003.1-200x,  
 2019 <**time.h**>:

2020 **CLOCK\_REALTIME** The identifier for the system-wide realtime clock.

2021 **TIMER\_ABSTIME** Flag indicating time is absolute with respect to the clock associated  
 2022 with a timer.

2023 MON **CLOCK\_MONOTONIC** The identifier for the system-wide monotonic clock, which is defined  
 2024 as a clock whose value cannot be set via *clock\_settime()* and which  
 2025 cannot have backward clock jumps. The maximum possible clock  
 2026 jump is implementation-defined.

2027 MON The maximum allowable resolution for **CLOCK\_REALTIME** and **CLOCK\_MONOTONIC** clocks  
 2028 and all time services based on these clocks is represented by **{\_POSIX\_CLOCKRES\_MIN}** and  
 2029 shall be defined as 20ms (1/50 of a second). Implementations may support smaller values of  
 2030 resolution for these clocks to provide finer granularity time bases. The actual resolution  
 2031 supported by an implementation for a specific clock is obtained using the *clock\_getres()* function.  
 2032 If the actual resolution supported for a time service based on one of these clocks differs from the  
 2033 resolution supported for that clock, the implementation shall document this difference.

2034 MON The minimum allowable maximum value for **CLOCK\_REALTIME** and **CLOCK\_MONOTONIC**  
 2035 clocks and all absolute time services based on them is the same as that defined by the ISO C  
 2036 standard for the **time\_t** type. If the maximum value supported by a time service based on one of  
 2037 these clocks differs from the maximum value supported by that clock, the implementation shall  
 2038 document this difference.

2039 **Execution Time Monitoring**

2040 CPT If `_POSIX_CPUTIME` is defined, process CPU-time clocks shall be supported in addition to the  
2041 clocks described in **Manifest Constants** (on page 499).

2042 TCT If `_POSIX_THREAD_CPUTIME` is defined, thread CPU-time clocks shall be supported.

2043 CPT|TCT CPU-time clocks measure execution or CPU time, which is defined in the Base Definitions  
2044 volume of IEEE Std 1003.1-200x, Section 3.117, CPU Time (Execution Time). The mechanism  
2045 used to measure execution time is described in the Base Definitions volume of  
2046 IEEE Std 1003.1-200x, Section 4.9, Measurement of Execution Time.

2047 CPT If `_POSIX_CPUTIME` is defined, the following constant of the type `clockid_t` is defined in  
2048 `<time.h>`:

2049 `CLOCK_PROCESS_CPUTIME_ID`

2050 When this value of the type `clockid_t` is used in a `clock()` or `timer*()` function call, it is  
2051 interpreted as the identifier of the CPU-time clock associated with the process making the  
2052 function call.  
2053

2054 TCT If `_POSIX_THREAD_CPUTIME` is defined, the following constant of the type `clockid_t` is  
2055 defined in `<time.h>`:

2056 `CLOCK_THREAD_CPUTIME_ID`

2057 When this value of the type `clockid_t` is used in a `clock()` or `timer*()` function call, it is  
2058 interpreted as the identifier of the CPU-time clock associated with the thread making the  
2059 function call.

2060 **2.9 Threads**

2061 THR The functionality described in this section is dependent on support of the Threads option (and  
2062 the rest of this section is not further shaded for this option).

2063 This section defines functionality to support multiple flows of control, called *threads*, within a  
2064 process. For the definition of *threads*, see the Base Definitions volume of IEEE Std 1003.1-200x,  
2065 Section 3.393, Thread.

2066 The specific functional areas covered by threads and their scope include:

- 2067 • Thread management: the creation, control, and termination of multiple flows of control in the  
2068 same process under the assumption of a common shared address space
- 2069 • Synchronization primitives optimized for tightly coupled operation of multiple control flows  
2070 in a common, shared address space

2071 **2.9.1 Thread-Safety**

2072 All functions defined by this volume of IEEE Std 1003.1-200x shall be thread-safe, except that the  
2073 following functions<sup>1</sup> need not be thread-safe.

2074 \_\_\_\_\_

2075 1. The functions in the table are not shaded to denote applicable options. Individual reference pages should be consulted.

2076	<i>asctime()</i>	<i>ecvt()</i>	<i>gethostent()</i>	<i>getutxline()</i>	<i>putenv()</i>	
2077	<i>basename()</i>	<i>encrypt()</i>	<i>getlogin()</i>	<i>gmtime()</i>	<i>pututxline()</i>	
2078	<i>catgets()</i>	<i>endgrent()</i>	<i>getnetbyaddr()</i>	<i>hcreate()</i>	<i>rand()</i>	
2079	<i>crypt()</i>	<i>endpwent()</i>	<i>getnetbyname()</i>	<i>hdestroy()</i>	<i>readdir()</i>	
2080	<i>ctime()</i>	<i>endtxent()</i>	<i>getnetent()</i>	<i>hsearch()</i>	<i>setenv()</i>	
2081	<i>dbm_clearerr()</i>	<i>fcvt()</i>	<i>getopt()</i>	<i>inet_ntoa()</i>	<i>setgrent()</i>	
2082	<i>dbm_close()</i>	<i>ftw()</i>	<i>getprotobyname()</i>	<i>l64a()</i>	<i>setkey()</i>	
2083	<i>dbm_delete()</i>	<i>gcvt()</i>	<i>getprotobynumber()</i>	<i>lgamma()</i>	<i>setpwent()</i>	
2084	<i>dbm_error()</i>	<i>getc_unlocked()</i>	<i>getprotoent()</i>	<i>localeconv()</i>	<i>setutxent()</i>	
2085	<i>dbm_fetch()</i>	<i>getchar_unlocked()</i>	<i>getpwent()</i>	<i>localtime()</i>	<i>strerror()</i>	
2086	<i>dbm_firstkey()</i>	<i>getdate()</i>	<i>getpwnam()</i>	<i>lrand48()</i>	<i>strtok()</i>	
2087	<i>dbm_nextkey()</i>	<i>getenv()</i>	<i>getpwuid()</i>	<i>mrand48()</i>	<i>ttyname()</i>	
2088	<i>dbm_open()</i>	<i>getgrent()</i>	<i>getservbyname()</i>	<i>nftw()</i>	<i>unsetenv()</i>	
2089	<i>dbm_store()</i>	<i>getgrgid()</i>	<i>getservbyport()</i>	<i>nl_langinfo()</i>	<i>wcstombs()</i>	
2090	<i>dirname()</i>	<i>getgrnam()</i>	<i>getservent()</i>	<i>ptsname()</i>	<i>wctomb()</i>	
2091	<i>derror()</i>	<i>gethostbyaddr()</i>	<i>getutxent()</i>	<i>putc_unlocked()</i>		
2092	<i>drand48()</i>	<i>gethostbyname()</i>	<i>getutxid()</i>	<i>putchar_unlocked()</i>		

2093 The *ctermid()* and *tmpnam()* functions need not be thread-safe if passed a NULL argument. The  
 2094 *wctomb()* and *wcsrtombs()* functions need not be thread-safe if passed a NULL *ps* argument.

2095 Implementations shall provide internal synchronization as necessary in order to satisfy this  
 2096 requirement.

## 2097 2.9.2 Thread IDs

2098 Although implementations may have thread IDs that are unique in a system, applications  
 2099 should only assume that thread IDs are usable and unique within a single process. The effect of  
 2100 calling any of the functions defined in this volume of IEEE Std 1003.1-200x and passing as an  
 2101 argument the thread ID of a thread from another process is unspecified. A conforming  
 2102 implementation is free to reuse a thread ID after the thread terminates if it was created with the  
 2103 *detachstate* attribute set to *PTHREAD\_CREATE\_DETACHED* or if *pthread\_detach()* or  
 2104 *pthread\_join()* has been called for that thread. If a thread is detached, its thread ID is invalid for  
 2105 use as an argument in a call to *pthread\_detach()* or *pthread\_join()*.

## 2106 2.9.3 Thread Mutexes

2107 A thread that has blocked shall not prevent any unblocked thread that is eligible to use the same  
 2108 processing resources from eventually making forward progress in its execution. Eligibility for  
 2109 processing resources is determined by the scheduling policy.

2110 A thread shall become the owner of a mutex, *m*, when one of the following occurs: |

- 2111 • It returns successfully from *pthread\_mutex\_lock()* with *m* as the *mutex* argument.
- 2112 • It returns successfully from *pthread\_mutex\_trylock()* with *m* as the *mutex* argument.
- 2113 TMO • It returns successfully from *pthread\_mutex\_timedwait()* with *m* as the *mutex* argument.
- 2114 • It returns (successfully or not) from *pthread\_cond\_wait()* with *m* as the *mutex* argument  
 2115 (except as explicitly indicated otherwise for certain errors).
- 2116 • It returns (successfully or not) from *pthread\_cond\_timedwait()* with *m* as the *mutex* argument  
 2117 (except as explicitly indicated otherwise for certain errors).

2118 The thread shall remain the owner of *m* until one of the following occurs: |

- 2119           • It executes `pthread_mutex_unlock()` with `m` as the *mutex* argument
- 2120           • It blocks in a call to `pthread_cond_wait()` with `m` as the *mutex* argument.
- 2121           • It blocks in a call to `pthread_cond_timedwait()` with `m` as the *mutex* argument.
- 2122           The implementation shall behave as if at all times there is at most one owner of any mutex. |
- 2123           A thread that becomes the owner of a mutex is said to have *acquired* the mutex and the mutex is
- 2124           said to have become *locked*; when a thread gives up ownership of a mutex it is said to have
- 2125           *released* the mutex and the mutex is said to have become *unlocked*.
- 2126 **2.9.4 Thread Scheduling**
- 2127 TPS       The functionality described in this section is dependent on support of the Thread Execution
- 2128           Scheduling option (and the rest of this section is not further shaded for this option).
- 2129           **Thread Scheduling Attributes**
- 2130           In support of the scheduling function, threads have attributes which are accessed through the
- 2131           **pthread\_attr\_t** thread creation attributes object.
- 2132           The *contentionscope* attribute defines the scheduling contention scope of the thread to be either
- 2133           PTHREAD\_SCOPE\_PROCESS or PTHREAD\_SCOPE\_SYSTEM.
- 2134           The *inheritsched* attribute specifies whether a newly created thread is to inherit the scheduling
- 2135           attributes of the creating thread or to have its scheduling values set according to the other
- 2136           scheduling attributes in the **pthread\_attr\_t** object.
- 2137           The *schedpolicy* attribute defines the scheduling policy for the thread. The *schedparam* attribute
- 2138           defines the scheduling parameters for the thread. The interaction of threads having different
- 2139           policies within a process is described as part of the definition of those policies.
- 2140           If the Thread Execution Scheduling option is defined, and the *schedpolicy* attribute specifies one
- 2141           of the priority-based policies defined under this option, the *schedparam* attribute contains the
- 2142           scheduling priority of the thread. A conforming implementation ensures that the priority value
- 2143           in *schedparam* is in the range associated with the scheduling policy when the thread attributes
- 2144           object is used to create a thread, or when the scheduling attributes of a thread are dynamically
- 2145           modified. The meaning of the priority value in *schedparam* is the same as that of *priority*.
- 2146 TSP       If `_POSIX_THREAD_SPORADIC_SERVER` is defined, the *schedparam* attribute supports four
- 2147           new members that are used for the sporadic server scheduling policy. These members are
- 2148           *sched\_ss\_low\_priority*, *sched\_ss\_repl\_period*, *sched\_ss\_init\_budget*, and *sched\_ss\_max\_repl*. The
- 2149           meaning of these attributes is the same as in the definitions that appear under Section 2.8.4 (on
- 2150           page 494).
- 2151           When a process is created, its single thread has a scheduling policy and associated attributes
- 2152           equal to the process' policy and attributes. The default scheduling contention scope value is
- 2153           implementation-defined. The default values of other scheduling attributes are implementation-
- 2154           defined.

**2155 Thread Scheduling Contention Scope**

2156 The scheduling contention scope of a thread defines the set of threads with which the thread  
2157 competes for use of the processing resources. The scheduling operation selects at most one  
2158 thread to execute on each processor at any point in time and the thread's scheduling attributes  
2159 (for example, *priority*), whether under process scheduling contention scope or system scheduling  
2160 contention scope, are the parameters used to determine the scheduling decision.

2161 The scheduling contention scope, in the context of scheduling a mixed scope environment,  
2162 affects threads as follows:

- 2163 • A thread created with `PTHREAD_SCOPE_SYSTEM` scheduling contention scope contends  
2164 for resources with all other threads in the same scheduling allocation domain relative to their  
2165 system scheduling attributes. The system scheduling attributes of a thread created with  
2166 `PTHREAD_SCOPE_SYSTEM` scheduling contention scope are the scheduling attributes with  
2167 which the thread was created. The system scheduling attributes of a thread created with  
2168 `PTHREAD_SCOPE_PROCESS` scheduling contention scope are the implementation-defined  
2169 mapping into system attribute space of the scheduling attributes with which the thread was  
2170 created.
- 2171 • Threads created with `PTHREAD_SCOPE_PROCESS` scheduling contention scope contend  
2172 directly with other threads within their process that were created with  
2173 `PTHREAD_SCOPE_PROCESS` scheduling contention scope. The contention is resolved  
2174 based on the threads' scheduling attributes and policies. It is unspecified how such threads  
2175 are scheduled relative to threads in other processes or threads with  
2176 `PTHREAD_SCOPE_SYSTEM` scheduling contention scope.
- 2177 • Conforming implementations shall support the `PTHREAD_SCOPE_PROCESS` scheduling  
2178 contention scope, the `PTHREAD_SCOPE_SYSTEM` scheduling contention scope, or both.

**2179 Scheduling Allocation Domain**

2180 Implementations shall support scheduling allocation domains containing one or more  
2181 processors. It should be noted that the presence of multiple processors does not automatically  
2182 indicate a scheduling allocation domain size greater than one. Conforming implementations on  
2183 multi-processors may map all or any subset of the CPUs to one or multiple scheduling allocation  
2184 domains, and could define these scheduling allocation domains on a per-thread, per-process, or  
2185 per-system basis, depending on the types of applications intended to be supported by the  
2186 implementation. The scheduling allocation domain is independent of scheduling contention  
2187 scope, as the scheduling contention scope merely defines the set of threads with which a thread  
2188 contends for processor resources, while scheduling allocation domain defines the set of  
2189 processors for which it contends. The semantics of how this contention is resolved among  
2190 threads for processors is determined by the scheduling policies of the threads.

2191 The choice of scheduling allocation domain size and the level of application control over  
2192 scheduling allocation domains is implementation-defined. Conforming implementations may  
2193 change the size of scheduling allocation domains and the binding of threads to scheduling  
2194 allocation domains at any time.

2195 For application threads with scheduling allocation domains of size equal to one, the scheduling  
2196 rules defined for `SCHED_FIFO` and `SCHED_RR` shall be used; see **Scheduling Policies** (on page  
2197 494). All threads with system scheduling contention scope, regardless of the processes in which  
2198 they reside, compete for the processor according to their priorities. Threads with process  
2199 scheduling contention scope compete only with other threads with process scheduling  
2200 contention scope within their process.

2201 For application threads with scheduling allocation domains of size greater than one, the rules  
 2202 TSP defined for SCHED\_FIFO, SCHED\_RR, and SCHED\_SPORADIC shall be used in an  
 2203 implementation-defined manner. Each thread with system scheduling contention scope  
 2204 competes for the processors in its scheduling allocation domain in an implementation-defined  
 2205 manner according to its priority. Threads with process scheduling contention scope are  
 2206 scheduled relative to other threads within the same scheduling contention scope in the process.

2207 TSP If \_POSIX\_THREAD\_SPORADIC\_SERVER is defined, the rules defined for SCHED\_SPORADIC  
 2208 in **Scheduling Policies** (on page 494) shall be used in an implementation-defined manner for  
 2209 application threads whose scheduling allocation domain size is greater than one.

## 2210 Scheduling Documentation

2211 If \_POSIX\_PRIORITY\_SCHEDULING is defined, then any scheduling policies beyond  
 2212 TSP SCHED\_OTHER, SCHED\_FIFO, SCHED\_RR, and SCHED\_SPORADIC, as well as the effects of  
 2213 the scheduling policies indicated by these other values, and the attributes required in order to  
 2214 support such a policy, are implementation-defined. Furthermore, the implementation shall  
 2215 document the effect of all processor scheduling allocation domain values supported for these  
 2216 policies.

## 2217 2.9.5 Thread Cancellation

2218 The thread cancellation mechanism allows a thread to terminate the execution of any other  
 2219 thread in the process in a controlled manner. The target thread (that is, the one that is being  
 2220 canceled) is allowed to hold cancellation requests pending in a number of ways and to perform  
 2221 application-specific cleanup processing when the notice of cancellation is acted upon.

2222 Cancellation is controlled by the cancellation control functions. Each thread maintains its own  
 2223 cancelability state. Cancellation may only occur at cancellation points or when the thread is  
 2224 asynchronously cancelable.

2225 The thread cancellation mechanism described in this section depends upon programs having set  
 2226 *deferred cancelability* state, which is specified as the default. Applications shall also carefully  
 2227 follow static lexical scoping rules in their execution behavior. For example, use of *setjmp()*,  
 2228 *return*, *goto*, and so on, to leave user-defined cancellation scopes without doing the necessary  
 2229 scope pop operation results in undefined behavior.

2230 Use of asynchronous cancelability while holding resources which potentially need to be released  
 2231 may result in resource loss. Similarly, cancellation scopes may only be safely manipulated  
 2232 (pushed and popped) when the thread is in the *deferred* or *disabled* cancelability states.

### 2233 2.9.5.1 Cancelability States

2234 The cancelability state of a thread determines the action taken upon receipt of a cancellation  
 2235 request. The thread may control cancellation in a number of ways.

2236 Each thread maintains its own cancelability state, which may be encoded in two bits:

- 2237 1. Cancelability-Enable: When cancelability is PTHREAD\_CANCEL\_DISABLE (as defined in  
 2238 the Base Definitions volume of IEEE Std 1003.1-200x, <pthread.h>), cancellation requests  
 2239 against the target thread are held pending. By default, cancelability is set to  
 2240 PTHREAD\_CANCEL\_ENABLE (as defined in <pthread.h>).
- 2241 2. Cancelability Type: When cancelability is enabled and the cancelability type is  
 2242 PTHREAD\_CANCEL\_ASYNCHRONOUS (as defined in <pthread.h>), new or pending  
 2243 cancellation requests may be acted upon at any time. When cancelability is enabled and the  
 2244 cancelability type is PTHREAD\_CANCEL\_DEFERRED (as defined in <pthread.h>),

2245 cancellation requests are held pending until a cancellation point (see below) is reached. If  
 2246 cancelability is disabled, the setting of the cancelability type has no immediate effect as all  
 2247 cancellation requests are held pending; however, once cancelability is enabled again the  
 2248 new type is in effect. The cancelability type is PTHREAD\_CANCEL\_DEFERRED in all  
 2249 newly created threads including the thread in which *main()* was first invoked.

### 2250 2.9.5.2 Cancellation Points

2251 Cancellation points shall occur when a thread is executing the following functions: |

2252	<i>accept()</i>	<i>mq_timedsend()</i>	<i>putpmsg()</i>	<i>sigsuspend()</i>
2253	<i>aio_suspend()</i>	<i>msgrcv()</i>	<i>pwrite()</i>	<i>sigtimedwait()</i>
2254	<i>clock_nanosleep()</i>	<i>msgsnd()</i>	<i>read()</i>	<i>sigwait()</i>
2255	<i>close()</i>	<i>msync()</i>	<i>readv()</i>	<i>sigwaitinfo()</i>
2256	<i>connect()</i>	<i>nanosleep()</i>	<i>recv()</i>	<i>sleep()</i>
2257	<i>creat()</i>	<i>open()</i>	<i>recvfrom()</i>	<i>system()</i>
2258	<i>fcntl()</i> <sup>2</sup>	<i>pause()</i>	<i>recvmsg()</i>	<i>tcdrain()</i>
2259	<i>fsync()</i>	<i>poll()</i>	<i>select()</i>	<i>usleep()</i>
2260	<i>getmsg()</i>	<i>pread()</i>	<i>sem_timedwait()</i>	<i>wait()</i>
2261	<i>getpmsg()</i>	<i>pthread_cond_timedwait()</i>	<i>sem_wait()</i>	<i>waitid()</i>
2262	<i>lockf()</i>	<i>pthread_cond_wait()</i>	<i>send()</i>	<i>waitpid()</i>
2263	<i>mq_receive()</i>	<i>pthread_join()</i>	<i>sendmsg()</i>	<i>write()</i>
2264	<i>mq_send()</i>	<i>pthread_testcancel()</i>	<i>sendto()</i>	<i>writev()</i>
2265	<i>mq_timedreceive()</i>	<i>putmsg()</i>	<i>sigpause()</i>	

2266 \_\_\_\_\_

2267 2. When the *cmd* argument is F\_SETLKW.

2268 A cancelation point may also occur when a thread is executing the following functions:

2269	<i>catclose()</i>	<i>ftell()</i>	<i>getwc()</i>	<i>pthread_rwlock_wrlock()</i>
2270	<i>catgets()</i>	<i>ftello()</i>	<i>getwchar()</i>	<i>putc()</i>
2271	<i>catopen()</i>	<i>ftw()</i>	<i>getwd()</i>	<i>putc_unlocked()</i>
2272	<i>closedir()</i>	<i>fwprintf()</i>	<i>glob()</i>	<i>putchar()</i>
2273	<i>closelog()</i>	<i>fwrite()</i>	<i>iconv_close()</i>	<i>putchar_unlocked()</i>
2274	<i>ctermid()</i>	<i>fwscanf()</i>	<i>iconv_open()</i>	<i>puts()</i>
2275	<i>dbm_close()</i>	<i>getc()</i>	<i>ioctl()</i>	<i>pututxline()</i>
2276	<i>dbm_delete()</i>	<i>getc_unlocked()</i>	<i>lseek()</i>	<i>putwc()</i>
2277	<i>dbm_fetch()</i>	<i>getchar()</i>	<i>mkstemp()</i>	<i>putwchar()</i>
2278	<i>dbm_nextkey()</i>	<i>getchar_unlocked()</i>	<i>nftw()</i>	<i>readdir()</i>
2279	<i>dbm_open()</i>	<i>getcwd()</i>	<i>opendir()</i>	<i>readdir_r()</i>
2280	<i>dbm_store()</i>	<i>getdate()</i>	<i>openlog()</i>	<i>remove()</i>
2281	<i>dlclose()</i>	<i>getgrent()</i>	<i>pclose()</i>	<i>rename()</i>
2282	<i>dlopen()</i>	<i>getgrgid()</i>	<i>perror()</i>	<i>rewind()</i>
2283	<i>endgrent()</i>	<i>getgrgid_r()</i>	<i>popen()</i>	<i>rewinddir()</i>
2284	<i>endhostent()</i>	<i>getgrnam()</i>	<i>posix_fadvise()</i>	<i>scanf()</i>
2285	<i>endnetent()</i>	<i>getgrnam_r()</i>	<i>posix_fallocate()</i>	<i>seekdir()</i>
2286	<i>endprotoent()</i>	<i>gethostbyaddr()</i>	<i>posix_madvise()</i>	<i>semop()</i>
2287	<i>endpwent()</i>	<i>gethostbyname()</i>	<i>posix_spawn()</i>	<i>setgrent()</i>
2288	<i>endservent()</i>	<i>gethostent()</i>	<i>posix_spawnnp()</i>	<i>sethostent()</i>
2289	<i>endutxent()</i>	<i>gethostname()</i>	<i>posix_trace_clear()</i>	<i>setnetent()</i>
2290	<i>fclose()</i>	<i>getlogin()</i>	<i>posix_trace_close()</i>	<i>setprotoent()</i>
2291	<i>fcntl()</i> <sup>3</sup>	<i>getlogin_r()</i>	<i>posix_trace_create()</i>	<i>setpwent()</i>
2292	<i>fflush()</i>	<i>getnetbyaddr()</i>	<i>posix_trace_create_withlog()</i>	<i>setservent()</i>
2293	<i>fgetc()</i>	<i>getnetbyname()</i>	<i>posix_trace_eventtypelist_getnext_id()</i>	<i>setutxent()</i>
2294	<i>fgetpos()</i>	<i>getnetent()</i>	<i>posix_trace_eventtypelist_rewind()</i>	<i>strerror()</i>
2295	<i>fgets()</i>	<i>getprotobyname()</i>	<i>posix_trace_flush()</i>	<i>syslog()</i>
2296	<i>fgetwc()</i>	<i>getprotobynumber()</i>	<i>posix_trace_get_attr()</i>	<i>tmpfile()</i>
2297	<i>fgetws()</i>	<i>getprotoent()</i>	<i>posix_trace_get_filter()</i>	<i>tmpnam()</i>
2298	<i>fopen()</i>	<i>getpwent()</i>	<i>posix_trace_get_status()</i>	<i>ttyname()</i>
2299	<i>fprintf()</i>	<i>getpwnam()</i>	<i>posix_trace_getnext_event()</i>	<i>ttyname_r()</i>
2300	<i>fputc()</i>	<i>getpwnam_r()</i>	<i>posix_trace_open()</i>	<i>ungetc()</i>
2301	<i>fputs()</i>	<i>getpwuid()</i>	<i>posix_trace_rewind()</i>	<i>ungetwc()</i>
2302	<i>fputwc()</i>	<i>getpwuid_r()</i>	<i>posix_trace_set_filter()</i>	<i>unlink()</i>
2303	<i>fputws()</i>	<i>gets()</i>	<i>posix_trace_shutdown()</i>	<i>vfprintf()</i>
2304	<i>fread()</i>	<i>getservbyname()</i>	<i>posix_trace_timedgetnext_event()</i>	<i>vwprintf()</i>
2305	<i>freopen()</i>	<i>getservbyport()</i>	<i>posix_typed_mem_open()</i>	<i>vprintf()</i>
2306	<i>fscanf()</i>	<i>getservent()</i>	<i>printf()</i>	<i>vwprintf()</i>
2307	<i>fseek()</i>	<i>getutxent()</i>	<i>pthread_rwlock_rdlock()</i>	<i>wprintf()</i>
2308	<i>fseeko()</i>	<i>getutxid()</i>	<i>pthread_rwlock_timedrdlock()</i>	<i>wscanf()</i>
2309	<i>fsetpos()</i>	<i>getutxline()</i>	<i>pthread_rwlock_timedwrlock()</i>	

2310 An implementation shall not introduce cancelation points into any other functions specified in  
2311 this volume of IEEE Std 1003.1-200x.

2312 \_\_\_\_\_  
2313 3. For any value of the *cmd* argument.

2314 The side effects of acting upon a cancelation request while suspended during a call of a function  
2315 are the same as the side effects that may be seen in a single-threaded program when a call to a  
2316 function is interrupted by a signal and the given function returns [EINTR]. Any such side effects  
2317 occur before any cancelation cleanup handlers are called.

2318 Whenever a thread has cancelability enabled and a cancelation request has been made with that  
2319 thread as the target, and the thread then calls any function that is a cancelation point (such as  
2320 *pthread\_testcancel()* or *read()*), the cancelation request shall be acted upon before the function  
2321 returns. If a thread has cancelability enabled and a cancelation request is made with the thread  
2322 as a target while the thread is suspended at a cancelation point, the thread shall be awakened  
2323 and the cancelation request shall be acted upon. However, if the thread is suspended at a  
2324 cancelation point and the event for which it is waiting occurs before the cancelation request is  
2325 acted upon, it is unspecified whether the cancelation request is acted upon or whether the  
2326 cancelation request remains pending and the thread resumes normal execution.

### 2327 2.9.5.3 Thread Cancelation Cleanup Handlers

2328 Each thread maintains a list of cancelation cleanup handlers. The programmer uses the  
2329 *pthread\_cleanup\_push()* and *pthread\_cleanup\_pop()* functions to place routines on and remove  
2330 routines from this list.

2331 When a cancelation request is acted upon, the routines in the list are invoked one by one in LIFO  
2332 sequence; that is, the last routine pushed onto the list (Last In) is the first to be invoked (First  
2333 Out). The thread invokes the cancelation cleanup handler with cancelation disabled until the last  
2334 cancelation cleanup handler returns. When the cancelation cleanup handler for a scope is  
2335 invoked, the storage for that scope remains valid. If the last cancelation cleanup handler returns,  
2336 thread execution is terminated and a status of `PTHREAD_CANCELED` is made available to any  
2337 threads joining with the target. The symbolic constant `PTHREAD_CANCELED` expands to a  
2338 constant expression of type `(void *)` whose value matches no pointer to an object in memory nor  
2339 the value `NULL`.

2340 The cancelation cleanup handlers are also invoked when the thread calls *pthread\_exit()*.

2341 A side effect of acting upon a cancelation request while in a condition variable wait is that the  
2342 mutex is re-acquired before calling the first cancelation cleanup handler. In addition, the thread  
2343 is no longer considered to be waiting for the condition and the thread shall not have consumed  
2344 any pending condition signals on the condition.

2345 A cancelation cleanup handler cannot exit via *longjmp()* or *siglongjmp()*.

### 2346 2.9.5.4 Async-Cancel Safety

2347 The *pthread\_cancel()*, *pthread\_setcancelstate()*, and *pthread\_setcanceltype()* functions are defined to  
2348 be async-cancel safe.

2349 No other functions in this volume of IEEE Std 1003.1-200x are required to be async-cancel-safe.

**2350 2.9.6 Thread Read-Write Locks**

2351 Multiple readers, single writer (read-write) locks allow many threads to have simultaneous  
2352 read-only access to data while allowing only one thread to have exclusive write access at any  
2353 given time. They are typically used to protect data that is read more frequently than it is  
2354 changed.

2355 One or more readers acquire read access to the resource by performing a read lock operation on  
2356 the associated read-write lock. A writer acquires exclusive write access by performing a write  
2357 lock operation. Basically, all readers exclude any writers and a writer excludes all readers and  
2358 any other writers.

2359 A thread that has blocked on a read-write lock (for example, has not yet returned from a  
2360 *pthread\_rwlock\_rdlock()* or *pthread\_rwlock\_wrlock()* call) shall not prevent any unblocked thread  
2361 that is eligible to use the same processing resources from eventually making forward progress in  
2362 its execution. Eligibility for processing resources shall be determined by the scheduling policy.

2363 Read-write locks can be used to synchronize threads in the current process and other processes if  
2364 they are allocated in memory that is writable and shared among the cooperating processes and  
2365 have been initialized for this behavior.

**2366 2.9.7 Thread Interactions with Regular File Operations**

2367 All of the functions *chmod()*, *close()*, *fchmod()*, *fcntl()*, *fstat()*, *ftruncate()*, *lseek()*, *open()*, *read()*,  
2368 *readlink()*, *stat()*, *symlink()*, and *write()* shall be atomic with respect to each other in the effects  
2369 specified in IEEE Std 1003.1-200x when they operate on regular files. If two threads each call one  
2370 of these functions, each call shall either see all of the specified effects of the other call, or none of  
2371 them.

**2372 2.10 Sockets**

2373 A socket is an endpoint for communication using the facilities described in this section. A socket  
2374 is created with a specific socket type, described in Section 2.10.6 (on page 509), and is associated  
2375 with a specific protocol, detailed in Section 2.10.3 (on page 509). A socket is accessed via a file  
2376 descriptor obtained when the socket is created. |

**2377 2.10.1 Address Families** |

2378 All network protocols are associated with a specific address family. An address family provides |  
2379 basic services to the protocol implementation to allow it to function within a specific network |  
2380 environment. These services may include packet fragmentation and reassembly, routing, |  
2381 addressing, and basic transport. An address family is normally comprised of a number of |  
2382 protocols, one per socket type. Each protocol is characterized by an abstract socket type. It is not |  
2383 required that an address family support all socket types. An address family may contain |  
2384 multiple protocols supporting the same socket abstraction. |

2385 Section 2.10.17 (on page 516), Section 2.10.19 (on page 517), and Section 2.10.20 (on page 517),  
2386 respectively, describe the use of sockets for local UNIX connections, for Internet protocols based  
2387 on IPv4, and for Internet protocols based on IPv6. |

**2388 2.10.2 Addressing**

2389 An address family defines the format of a socket address. All network addresses are described  
2390 using a general structure, called a **sockaddr**, as defined in the Base Definitions volume of  
2391 IEEE Std 1003.1-200x, `<sys/socket.h>`. However, each address family imposes finer and more  
2392 specific structure, generally defining a structure with fields specific to the address family. The  
2393 field *sa\_family* in the **sockaddr** structure contains the address family identifier, specifying the  
2394 format of the *sa\_data* area. The size of the *sa\_data* area is unspecified.

**2395 2.10.3 Protocols**

2396 A protocol supports one of the socket abstractions detailed in Section 2.10.6. Selecting a protocol  
2397 involves specifying the address family, socket type, and protocol number to the *socket()*  
2398 function. Certain semantics of the basic socket abstractions are protocol-specific. All protocols  
2399 are expected to support the basic model for their particular socket type, but may, in addition,  
2400 provide non-standard facilities or extensions to a mechanism.

**2401 2.10.4 Routing**

2402 Sockets provides packet routing facilities. A routing information database is maintained, which  
2403 is used in selecting the appropriate network interface when transmitting packets.

**2404 2.10.5 Interfaces**

2405 Each network interface in a system corresponds to a path through which messages can be sent  
2406 and received. A network interface usually has a hardware device associated with it, though  
2407 certain interfaces such as the loopback interface, do not.

**2408 2.10.6 Socket Types**

2409 A socket is created with a specific type, which defines the communication semantics and which  
2410 RS allows the selection of an appropriate communication protocol. Four types are defined:  
2411 `SOCK_RAW`, `SOCK_STREAM`, `SOCK_SEQPACKET`, and `SOCK_DGRAM`. Implementations  
2412 may specify additional socket types.

2413 The `SOCK_STREAM` socket type provides reliable, sequenced, full-duplex octet streams  
2414 between the socket and a peer to which the socket is connected. A socket of type  
2415 `SOCK_STREAM` must be in a connected state before any data may be sent or received. Record  
2416 boundaries are not maintained; data sent on a stream socket using output operations of one size  
2417 may be received using input operations of smaller or larger sizes without loss of data. Data may  
2418 be buffered; successful return from an output function does not imply that the data has been  
2419 delivered to the peer or even transmitted from the local system. If data cannot be successfully  
2420 transmitted within a given time then the connection is considered broken, and subsequent  
2421 operations shall fail. A `SIGPIPE` signal is raised if a thread sends on a broken stream (one that is  
2422 no longer connected). Support for an out-of-band data transmission facility is protocol-specific.

2423 The `SOCK_SEQPACKET` socket type is similar to the `SOCK_STREAM` type, and is also  
2424 connection-oriented. The only difference between these types is that record boundaries are  
2425 maintained using the `SOCK_SEQPACKET` type. A record can be sent using one or more output  
2426 operations and received using one or more input operations, but a single operation never  
2427 transfers parts of more than one record. Record boundaries are visible to the receiver via the  
2428 `MSG_EOR` flag in the received message flags returned by the *recvmsg()* function. It is protocol-  
2429 specific whether a maximum record size is imposed.

2430 The `SOCK_DGRAM` socket type supports connectionless data transfer which is not necessarily  
2431 acknowledged or reliable. Datagrams may be sent to the address specified (possibly multicast or

2432 broadcast) in each output operation, and incoming datagrams may be received from multiple |  
2433 sources. The source address of each datagram is available when receiving the datagram. An |  
2434 application may also pre-specify a peer address, in which case calls to output functions shall |  
2435 send to the pre-specified peer. If a peer has been specified, only datagrams from that peer shall |  
2436 be received. A datagram must be sent in a single output operation, and must be received in a |  
2437 single input operation. The maximum size of a datagram is protocol-specific; with some |  
2438 protocols, the limit is implementation-defined. Output datagrams may be buffered within the |  
2439 system; thus, a successful return from an output function does not guarantee that a datagram is |  
2440 actually sent or received. However, implementations should attempt to detect any errors |  
2441 possible before the return of an output function, reporting any error by an unsuccessful return |  
2442 value.

2443 RS The SOCK\_RAW socket type is similar to the SOCK\_DGRAM type. It differs in that it is |  
2444 normally used with communication providers that underlie those used for the other socket |  
2445 types. For this reason, the creation of a socket with type SOCK\_RAW shall require appropriate |  
2446 privilege. The format of datagrams sent and received with this socket type generally include |  
2447 specific protocol headers, and the formats are protocol-specific and implementation-defined.

### 2448 **2.10.7 Socket I/O Mode**

2449 The I/O mode of a socket is described by the O\_NONBLOCK file status flag which pertains to |  
2450 the open file description for the socket. This flag is initially off when a socket is created, but may |  
2451 be set and cleared by the use of the F\_SETFL command of the *fcntl()* function.

2452 When the O\_NONBLOCK flag is set, functions that would normally block until they are |  
2453 complete shall either return immediately with an error, or shall complete asynchronously to the |  
2454 execution of the calling process. Data transfer operations (the *read()*, *write()*, *send()*, and *recv()* |  
2455 functions) shall complete immediately, transfer only as much as is available, and then return |  
2456 without blocking, or return an error indicating that no transfer could be made without blocking. |  
2457 The *connect()* function initiates a connection and shall return without blocking when |  
2458 O\_NONBLOCK is set; it shall return the error [EINPROGRESS] to indicate that the connection |  
2459 was initiated successfully, but that it has not yet completed.

### 2460 **2.10.8 Socket Owner**

2461 The owner of a socket is unset when a socket is created. The owner may be set to a process ID or |  
2462 process group ID using the F\_SETOWN command of the *fcntl()* function.

### 2463 **2.10.9 Socket Queue Limits**

2464 The transmit and receive queue sizes for a socket are set when the socket is created. The default |  
2465 sizes used are both protocol-specific and implementation-defined. The sizes may be changed |  
2466 using the *setsockopt()* function.

### 2467 **2.10.10 Pending Error**

2468 Errors may occur asynchronously, and be reported to the socket in response to input from the |  
2469 network protocol. The socket stores the pending error to be reported to a user of the socket at the |  
2470 next opportunity. The error is returned in response to a subsequent *send()*, *recv()*, or *getsockopt()* |  
2471 operation on the socket, and the pending error is then cleared.

**2.10.11 Socket Receive Queue**

A socket has a receive queue that buffers data when it is received by the system until it is removed by a receive call. Depending on the type of the socket and the communication provider, the receive queue may also contain ancillary data such as the addressing and other protocol data associated with the normal data in the queue, and may contain out-of-band or expedited data. The limit on the queue size includes any normal, out-of-band data, datagram source addresses, and ancillary data in the queue. The description in this section applies to all sockets, even though some elements cannot be present in some instances.

The contents of a receive buffer are logically structured as a series of data segments with associated ancillary data and other information. A data segment may contain normal data or out-of-band data, but never both. A data segment may complete a record if the protocol supports records (always true for types `SOCK_SEQPACKET` and `SOCK_DGRAM`). A record may be stored as more than one segment; the complete record might never be present in the receive buffer at one time, as a portion might already have been returned to the application, and another portion might not yet have been received from the communications provider. A data segment may contain ancillary protocol data, which is logically associated with the segment. Ancillary data is received as if it were queued along with the first normal data octet in the segment (if any). A segment may contain ancillary data only, with no normal or out-of-band data. For the purposes of this section, a datagram is considered to be a data segment that terminates a record, and that includes a source address as a special type of ancillary data. Data segments are placed into the queue as data is delivered to the socket by the protocol. Normal data segments are placed at the end of the queue as they are delivered. If a new segment contains the same type of data as the preceding segment and includes no ancillary data, and if the preceding segment does not terminate a record, the segments are logically merged into a single segment.

The receive queue is logically terminated if an end-of-file indication has been received or a connection has been terminated. A segment shall be considered to be terminated if another segment follows it in the queue, if the segment completes a record, or if an end-of-file or other connection termination has been reported. The last segment in the receive queue shall also be considered to be terminated while the socket has a pending error to be reported.

A receive operation shall never return data or ancillary data from more than one segment.

**2.10.12 Socket Out-of-Band Data State**

The handling of received out-of-band data is protocol-specific. Out-of-band data may be placed in the socket receive queue, either at the end of the queue or before all normal data in the queue. In this case, out-of-band data is returned to an application program by a normal receive call. Out-of-band data may also be queued separately rather than being placed in the socket receive queue, in which case it shall be returned only in response to a receive call that requests out-of-band data. It is protocol-specific whether an out-of-band data mark is placed in the receive queue to demarcate data preceding the out-of-band data and following the out-of-band data. An out-of-band data mark is logically an empty data segment that cannot be merged with other segments in the queue. An out-of-band data mark is never returned in response to an input operation. The `socketmark()` function can be used to test whether an out-of-band data mark is the first element in the queue. If an out-of-band data mark is the first element in the queue when an input function is called without the `MSG_PEEK` option, the mark is removed from the queue and the following data (if any) is processed as if the mark had not been present.

**2517 2.10.13 Connection Indication Queue**

2518 Sockets that are used to accept incoming connections maintain a queue of outstanding  
2519 connection indications. This queue is a list of connections that are awaiting acceptance by the  
2520 application; see *listen()*.

**2521 2.10.14 Signals**

2522 One category of event at the socket interface is the generation of signals. These signals report  
2523 protocol events or process errors relating to the state of the socket. The generation or delivery of  
2524 a signal does not change the state of the socket, although the generation of the signal may have  
2525 been caused by a state change.

2526 The SIGPIPE signal shall be sent to a thread that attempts to send data on a socket that is no  
2527 longer able to send. In addition, the send operation fails with the error [EPIPE].

2528 If a socket has an owner, the SIGURG signal is sent to the owner of the socket when it is notified  
2529 of expedited or out-of-band data. The socket state at this time is protocol-dependent, and the  
2530 status of the socket is specified in Section 2.10.17 (on page 516), Section 2.10.19 (on page 517),  
2531 and Section 2.10.20 (on page 517). Depending on the protocol, the expedited data may or may  
2532 not have arrived at the time of signal generation.

**2533 2.10.15 Asynchronous Errors**

2534 If any of the following conditions occur asynchronously for a socket, the corresponding value  
2535 listed below shall become the pending error for the socket:

2536 [ECONNABORTED]

2537 The connection was aborted locally.

2538 [ECONNREFUSED]

2539 For a connection-mode socket attempting a non-blocking connection, the attempt to connect  
2540 was forcefully rejected. For a connectionless-mode socket, an attempt to deliver a datagram  
2541 was forcefully rejected.

2542 [ECONNRESET]

2543 The peer has aborted the connection.

2544 [EHOSTDOWN]

2545 The destination host has been determined to be down or disconnected.

2546 [EHOSTUNREACH]

2547 The destination host is not reachable.

2548 [EMSGSIZE]

2549 For a connectionless-mode socket, the size of a previously sent datagram prevented  
2550 delivery.

2551 [ENETDOWN]

2552 The local network connection is not operational.

2553 [ENETRESET]

2554 The connection was aborted by the network.

2555 [ENETUNREACH]

2556 The destination network is not reachable.

2557 **2.10.16 Use of Options**

2558 There are a number of socket options which either specialize the behavior of a socket or provide  
 2559 useful information. These options may be set at different protocol levels and are always present  
 2560 at the uppermost “socket” level.

2561 Socket options are manipulated by two functions, *getsockopt()* and *setsockopt()*. These functions  
 2562 allow an application program to customize the behavior and characteristics of a socket to  
 2563 provide the desired effect.

2564 All of the options have default values. The type and meaning of these values is defined by the  
 2565 protocol level to which they apply. Instead of using the default values, an application program  
 2566 may choose to customize one or more of the options. However, in the bulk of cases, the default  
 2567 values are sufficient for the application.

2568 Some of the options are used to enable or disable certain behavior within the protocol modules  
 2569 (for example, turn on debugging) while others may be used to set protocol-specific information  
 2570 (for example, IP time-to-live on all the application’s outgoing packets). As each of the options is  
 2571 introduced, its effect on the underlying protocol modules is described.

2572 Table 2-1 shows the value for the socket level.

2573 **Table 2-1** Value of Level for Socket Options

Name	Description
SOL_SOCKET	Options are intended for the sockets level.

2576 Table 2-2 (on page 514) lists those options present at the socket level; that is, when the *level*  
 2577 parameter of the *getsockopt()* or *setsockopt()* function is SOL\_SOCKET, the types of the option  
 2578 value parameters associated with each option, and a brief synopsis of the meaning of the option  
 2579 value parameter. Unless otherwise noted, each may be examined with *getsockopt()* and set with  
 2580 *setsockopt()* on all types of socket.

Table 2-2 Socket-Level Options

2581  
2582  
2583  
2584  
2585  
2586  
2587  
2588  
2589  
2590  
2591  
2592  
2593  
2594  
2595  
2596  
2597  
2598  
2599  
2600  
2601  
2602  
2603  
2604  
2605  
2606  
2607  
2608  
2609  
2610

Option	Parameter Type	Parameter Meaning
SO_BROADCAST	int	Non-zero requests permission to transmit broadcast datagrams (SOCK_DGRAM sockets only).
SO_DEBUG	int	Non-zero requests debugging in underlying protocol modules.
SO_DONTROUTE	int	Non-zero requests bypass of normal routing; route based on destination address only.
SO_ERROR	int	Requests and clears pending error information on the socket ( <i>getsockopt()</i> only).
SO_KEEPALIVE	int	Non-zero requests periodic transmission of keepalive messages (protocol-specific).
SO_LINGER	struct linger	Specify actions to be taken for queued, unsent data on <i>close()</i> : linger on/off and linger time in seconds.
SO_OOBINLINE	int	Non-zero requests that out-of-band data be placed into normal data input queue as received.
SO_RCVBUF	int	Size of receive buffer (in bytes).
SO_RCVLOWAT	int	Minimum amount of data to return to application for input operations (in bytes).
SO_RCVTIMEO	struct timeval	Timeout value for a socket receive operation.
SO_REUSEADDR	int	Non-zero requests reuse of local addresses in <i>bind()</i> (protocol-specific).
SO_SNDBUF	int	Size of send buffer (in bytes).
SO_SNDLOWAT	int	Minimum amount of data to send for output operations (in bytes).
SO_SNDTIMEO	struct timeval	Timeout value for a socket send operation.
SO_TYPE	int	Identify socket type ( <i>getsockopt()</i> only).

2611  
2612  
2613  
2614  
2615  
2616  
2617  
2618  
2619  
2620  
2621  
2622  
2623  
2624  
2625  
2626  
2627

The SO\_BROADCAST option requests permission to send broadcast datagrams on the socket. Support for SO\_BROADCAST is protocol-specific. The default for SO\_BROADCAST is that the ability to send broadcast datagrams on a socket is disabled.

The SO\_DEBUG option enables debugging in the underlying protocol modules. This can be useful for tracing the behavior of the underlying protocol modules during normal system operation. The semantics of the debug reports are implementation-defined. The default value for SO\_DEBUG is for debugging to be turned off.

The SO\_DONTROUTE option requests that outgoing messages bypass the standard routing facilities. The destination must be on a directly-connected network, and messages are directed to the appropriate network interface according to the destination address. It is protocol-specific whether this option has any effect and how the outgoing network interface is chosen. Support for this option with each protocol is implementation-defined.

The SO\_ERROR option is used only on *getsockopt()*. When this option is specified, *getsockopt()* shall return any pending error on the socket and clear the error status. It shall return a value of 0 if there is no pending error. SO\_ERROR may be used to check for asynchronous errors on connected connectionless-mode sockets or for other types of asynchronous errors. SO\_ERROR has no default value.

2628 The SO\_KEEPALIVE option enables the periodic transmission of messages on a connected |  
2629 socket. The behavior of this option is protocol-specific. The default value for SO\_KEEPALIVE is |  
2630 zero, specifying that this capability is turned off. |

2631 The SO\_LINGER option controls the action of the interface when unsent messages are queued |  
2632 on a socket and a *close()* is performed. The details of this option are protocol-specific. The |  
2633 default value for SO\_LINGER is zero, or off, for the *L\_onoff* element of the option value and zero |  
2634 seconds for the linger time specified by the *L\_linger* element. |

2635 The SO\_OOBINLINE option is valid only on protocols that support out-of-band data. The |  
2636 SO\_OOBINLINE option requests that out-of-band data be placed in the normal data input queue |  
2637 as received; it is then accessible using the *read()* or *recv()* functions without the MSG\_OOB flag |  
2638 set. The default for SO\_OOBINLINE is off; that is, for out-of-band data not to be placed in the |  
2639 normal data input queue. |

2640 The SO\_RCVBUF option requests that the buffer space allocated for receive operations on this |  
2641 socket be set to the value, in bytes, of the option value. Applications may wish to increase buffer |  
2642 size for high volume connections, or may decrease buffer size to limit the possible backlog of |  
2643 incoming data. The default value for the SO\_RCVBUF option value is implementation-defined, |  
2644 and may vary by protocol. |

2645 The maximum value for the option for a socket may be obtained by the use of the *fpathconf()* |  
2646 function, using the value *\_PC\_SOCKET\_MAXBUF*. |

2647 The SO\_RCVLOWAT option sets the minimum number of bytes to process for socket input |  
2648 operations. In general, receive calls block until any (non-zero) amount of data is received, then |  
2649 return the smaller of the amount available or the amount requested. The default value for |  
2650 SO\_RCVLOWAT is 1, and does not affect the general case. If SO\_RCVLOWAT is set to a larger |  
2651 value, blocking receive calls normally wait until they have received the smaller of the low water |  
2652 mark value or the requested amount. Receive calls may still return less than the low water mark |  
2653 if an error occurs, a signal is caught, or the type of data next in the receive queue is different |  
2654 from that returned (for example, out-of-band data). As mentioned previously, the default value |  
2655 for SO\_RCVLOWAT is 1 byte. It is implementation-defined whether the SO\_RCVLOWAT option |  
2656 can be set. |

2657 The SO\_RCVTIMEO option is an option to set a timeout value for input operations. It accepts a |  
2658 **timeval** structure with the number of seconds and microseconds specifying the limit on how |  
2659 long to wait for an input operation to complete. If a receive operation has blocked for this much |  
2660 time without receiving additional data, it shall return with a partial count or *errno* shall be set to |  
2661 [EWOULDBLOCK] if no data were received. The default for this option is the value zero, which |  
2662 indicates that a receive operation will not timeout. It is implementation-defined whether the |  
2663 SO\_RCVTIMEO option can be set. |

2664 The SO\_REUSEADDR option indicates that the rules used in validating addresses supplied in a |  
2665 *bind()* should allow reuse of local addresses. Operation of this option is protocol-specific. The |  
2666 default value for SO\_REUSEADDR is off; that is, reuse of local addresses is not permitted. |

2667 The SO\_SNDBUF option requests that the buffer space allocated for send operations on this |  
2668 socket be set to the value, in bytes, of the option value. The default value for the SO\_SNDBUF |  
2669 option value is implementation-defined, and may vary by protocol. The maximum value for the |  
2670 option for a socket may be obtained by the use of the *fpathconf()* function, using the value |  
2671 *\_PC\_SOCKET\_MAXBUF*. |

2672 The SO\_SNDLOWAT option sets the minimum number of bytes to process for socket output |  
2673 operations. Most output operations process all of the data supplied by the call, delivering data to |  
2674 the protocol for transmission and blocking as necessary for flow control. Non-blocking output |  
2675 operations process as much data as permitted subject to flow control without blocking, but |

2676 process no data if flow control does not allow the smaller of the send low water mark value or  
 2677 the entire request to be processed. A *select()* operation testing the ability to write to a socket shall  
 2678 return true only if the send low water mark could be processed. The default value for  
 2679 SO\_SNDLOWAT is implementation-defined and protocol-specific. It is implementation-defined  
 2680 whether the SO\_SNDLOWAT option can be set.

2681 The SO\_SNDTIMEO option is an option to set a timeout value for the amount of time that an  
 2682 output function shall block because flow control prevents data from being sent. As noted in  
 2683 Table 2-2 (on page 514), the option value is a **timeval** structure with the number of seconds and  
 2684 microseconds specifying the limit on how long to wait for an output operation to complete. If a  
 2685 send operation has blocked for this much time, it shall return with a partial count or *errno* set to  
 2686 [EWOULDBLOCK] if no data were sent. The default for this option is the value zero, which  
 2687 indicates that a send operation will not timeout. It is implementation-defined whether the  
 2688 SO\_SNDTIMEO option can be set.

2689 The SO\_TYPE option is used only on *getsockopt()*. When this option is specified, *getsockopt()*  
 2690 shall return the type of the socket (for example, SOCK\_STREAM). This option is useful to  
 2691 servers that inherit sockets on start-up. SO\_TYPE has no default value.

## 2692 2.10.17 Use of Sockets for Local UNIX Connections

2693 Support for UNIX domain sockets is mandatory.

2694 UNIX domain sockets provide process-to-process communication in a single system.

### 2695 2.10.17.1 Headers

2696 The symbolic constant AF\_UNIX defined in the `<sys/socket.h>` header is used to identify the  
 2697 UNIX domain address family. The `<sys/un.h>` header contains other definitions used in  
 2698 connection with UNIX domain sockets. See the Base Definitions volume of IEEE Std 1003.1-200x,  
 2699 Chapter 13, Headers.

2700 The **sockaddr\_storage** structure defined in `<sys/socket.h>` shall be large enough to  
 2701 accommodate a **sockaddr\_un** structure (see the `<sys/un.h>` header defined in the Base  
 2702 Definitions volume of IEEE Std 1003.1-200x, Chapter 13, Headers) and shall be aligned at an  
 2703 appropriate boundary so that pointers to it can be cast as pointers to **sockaddr\_un** structures  
 2704 and used to access the fields of those structures without alignment problems. When a  
 2705 **sockaddr\_storage** structure is cast as a **sockaddr\_un** structure, the *ss\_family* field maps onto the  
 2706 *sun\_family* field.

## 2707 2.10.18 Use of Sockets over Internet Protocols

2708 When a socket is created in the Internet family with a protocol value of zero, the implementation  
 2709 shall use the protocol listed below for the type of socket created.

2710 SOCK\_STREAM      IPPROTO\_TCP.

2711 SOCK\_DGRAM      IPPROTO\_UDP.

2712 RS      SOCK\_RAW      IPPROTO\_RAW.

2713 SOCK\_SEQPACKET    Unspecified.

2714 RS      A raw interface to IP is available by creating an Internet socket of type SOCK\_RAW. The default  
 2715 protocol for type SOCK\_RAW shall be identified in the IP header with the value  
 2716 IPPROTO\_RAW. Applications should not use the default protocol when creating a socket with  
 2717 type SOCK\_RAW, but should identify a specific protocol by value. The ICMP control protocol is  
 2718 accessible from a raw socket by specifying a value of IPPROTO\_ICMP for protocol.

## 2719 **2.10.19 Use of Sockets over Internet Protocols Based on IPv4**

2720 Support for sockets over Internet protocols based on IPv4 is mandatory.

### 2721 *2.10.19.1 Headers*

2722 The symbolic constant `AF_INET` defined in the `<sys/socket.h>` header is used to identify the  
 2723 IPv4 Internet address family. The `<netinet/in.h>` header contains other definitions used in  
 2724 connection with IPv4 Internet sockets. See the Base Definitions volume of IEEE Std 1003.1-200x,  
 2725 Chapter 13, Headers.

2726 The `sockaddr_storage` structure defined in `<sys/socket.h>` shall be large enough to  
 2727 accommodate a `sockaddr_in` structure (see the `<netinet/in.h>` header defined in the Base  
 2728 Definitions volume of IEEE Std 1003.1-200x, Chapter 13, Headers) and shall be aligned at an  
 2729 appropriate boundary so that pointers to it can be cast as pointers to `sockaddr_in` structures and  
 2730 used to access the fields of those structures without alignment problems. When a  
 2731 `sockaddr_storage` structure is cast as a `sockaddr_in` structure, the `ss_family` field maps onto the  
 2732 `sin_family` field.

## 2733 **2.10.20 Use of Sockets over Internet Protocols Based on IPv6**

2734 IP6 This section describes extensions to support sockets over Internet protocols based on IPv6. This  
 2735 functionality is dependent on support of the IPV6 option (and the rest of this section is not  
 2736 further shaded for this option).

2737 To enable smooth transition from IPv4 to IPv6, the features defined in this section may, in certain  
 2738 circumstances, also be used in connection with IPv4; see Section 2.10.20.2 (on page 518).

### 2739 *2.10.20.1 Addressing*

2740 IPv6 overcomes the addressing limitations of previous versions by using 128-bit addresses  
 2741 instead of 32-bit addresses. The IPv6 address architecture is described in RFC 2373.

2742 There are three kinds of IPv6 address:

#### 2743 Unicast

2744 Identifies a single interface.

2745 A unicast address can be global, link-local (designed for use on a single link), or site-local  
 2746 (designed for systems not connected to the Internet). Link-local and site-local addresses  
 2747 need not be globally unique.

#### 2748 Anycast

2749 Identifies a set of interfaces such that a packet sent to the address can be delivered to any  
 2750 member of the set.

2751 An anycast address is similar to a unicast address; the nodes to which an anycast address is  
 2752 assigned must be explicitly configured to know that it is an anycast address.

#### 2753 Multicast

2754 Identifies a set of interfaces such that a packet sent to the address should be delivered to  
 2755 every member of the set.

2756 An application can send multicast datagrams by simply specifying an IPv6 multicast  
 2757 address in the `address` argument of `sendto()`. To receive multicast datagrams, an application  
 2758 must join the multicast group (using `setsockopt()` with `IPV6_JOIN_GROUP`) and must bind  
 2759 to the socket the UDP port on which datagrams will be received. Some applications should  
 2760 also bind the multicast group address to the socket, to prevent other datagrams destined to  
 2761 that port from being delivered to the socket.

2762 A multicast address can be global, node-local, link-local, site-local, or organization-local.  
2763 The following special IPv6 addresses are defined:  
2764 Unspecified  
2765 An address that is not assigned to any interface and is used to indicate the absence of an  
2766 address.  
2767 Loopback  
2768 A unicast address that is not assigned to any interface and can be used by a node to send  
2769 packets to itself.  
2770 Two sets of IPv6 addresses are defined to correspond to IPv4 addresses:  
2771 IPv4-compatible addresses  
2772 These are assigned to nodes that support IPv6 and can be used when traffic is “tunneled”  
2773 through IPv4.  
2774 IPv4-mapped addresses  
2775 These are used to represent IPv4 addresses in IPv6 address format; see Section 2.10.20.2.  
2776 Note that the unspecified address and the loopback address must not be treated as IPv4-  
2777 compatible addresses.

#### 2778 2.10.20.2 Compatibility with IPv4

2779 The API provides the ability for IPv6 applications to interoperate with applications using IPv4,  
2780 by using IPv4-mapped IPv6 addresses. These addresses can be generated automatically by the  
2781 *getaddrinfo()* function when the specified host has only IPv4 addresses.

2782 Applications may use AF\_INET6 sockets to open TCP connections to IPv4 nodes, or send UDP  
2783 packets to IPv4 nodes, by simply encoding the destination's IPv4 address as an IPv4-mapped  
2784 IPv6 address, and passing that address, within a **sockaddr\_in6** structure, in the *connect()*,  
2785 *sendto()* or *sendmsg()* function. When applications use AF\_INET6 sockets to accept TCP  
2786 connections from IPv4 nodes, or receive UDP packets from IPv4 nodes, the system shall return  
2787 the peer's address to the application in the *accept()*, *recvfrom()*, *recvmsg()*, or *getpeername()*  
2788 function using a **sockaddr\_in6** structure encoded this way. If a node has an IPv4 address, then  
2789 the implementation may allow applications to communicate using that address via an  
2790 AF\_INET6 socket. In such a case, the address will be represented at the API by the  
2791 corresponding IPv4-mapped IPv6 address. Also, the implementation may allow an AF\_INET6  
2792 socket bound to **in6addr\_any** to receive inbound connections and packets destined to one of the  
2793 node's IPv4 addresses.

2794 An application may use AF\_INET6 sockets to bind to a node's IPv4 address by specifying the  
2795 address as an IPv4-mapped IPv6 address in a **sockaddr\_in6** structure in the *bind()* function. For  
2796 an AF\_INET6 socket bound to a node's IPv4 address, the system shall return the address in the  
2797 *getsockname()* function as an IPv4-mapped IPv6 address in a **sockaddr\_in6** structure.

#### 2798 2.10.20.3 Interface Identification

2799 Each local interface is assigned a unique positive integer as a numeric index. Indexes start at 1;  
2800 zero is not used. There may be gaps so that there is no current interface for a particular positive  
2801 index. Each interface also has a unique implementation-defined name.

## 2802 2.10.20.4 Options

2803 The following options apply at the IPPROTO\_IPV6 level:

## 2804 IPV6\_JOIN\_GROUP

2805 When set via *setsockopt()*, it joins the application to a multicast group on an interface  
 2806 (identified by its index) and addressed by a given multicast address, enabling packets sent  
 2807 to that address to be read via the socket. If the interface index is specified as zero, the  
 2808 system selects the interface (for example, by looking up the address in a routing table and  
 2809 using the resulting interface).

2810 An attempt to read this option using *getsockopt()* shall result in an [EOPNOTSUPP] error.

2811 The value of this option is an **ipv6\_mreq** structure.

## 2812 IPV6\_LEAVE\_GROUP

2813 When set via *setsockopt()*, it removes the application from the multicast group on an  
 2814 interface (identified by its index) and addressed by a given multicast address.

2815 An attempt to read this option using *getsockopt()* shall result in an [EOPNOTSUPP] error.

2816 The value of this option is an **ipv6\_mreq** structure.

## 2817 IPV6\_MULTICAST\_HOPS

2818 The value of this option is the hop limit for outgoing multicast IPv6 packets sent via the  
 2819 socket. Its possible values are the same as those of IPV6\_UNICAST\_HOPS. If the  
 2820 IPV6\_MULTICAST\_HOPS option is not set, a value of 1 is assumed. This option can be set  
 2821 via *setsockopt()* and read via *getsockopt()*.

## 2822 IPV6\_MULTICAST\_IF

2823 The index of the interface to be used for outgoing multicast packets. It can be set via  
 2824 *setsockopt()* and read via *getsockopt()*.

## 2825 IPV6\_MULTICAST\_LOOP

2826 This option controls whether outgoing multicast packets should be delivered back to the  
 2827 local application when the sending interface is itself a member of the destination multicast  
 2828 group. If it is set to 1 they are delivered. If it is set to 0 they are not. Other values result in an  
 2829 [EINVAL] error. This option can be set via *setsockopt()* and read via *getsockopt()*.

## 2830 IPV6\_UNICAST\_HOPS

2831 The value of this option is the hop limit for outgoing unicast IPv6 packets sent via the  
 2832 socket. If the option is not set, or is set to  $-1$ , the system selects a default value. Attempts to  
 2833 set a value less than  $-1$  or greater than 255 shall result in an [EINVAL] error. This option can  
 2834 be set via *setsockopt()* and read via *getsockopt()*.

## 2835 IPV6\_V6ONLY

2836 This socket option restricts AF\_INET6 sockets to IPv6 communications only. AF\_INET6  
 2837 sockets may be used for both IPv4 and IPv6 communications. Some applications may want  
 2838 to restrict their use of an AF\_INET6 socket to IPv6 communications only. For these  
 2839 applications, the IPV6\_V6ONLY socket option is defined. When this option is turned on, the  
 2840 socket can be used to send and receive IPv6 packets only. This is an IPPROTO\_IPV6-level  
 2841 option. This option takes an **int** value. This is a Boolean option. By default, this option is  
 2842 turned off.

2843 An [EOPNOTSUPP] error shall result if IPV6\_JOIN\_GROUP or IPV6\_LEAVE\_GROUP is used  
 2844 with *getsockopt()*.

## 2845 2.10.20.5 Headers

2846 The symbolic constant AF\_INET6 is defined in the `<sys/socket.h>` header to identify the IPv6  
 2847 Internet address family. See the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 13,  
 2848 Headers.

2849 The `sockaddr_storage` structure defined in `<sys/socket.h>` shall be large enough to  
 2850 accommodate a `sockaddr_in6` structure (see the `<netinet/in.h>` header defined in the Base  
 2851 Definitions volume of IEEE Std 1003.1-200x, Chapter 13, Headers) and shall be aligned at an  
 2852 appropriate boundary so that pointers to it can be cast as pointers to `sockaddr_in6` structures  
 2853 and used to access the fields of those structures without alignment problems. When a  
 2854 `sockaddr_storage` structure is cast as a `sockaddr_in6` structure, the `ss_family` field maps onto the  
 2855 `sin6_family` field.

2856 The `<netinet/in.h>`, `<arpa/inet.h>`, and `<netdb.h>` headers contain other definitions used in  
 2857 connection with IPv6 Internet sockets; see the Base Definitions volume of IEEE Std 1003.1-200x,  
 2858 Chapter 13, Headers.

## 2859 2.11 Tracing

2860 TRC This section describes extensions to support tracing of user applications. This functionality is  
 2861 dependent on support of the Trace option (and the rest of this section is not further shaded for  
 2862 this option).

2863 The tracing facilities defined in IEEE Std 1003.1-200x allow a process to select a set of trace event  
 2864 types, to activate a trace stream of the selected trace events as they occur in the flow of  
 2865 execution, and to retrieve the recorded trace events.

2866 The tracing operation relies on three logically different components: the traced process, the  
 2867 controller process, and the analyzer process. During the execution of the traced process, when a  
 2868 trace point is reached, a trace event is recorded into the trace streams created for that process in  
 2869 which the associated trace event type identifier is not being filtered out. The controller process  
 2870 controls the operation of recording the trace events into the trace stream. It shall be able to:

- 2871 • Initialize the attributes of a trace stream
- 2872 • Create the trace stream (for a specified traced process) using those attributes
- 2873 • Start and stop tracing for the trace stream
- 2874 • Filter the type of trace events to be recorded, if the Trace Event Filter option is supported
- 2875 • Shut a trace stream down

2876 These operations can be done for an active trace stream. The analyzer process retrieves the  
 2877 traced events either at runtime, when the trace stream has not yet been shut down, but is still  
 2878 recording trace events; or after opening a trace log that had been previously recorded and shut  
 2879 down. These three logically different operations can be performed by the same process, or can be  
 2880 distributed into different processes.

2881 A trace stream identifier can be created by a call to `posix_trace_create()`,  
 2882 `posix_trace_create_withlog()`, or `posix_trace_open()`. The `posix_trace_create()` and  
 2883 `posix_trace_create_withlog()` functions should be used by a controller process. The  
 2884 `posix_trace_open()` should be used by an analyzer process.

2885 The tracing functions can serve different purposes. One purpose is debugging the possibly pre-  
 2886 instrumented code, while another is post-mortem fault analysis. These two potential uses differ  
 2887 in that the first requires pre-filtering capabilities to avoid overwhelming the trace stream and

2888 permits focusing on expected information; while the second needs comprehensive trace  
2889 capabilities in order to be able to record all types of information.

2890 The events to be traced belong to two classes:

- 2891 1. User trace events (generated by the application instrumentation)
- 2892 2. System trace events (generated by the operating system)

2893 The trace interface defines several system trace event types associated with control of and  
2894 operation of the trace stream. This small set of system trace events includes the minimum  
2895 required to interpret correctly the trace event information present in the stream. Other desirable  
2896 system trace events for some particular application profile may be implemented and are  
2897 encouraged; for example, process and thread scheduling, signal occurrence, and so on.

2898 Each traced process shall have a mapping of the trace event names to trace event type identifiers  
2899 that have been defined for that process. Each active trace stream shall have a mapping that  
2900 incorporates all the trace event type identifiers predefined by the trace system plus all the  
2901 mappings of trace event names to trace event type identifiers of the processes that are being  
2902 traced into that trace stream. These mappings are defined from the instrumented application by  
2903 calling the *posix\_trace\_eventid\_open()* function and from the controller process by calling the  
2904 *posix\_trace\_trid\_eventid\_open()* function. For a pre-recorded trace stream, the list of trace event  
2905 types is obtained from the pre-recorded trace log.

2906 The *st\_ctime* and *st\_mtime* fields of a file associated with an active trace stream shall be marked  
2907 for update every time any of the tracing operations modifies that file.

2908 The *st\_atime* field of a file associated with a trace stream shall be marked for update every time  
2909 any of the tracing operations causes data to be read from that file.

2910 Results are undefined if the application performs any operation on a file descriptor associated  
2911 with an active or pre-recorded trace stream until *posix\_trace\_shutdown()* or *posix\_trace\_close()* is  
2912 called for that trace stream.

2913 The main purpose of this option is to define a complete set of functions and concepts that allow  
2914 a conforming application to be traced from creation to termination, whatever its realtime  
2915 constraints and properties.

## 2916 2.11.1 Tracing Data Definitions

### 2917 2.11.1.1 Structures

2918 The `<trace.h>` header shall define the *posix\_trace\_status\_info* and *posix\_trace\_event\_info* structures  
2919 described below. Implementations may add extensions to these structures.

#### 2920 **posix\_trace\_status\_info Structure**

2921 To facilitate control of a trace stream, information about the current state of an active trace  
2922 stream can be obtained dynamically. This structure is returned by a call to the  
2923 *posix\_trace\_get\_status()* function.

2924 The **posix\_trace\_status\_info** structure defined in `<trace.h>` shall contain at least the following  
2925 members:

2926  
2927  
2928  
2929  
2930  
2931

Member Type	Member Name	Description
<b>int</b>	<i>posix_stream_status</i>	The operating mode of the trace stream.
<b>int</b>	<i>posix_stream_full_status</i>	The full status of the trace stream.
<b>int</b>	<i>posix_stream_overrun_status</i>	Indicates whether trace events were lost in the trace stream.

2932  
2933

If the Trace Log option is supported in addition to the Trace option, the **posix\_trace\_status\_info** structure defined in <trace.h> shall contain at least the following additional members:

2934  
2935  
2936  
2937  
2938  
2939  
2940  
2941

Member Type	Member Name	Description
<b>int</b>	<i>posix_stream_flush_status</i>	Indicates whether a flush is in progress.
<b>int</b>	<i>posix_stream_flush_error</i>	Indicates whether any error occurred during the last flush operation.
<b>int</b>	<i>posix_log_overrun_status</i>	Indicates whether trace events were lost in the trace log.
<b>int</b>	<i>posix_log_full_status</i>	The full status of the trace log.

2942  
2943

The *posix\_stream\_status* member indicates the operating mode of the trace stream and shall have one of the following values defined by manifest constants in the <trace.h> header:

2944  
2945

POSIX\_TRACE\_RUNNING

Tracing is in progress; that is, the trace stream is accepting trace events.

2946  
2947  
2948  
2949

POSIX\_TRACE\_SUSPENDED

The trace stream is not accepting trace events. The tracing operation has not yet started or has stopped, either following a *posix\_trace\_stop()* function call or because the trace resources are exhausted.

2950  
2951

The *posix\_stream\_full\_status* member indicates the full status of the trace stream, and it shall have one of the following values defined by manifest constants in the <trace.h> header:

2952  
2953

POSIX\_TRACE\_FULL

The space in the trace stream for trace events is exhausted.

2954  
2955

POSIX\_TRACE\_NOT\_FULL

There is still space available in the trace stream.

2956  
2957

The combination of the *posix\_stream\_status* and *posix\_stream\_full\_status* members also indicates the actual status of the stream. The status shall be interpreted as follows:

2958  
2959

POSIX\_TRACE\_RUNNING and POSIX\_TRACE\_NOT\_FULL

This status combination indicates that tracing is in progress, and there is space available for recording more trace events.

2960  
2961  
2962  
2963  
2964  
2965  
2966

POSIX\_TRACE\_RUNNING and POSIX\_TRACE\_FULL

This status combination indicates that tracing is in progress and that the trace stream is full of trace events. This status combination cannot occur unless the *stream-full-policy* is set to POSIX\_TRACE\_LOOP. The trace stream contains trace events recorded during a moving time window of prior trace events, and some older trace events may have been overwritten and thus lost.

2967  
2968  
2969

POSIX\_TRACE\_SUSPENDED and POSIX\_TRACE\_NOT\_FULL

This status combination indicates that tracing has not yet been started, has been stopped by the *posix\_trace\_stop()* function, or has been cleared by the *posix\_trace\_clear()* function.

2970 POSIX\_TRACE\_SUSPENDED and POSIX\_TRACE\_FULL  
 2971 This status combination indicates that tracing has been stopped by the implementation  
 2972 because the *stream-full-policy* attribute was POSIX\_TRACE\_UNTIL\_FULL and trace  
 2973 resources were exhausted, or that the trace stream was stopped by the function  
 2974 *posix\_trace\_stop()* at a time when trace resources were exhausted.

2975 The *posix\_stream\_ouerrun\_status* member indicates whether trace events were lost in the trace  
 2976 stream, and shall have one of the following values defined by manifest constants in the  
 2977 `<trace.h>` header:

2978 POSIX\_TRACE\_OVERRUN  
 2979 At least one trace event was lost and thus was not recorded in the trace stream.

2980 POSIX\_TRACE\_NO\_OVERRUN  
 2981 No trace events were lost.

2982 When the corresponding trace stream is created, the *posix\_stream\_ouerrun\_status* member shall be  
 2983 set to POSIX\_TRACE\_NO\_OVERRUN.

2984 Whenever an overrun occurs, *posix\_stream\_ouerrun\_status* member shall be set to  
 2985 POSIX\_TRACE\_OVERRUN.

2986 An overrun occurs when:

- 2987 • The policy is POSIX\_TRACE\_LOOP and a recorded trace event is overwritten.
- 2988 • The policy is POSIX\_TRACE\_UNTIL\_FULL and the trace stream is full when a trace event is  
 2989 generated.
- 2990 • If the Trace Log option is supported, the policy is POSIX\_TRACE\_FLUSH and at least one  
 2991 trace event is lost while flushing the trace stream to the trace log.

2992 The *posix\_stream\_ouerrun\_status* member is reset to zero after its value is read.

2993 If the Trace Log option is supported in addition to the Trace option, the *posix\_stream\_flush\_status*,  
 2994 *posix\_stream\_flush\_error*, *posix\_log\_ouerrun\_status*, and *posix\_log\_full\_status* members are defined  
 2995 as follows; otherwise, they are undefined.

2996 The *posix\_stream\_flush\_status* member indicates whether a flush operation is being performed  
 2997 and shall have one of the following values defined by manifest constants in the header  
 2998 `<trace.h>`:

2999 POSIX\_TRACE\_FLUSHING  
 3000 The trace stream is currently being flushed to the trace log.

3001 POSIX\_TRACE\_NOT\_FLUSHING  
 3002 No flush operation is in progress.

3003 The *posix\_stream\_flush\_status* member shall be set to POSIX\_TRACE\_FLUSHING if a flush  
 3004 operation is in progress either due to a call to the *posix\_trace\_flush()* function (explicit or caused  
 3005 by a trace stream shutdown operation) or because the trace stream has become full with the  
 3006 *stream-full-policy* attribute set to POSIX\_TRACE\_FLUSH. The *posix\_stream\_flush\_status* member  
 3007 shall be set to POSIX\_TRACE\_NOT\_FLUSHING if no flush operation is in progress.

3008 The *posix\_stream\_flush\_error* member shall be set to zero if no error occurred during flushing. If  
 3009 an error occurred during a previous flushing operation, the *posix\_stream\_flush\_error* member  
 3010 shall be set to the value of the first error that occurred. If more than one error occurs while  
 3011 flushing, error values after the first shall be discarded. The *posix\_stream\_flush\_error* member is  
 3012 reset to zero after its value is read.

3013 The *posix\_log\_outrun\_status* member indicates whether trace events were lost in the trace log,  
 3014 and shall have one of the following values defined by manifest constants in the `<trace.h>`  
 3015 header:

3016 POSIX\_TRACE\_OVERRUN

3017 At least one trace event was lost.

3018 POSIX\_TRACE\_NO\_OVERRUN

3019 No trace events were lost.

3020 When the corresponding trace stream is created, the *posix\_log\_outrun\_status* member shall be set  
 3021 to POSIX\_TRACE\_NO\_OVERRUN. Whenever an overrun occurs, this status shall be set to  
 3022 POSIX\_TRACE\_OVERRUN. The *posix\_log\_outrun\_status* member is reset to zero after its value  
 3023 is read.

3024 The *posix\_log\_full\_status* member indicates the full status of the trace log, and it shall have one of  
 3025 the following values defined by manifest constants in the `<trace.h>` header:

3026 POSIX\_TRACE\_FULL

3027 The space in the trace log is exhausted.

3028 POSIX\_TRACE\_NOT\_FULL

3029 There is still space available in the trace log.

3030 The *posix\_log\_full\_status* member is only meaningful if the *log-full-policy* attribute is either  
 3031 POSIX\_TRACE\_UNTIL\_FULL or POSIX\_TRACE\_LOOP.

3032 For an active trace stream without log, that is created by the *posix\_trace\_create()* function, the  
 3033 *posix\_log\_outrun\_status* member shall be set to POSIX\_TRACE\_NO\_OVERRUN and the  
 3034 *posix\_log\_full\_status* member shall be set to POSIX\_TRACE\_NOT\_FULL.

### 3035 **posix\_trace\_event\_info** Structure

3036 The trace event structure **posix\_trace\_event\_info** contains the information for one recorded  
 3037 trace event. This structure is returned by the set of functions *posix\_trace\_getnext\_event()*,  
 3038 *posix\_trace\_timedgetnext\_event()*, and *posix\_trace\_trygetnext\_event()*.

3039 The **posix\_trace\_event\_info** structure defined in `<trace.h>` shall contain at least the following  
 3040 members:

3041

Member Type	Member Name	Description
<b>trace_event_id_t</b>	<i>posix_event_id</i>	Trace event type identification.
<b>pid_t</b>	<i>posix_pid</i>	Process ID of the process that generated the trace event.
<b>void *</b>	<i>posix_prog_address</i>	Address at which the trace point was invoked.
<b>int</b>	<i>posix_truncation_status</i>	Status about the truncation of the data associated with this trace event.
<b>struct timespec</b>	<i>posix_timestamp</i>	Time at which the trace event was generated.

3050 In addition, if the Trace option and the Threads option are both supported, the  
 3051 **posix\_trace\_event\_info** structure defined in `<trace.h>` shall contain the following additional  
 3052 member:

3053  
3054  
3055  
3056  
3057

Member Type	Member Name	Description
<b>pthread_t</b>	<i>posix_thread_id</i>	Thread ID of the thread that generated the trace event.

3058  
3059  
3060  
3061  
3062

The *posix\_event\_id* member represents the identification of the trace event type and its value is not directly defined by the user. This identification is returned by a call to one of the following functions: *posix\_trace\_trid\_eventid\_open()*, *posix\_trace\_eventtypelist\_getnext\_id()*, or *posix\_trace\_eventid\_open()*. The name of the trace event type can be obtained by calling *posix\_trace\_eventid\_get\_name()*.

3063  
3064  
3065

The *posix\_pid* is the process identifier of the traced process which generated the trace event. If the *posix\_event\_id* member is one of the implementation-defined system trace events and that trace event is not associated with any process, the *posix\_pid* member shall be set to zero.

3066  
3067  
3068  
3069  
3070

For a user trace event, the *posix\_prog\_address* member is the process mapped address of the point at which the associated call to the *posix\_trace\_event()* function was made. For a system trace event, if the trace event is caused by a system service explicitly called by the application, the *posix\_prog\_address* member shall be the address of the process at the point where the call to that system service was made.

3071  
3072  
3073  
3074  
3075

The *posix\_truncation\_status* member defines whether the data associated with a trace event has been truncated at the time the trace event was generated, or at the time the trace event was read from the trace stream, or (if the Trace Log option is supported) from the trace log (see the *event* argument from the *posix\_trace\_getnext\_event()* function). The *posix\_truncation\_status* member shall have one of the following values defined by manifest constants in the `<trace.h>` header:

3076  
3077

POSIX\_TRACE\_NOT\_TRUNCATED

All the traced data is available.

3078  
3079

POSIX\_TRACE\_TRUNCATED\_RECORD

Data was truncated at the time the trace event was generated.

3080  
3081  
3082  
3083

POSIX\_TRACE\_TRUNCATED\_READ

Data was truncated at the time the trace event was read from a trace stream or a trace log because the reader's buffer was too small. This truncation status overrides the POSIX\_TRACE\_TRUNCATED\_RECORD status.

3084  
3085  
3086

The *posix\_timestamp* member shall be the time at which the trace event was generated. The clock used is implementation-defined, but the resolution of this clock can be retrieved by a call to the *posix\_trace\_attr\_getclockres()* function.

3087

If the Threads option is supported in addition to the Trace option:

3088  
3089  
3090

- The *posix\_thread\_id* member is the identifier of the thread that generated the trace event. If the *posix\_event\_id* member is one of the implementation-defined system trace events and that trace event is not associated with any thread, the *posix\_thread\_id* member shall be set to zero.

3091

Otherwise, this member is undefined.

### 3092 2.11.1.2 Trace Stream Attributes

3093  
3094

Trace streams have attributes that compose the **posix\_trace\_attr\_t** trace stream attributes object. This object shall contain at least the following attributes:

3095

- The *generation-version* attribute identifies the origin and version of the trace system.

- 3096 • The *trace-name* attribute is a character string defined by the trace controller, and that  
3097 identifies the trace stream.
- 3098 • The *creation-time* attribute represents the time of the creation of the trace stream.
- 3099 • The *clock-resolution* attribute defines the clock resolution of the clock used to generate  
3100 timestamps.
- 3101 • The *stream-min-size* attribute defines the minimum size in bytes of the trace stream strictly  
3102 reserved for the trace events.
- 3103 • The *stream-full-policy* attribute defines the policy followed when the trace stream is full; its  
3104 value is `POSIX_TRACE_LOOP`, `POSIX_TRACE_UNTIL_FULL`, or `POSIX_TRACE_FLUSH`.
- 3105 • The *max-data-size* attribute defines the maximum record size in bytes of a trace event.

3106 In addition, if the Trace option and the Trace Inherit option are both supported, the  
3107 **posix\_trace\_attr\_t** trace stream creation attributes object shall contain at least the following  
3108 attributes:

- 3109 • The *inheritance* attribute specifies whether a newly created trace stream will inherit tracing in  
3110 its parent's process trace stream. It is either `POSIX_TRACE_INHERITED` or  
3111 `POSIX_TRACE_CLOSE_FOR_CHILD`.

3112 In addition, if the Trace option and the Trace Log option are both supported, the  
3113 **posix\_trace\_attr\_t** trace stream creation attributes object shall contain at least the following  
3114 attribute:

- 3115 • If the file type corresponding to the trace log supports the `POSIX_TRACE_LOOP` or the  
3116 `POSIX_TRACE_UNTIL_FULL` policies, the *log-max-size* attribute defines the maximum size  
3117 in bytes of the trace log associated with an active trace stream. Other stream data—for  
3118 example, trace attribute values—shall not be included in this size.
- 3119 • The *log-full-policy* attribute defines the policy of a trace log associated with an active trace  
3120 stream to be `POSIX_TRACE_LOOP`, `POSIX_TRACE_UNTIL_FULL`, or  
3121 `POSIX_TRACE_APPEND`.

## 3122 2.11.2 Trace Event Type Definitions

### 3123 2.11.2.1 System Trace Event Type Definitions

3124 The following system trace event types, defined in the `<trace.h>` header, track the invocation of  
3125 the trace operations:

- 3126 • `POSIX_TRACE_START` shall be associated with a trace start operation.
- 3127 • `POSIX_TRACE_STOP` shall be associated with a trace stop operation.
- 3128 • if the Trace Event Filter option is supported, `POSIX_TRACE_FILTER` shall be associated with  
3129 a trace event type filter change operation.

3130 The following system trace event types, defined in the `<trace.h>` header, report operational trace  
3131 events:

- 3132 • `POSIX_TRACE_OVERFLOW` shall mark the beginning of a trace overflow condition.
- 3133 • `POSIX_TRACE_RESUME` shall mark the end of a trace overflow condition.
- 3134 • If the Trace Log option is supported, `POSIX_TRACE_FLUSH_START` shall mark the  
3135 beginning of a flush operation.

3136 • If the Trace Log option is supported, POSIX\_TRACE\_FLUSH\_STOP shall mark the end of a  
 3137 flush operation.

3138 • If an implementation-defined trace error condition is reported, it shall be marked  
 3139 POSIX\_TRACE\_ERROR.

3140 The interpretation of a trace stream or a trace log by a trace analyzer process relies on the  
 3141 information recorded for each trace event, and also on system trace events that indicate the  
 3142 invocation of trace control operations and trace system operational trace events.

3143 The POSIX\_TRACE\_START and POSIX\_TRACE\_STOP trace events specify the time windows  
 3144 during which the trace stream is running.

3145 The POSIX\_TRACE\_STOP trace event with an associated data that is equal to zero indicates  
 3146 a call of the function *posix\_trace\_stop()*.

3147 The POSIX\_TRACE\_STOP trace event with an associated data that is different from zero  
 3148 indicates an automatic stop of the trace stream (see *posix\_trace\_attr\_getstreamfullpolicy()*  
 3149 defined in the System Interfaces volume of IEEE Std 1003.1-200x).

3150 The POSIX\_TRACE\_FILTER trace event indicates that a trace event type filter value changed  
 3151 while the trace stream was running.

3152 The POSIX\_TRACE\_ERROR serves to inform the analyzer process that an implementation-  
 3153 defined internal error of the trace system occurred.

3154 The POSIX\_TRACE\_OVERFLOW trace event shall be reported with a timestamp equal to the  
 3155 timestamp of the first trace event overwritten. This is an indication that some generated trace  
 3156 events have been lost.

3157 The POSIX\_TRACE\_RESUME trace event shall be reported with a timestamp equal to the  
 3158 timestamp of the first valid trace event reported after the overflow condition ends and shall be  
 3159 reported before this first valid trace event. This is an indication that the trace system is reliably  
 3160 recording trace events after an overflow condition.

3161 Each of these trace event types shall be defined by a constant trace event name and a  
 3162 **trace\_event\_id\_t** constant; trace event data is associated with some of these trace events.

3163 If the Trace option is supported and the Trace Event Filter option and the Trace Log option are  
 3164 not supported, the following predefined system trace events in Table 2-3 shall be defined:

3165 **Table 2-3** Trace Option: System Trace Events

Event Name	Constant	Associated Data
		Data Type
posix_trace_error	POSIX_TRACE_ERROR	error int
posix_trace_start	POSIX_TRACE_START	None.
posix_trace_stop	POSIX_TRACE_STOP	auto int
posix_trace_overflow	POSIX_TRACE_OVERFLOW	None.
posix_trace_resume	POSIX_TRACE_RESUME	None.

3175 If the Trace option and the Trace Event Filter option are both supported, and if the Trace Log  
 3176 option is not supported, the following predefined system trace events in Table 2-4 (on page 528)  
 3177 shall be defined:

3178

**Table 2-4** Trace and Trace Event Filter Options: System Trace Events

3179

3180

3181

3182

3183

3184

3185

3186

3187

3188

3189

3190

3191

Event Name	Constant	Associated Data
		Data Type
posix_trace_error	POSIX_TRACE_ERROR	error
		int
posix_trace_start	POSIX_TRACE_START	event_filter
		trace_event_set_t
posix_trace_stop	POSIX_TRACE_STOP	auto
		int
posix_trace_filter	POSIX_TRACE_FILTER	old_event_filter
		new_event_filter
		trace_event_set_t
posix_trace_overflow	POSIX_TRACE_OVERFLOW	None.
posix_trace_resume	POSIX_TRACE_RESUME	None.

3192

3193

3194

If the Trace option and the Trace Log option are both supported, and if the Trace Event Filter option is not supported, the following predefined system trace events in Table 2-5 shall be defined:

3195

**Table 2-5** Trace and Trace Log Options: System Trace Events

3196

3197

3198

3199

3200

3201

3202

3203

3204

3205

3206

3207

Event Name	Constant	Associated Data
		Data Type
posix_trace_error	POSIX_TRACE_ERROR	error
		int
posix_trace_start	POSIX_TRACE_START	None.
posix_trace_stop	POSIX_TRACE_STOP	auto
		int
posix_trace_overflow	POSIX_TRACE_OVERFLOW	None.
posix_trace_resume	POSIX_TRACE_RESUME	None.
posix_trace_flush_start	POSIX_TRACE_FLUSH_START	None.
posix_trace_flush_stop	POSIX_TRACE_FLUSH_STOP	None.

3208

3209

If the Trace option, the Trace Event Filter option, and the Trace Log option are all supported, the following predefined system trace events in Table 2-6 (on page 529) shall be defined:

3210 **Table 2-6** Trace, Trace Log, and Trace Event Filter Options: System Trace Events

Event Name	Constant	Associated Data
		Data Type
posix_trace_error	POSIX_TRACE_ERROR	error
		int
posix_trace_start	POSIX_TRACE_START	event_filter
		trace_event_set_t
posix_trace_stop	POSIX_TRACE_STOP	auto
		int
posix_trace_filter	POSIX_TRACE_FILTER	old_event_filter
		new_event_filter
		trace_event_set_t
posix_trace_overflow	POSIX_TRACE_OVERFLOW	None.
posix_trace_resume	POSIX_TRACE_RESUME	None.
posix_trace_flush_start	POSIX_TRACE_FLUSH_START	None.
posix_trace_flush_stop	POSIX_TRACE_FLUSH_STOP	None.

3226 **2.11.2.2** User Trace Event Type Definitions

3227 The user trace event `POSIX_TRACE_UNNAMED_USEREVENT` is defined in the `<trace.h>`  
 3228 header. If the limit of per-process user trace event names represented by  
 3229 `{TRACE_USER_EVENT_MAX}` has already been reached, this predefined user event shall be  
 3230 returned when the application tries to register more events than allowed. The data associated  
 3231 with this trace event is application-defined.

3232 The following predefined user trace event in Table 2-7 shall be defined:

3233 **Table 2-7** Trace Option: User Trace Event

Event Name	Constant
posix_trace_unnamed_userevent	POSIX_TRACE_UNNAMED_USEREVENT

3236 **2.11.3** Trace Functions

3237 The trace interface is built and structured to improve portability through use of trace data of  
 3238 opaque type. The object-oriented approach for the manipulation of trace attributes and trace  
 3239 event type identifiers requires definition of many constructor and selector functions which  
 3240 operate on these opaque types. Also, the trace interface must support several different tracing  
 3241 roles. To facilitate reading the trace interface, the trace functions are grouped into small  
 3242 functional sets supporting the three different roles:

- 3243 • A trace controller process requires functions to set up and customize all the resources needed  
 3244 to run a trace stream, including:
  - 3245 — Attribute initialization and destruction (`posix_trace_attr_init()`)
  - 3246 — Identification information manipulation (`posix_trace_attr_getgenversion()`)
  - 3247 — Trace system behavior modification (`posix_trace_attr_getinherited()`)
  - 3248 — Trace stream and trace log size set (`posix_trace_attr_getmaxusereventsize()`)

- 3249 — Trace stream creation, flush, and shutdown (*posix\_trace\_create()*)
- 3250 — Trace stream and trace log clear (*posix\_trace\_clear()*)
- 3251 — Trace event type identifier manipulation (*posix\_trace\_trid\_eventid\_open()*)
- 3252 — Trace event type identifier list exploration (*posix\_trace\_eventtypelist\_getnext\_id()*)
- 3253 — Trace event type set manipulation (*posix\_trace\_eventset\_empty()*)
- 3254 — Trace event type filter set (*posix\_trace\_set\_filter()*)
- 3255 — Trace stream start and stop (*posix\_trace\_start()*)
- 3256 — Trace stream information and status read (*posix\_trace\_get\_attr()*)
- 3257 • A traced process requires functions to instrument trace points:
- 3258 — Trace event type identifiers definition and trace points insertion (*posix\_trace\_event()*)
- 3259 • A trace analyzer process requires functions to retrieve information from a trace stream and
- 3260 trace log:
- 3261 — Identification information read (*posix\_trace\_attr\_getgenversion()*)
- 3262 — Trace system behavior information read (*posix\_trace\_attr\_getinherited()*)
- 3263 — Trace stream and trace log size get (*posix\_trace\_attr\_getmaxusereventsized()*)
- 3264 — Trace event type identifier manipulation (*posix\_trace\_trid\_eventid\_open()*)
- 3265 — Trace event type identifier list exploration (*posix\_trace\_eventtypelist\_getnext\_id()*)
- 3266 — Trace log open, rewind, and close (*posix\_trace\_open()*)
- 3267 — Trace stream information and status read (*posix\_trace\_get\_attr()*)
- 3268 — trace event read (*posix\_trace\_getnext\_event()*)

## 3269 2.12 Data Types

3270 All of the data types used by various functions are defined by the implementation. The  
 3271 following table describes some of these types. Other types referenced in the description of a  
 3272 function, not mentioned here, can be found in the appropriate header for that function.

3273

3274

3275

3276

3277

3278

3279

3280

3281

3282

3283

3284

Defined Type	Description
<b>cc_t</b>	Type used for terminal special characters.
<b>clock_t</b>	Arithmetic type used for processor times, as defined in the ISO C standard.
<b>clockid_t</b>	Used for clock ID type in some timer functions.
<b>dev_t</b>	Arithmetic type used for device numbers.
<b>DIR</b>	Type representing a directory stream.
<b>div_t</b>	Structure type returned by the <i>div()</i> function.
<b>FILE</b>	Structure containing information about a file.
<b>glob_t</b>	Structure type used in pathname pattern matching.
<b>fpos_t</b>	Type containing all information needed to specify uniquely every

3285	Defined Type	Description
3286		
3287		position within a file.
3288	<b>gid_t</b>	Arithmetic type used for group IDs.
3289	<b>iconv_t</b>	Type used for conversion descriptors.
3290	<b>id_t</b>	Arithmetic type used as a general identifier; can be used to contain at least the largest of a <b>pid_t</b> , <b>uid_t</b> , or <b>gid_t</b> .
3291		
3292	<b>ino_t</b>	Arithmetic type used for file serial numbers.
3293	<b>key_t</b>	Arithmetic type used for XSI interprocess communication.
3294	<b>ldiv_t</b>	Structure type returned by the <i>ldiv()</i> function.
3295	<b>mode_t</b>	Arithmetic type used for file attributes.
3296	<b>mqd_t</b>	Used for message queue descriptors.
3297	<b>nfds_t</b>	Integer type used for the number of file descriptors.
3298	<b>nlink_t</b>	Arithmetic type used for link counts.
3299	<b>off_t</b>	Signed arithmetic type used for file sizes.
3300	<b>pid_t</b>	Signed arithmetic type used for process and process group IDs.
3301	<b>pthread_attr_t</b>	Used to identify a thread attribute object.
3302	<b>pthread_cond_t</b>	Used for condition variables.
3303	<b>pthread_condattr_t</b>	Used to identify a condition attribute object.
3304	<b>pthread_key_t</b>	Used for thread-specific data keys.
3305	<b>pthread_mutex_t</b>	Used for mutexes.
3306	<b>pthread_mutexattr_t</b>	Used to identify a mutex attribute object.
3307	<b>pthread_once_t</b>	Used for dynamic package initialization.
3308	<b>pthread_rwlock_t</b>	Used for read-write locks.
3309	<b>pthread_rwlockattr_t</b>	Used for read-write lock attributes.
3310	<b>pthread_t</b>	Used to identify a thread.
3311	<b>ptrdiff_t</b>	Signed integer type of the result of subtracting two pointers.
3312	<b>regex_t</b>	Structure type used in regular expression matching.
3313	<b>regmatch_t</b>	Structure type used in regular expression matching.
3314	<b>rlim_t</b>	Unsigned arithmetic type used for limit values, to which objects of type <b>int</b> and <b>off_t</b> can be cast without loss of value.
3315		
3316	<b>sem_t</b>	Type used in performing semaphore operations.
3317	<b>sig_atomic_t</b>	Integer type of an object that can be accessed as an atomic entity, even in the presence of asynchronous interrupts.
3318		
3319	<b>sigset_t</b>	Integer or structure type of an object used to represent sets of signals.
3320		
3321	<b>size_t</b>	Unsigned integer type used for size of objects.
3322	<b>speed_t</b>	Type used for terminal baud rates.
3323	<b>ssize_t</b>	Arithmetic type used for a count of bytes or an error indication.
3324	<b>suseconds_t</b>	Signed arithmetic type used for time in microseconds.
3325	<b>tcflag_t</b>	Type used for terminal modes.
3326	<b>time_t</b>	Arithmetic type used for time in seconds, as defined in the ISO C standard.
3327		
3328	<b>timer_t</b>	Used for timer ID returned by the <i>timer_create()</i> function.
3329	<b>uid_t</b>	Arithmetic type used for user IDs.
3330	<b>useconds_t</b>	Integer type used for time in microseconds.
3331	<b>va_list</b>	Type used for traversing variable argument lists.
3332	<b>wchar_t</b>	Integer type whose range of values can represent distinct codes for all members of the largest extended character set specified by the
3333		

3334  
3335  
3336  
3337  
3338  
3339  
3340  
3341

Defined Type	Description
<b>wctype_t</b>	supported locales.
<b>wint_t</b>	Scalar type which represents a character class descriptor.
	Integer type capable of storing any valid value of <b>wchar_t</b> or WEOF.
<b>wordexp_t</b>	Structure type used in word expansion.

# *System Interfaces*

3342

3343  
3344

This chapter describes the functions, macros, and external variables to support applications portability at the C-language source level.

3345 **NAME**

3346       FD\_CLR — macros for synchronous I/O multiplexing

3347 **SYNOPSIS**

3348       #include &lt;sys/time.h&gt;

3349       FD\_CLR(int *fd*, fd\_set \**fdset*);3350       FD\_ISSET(int *fd*, fd\_set \**fdset*);3351       FD\_SET(int *fd*, fd\_set \**fdset*);3352       FD\_ZERO(fd\_set \**fdset*);3353 **DESCRIPTION**3354       Refer to *pselect()*. |

3355 **NAME**

3356        \_Exit, \_exit — terminate a process

3357 **SYNOPSIS**

3358        #include &lt;unistd.h&gt;

3359        void \_Exit(int *status*);3360        void \_exit(int *status*);3361 **DESCRIPTION**3362        Refer to *exit()*.

3363 **NAME**

3364        \_longjmp, \_setjmp — non-local goto

3365 **SYNOPSIS**

3366 XSI       #include <setjmp.h>

3367       void \_longjmp(jmp\_buf env, int val);

3368       int \_setjmp(jmp\_buf env);

3369

3370 **DESCRIPTION**

3371       The *\_longjmp()* and *\_setjmp()* functions shall be equivalent to *longjmp()* and *setjmp()*,  
3372       respectively, with the additional restriction that *\_longjmp()* and *\_setjmp()* shall not manipulate  
3373       the signal mask.

3374       If *\_longjmp()* is called even though *env* was never initialized by a call to *\_setjmp()*, or when the  
3375       last such call was in a function that has since returned, the results are undefined.

3376 **RETURN VALUE**

3377       Refer to *longjmp()* and *setjmp()*.

3378 **ERRORS**

3379       No errors are defined.

3380 **EXAMPLES**

3381       None.

3382 **APPLICATION USAGE**

3383       If *\_longjmp()* is executed and the environment in which *\_setjmp()* was executed no longer exists,  
3384       errors can occur. The conditions under which the environment of the *\_setjmp()* no longer exists  
3385       include exiting the function that contains the *\_setjmp()* call, and exiting an inner block with  
3386       temporary storage. This condition might not be detectable, in which case the *\_longjmp()* occurs  
3387       and, if the environment no longer exists, the contents of the temporary storage of an inner block  
3388       are unpredictable. This condition might also cause unexpected process termination. If the  
3389       function has returned, the results are undefined.

3390       Passing *longjmp()* a pointer to a buffer not created by *setjmp()*, passing *\_longjmp()* a pointer to a  
3391       buffer not created by *\_setjmp()*, passing *siglongjmp()* a pointer to a buffer not created by  
3392       *sigsetjmp()*, or passing any of these three functions a buffer that has been modified by the user  
3393       can cause all the problems listed above, and more.

3394       The *\_longjmp()* and *\_setjmp()* functions are included to support programs written to historical  
3395       system interfaces. New applications should use *siglongjmp()* and *sigsetjmp()* respectively.

3396 **RATIONALE**

3397       None.

3398 **FUTURE DIRECTIONS**

3399       The *\_longjmp()* and *\_setjmp()* functions may be marked LEGACY in a future version.

3400 **SEE ALSO**

3401       *longjmp()*, *setjmp()*, *siglongjmp()*, *sigsetjmp()*, the Base Definitions volume of  
3402       IEEE Std 1003.1-200x, <setjmp.h>

3403 **CHANGE HISTORY**

3404       First released in Issue 4, Version 2.

3405 **Issue 5**

3406 Moved from X/OPEN UNIX extension to BASE.

3407 **NAME**

3408 `_setjmp` — set jump point for a non-local goto

3409 **SYNOPSIS**

3410 xSI `#include <setjmp.h>`

3411 `int _setjmp(jmp_buf env);`

3412

3413 **DESCRIPTION**

3414 Refer to `_longjmp()`.

3415 **NAME**

3416        \_toupper — transliterate uppercase characters to lowercase

3417 **SYNOPSIS**

3418 xSI        #include &lt;ctype.h&gt;

3419        int \_tolower(int c);

3420

3421 **DESCRIPTION**3422        The *\_tolower()* macro shall be equivalent to *tolower(c)* except that the application shall ensure  
3423        that the argument *c* is an uppercase letter.3424 **RETURN VALUE**3425        Upon successful completion, *\_tolower()* shall return the lowercase letter corresponding to the  
3426        argument passed.3427 **ERRORS**

3428        No errors are defined.

3429 **EXAMPLES**

3430        None.

3431 **APPLICATION USAGE**

3432        None.

3433 **RATIONALE**

3434        None.

3435 **FUTURE DIRECTIONS**

3436        None.

3437 **SEE ALSO**3438        *tolower()*, *isupper()*, the Base Definitions volume of IEEE Std 1003.1-200x, <ctype.h>, the Base  
3439        Definitions volume of IEEE Std 1003.1-200x, Chapter 7, Locale3440 **CHANGE HISTORY**

3441        First released in Issue 1. Derived from Issue 1 of the SVID.

3442 **Issue 6**

3443        The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

3444 **NAME**

3445        \_toupper — transliterate lowercase characters to uppercase

3446 **SYNOPSIS**

```
3447 xSI     #include <ctype.h>
```

```
3448        int _toupper(int c);
```

3449

3450 **DESCRIPTION**

3451        The *\_toupper()* macro shall be equivalent to *toupper()* except that the application shall ensure  
3452        that the argument *c* is a lowercase letter.

3453 **RETURN VALUE**

3454        Upon successful completion, *\_toupper()* shall return the uppercase letter corresponding to the  
3455        argument passed.

3456 **ERRORS**

3457        No errors are defined.

3458 **EXAMPLES**

3459        None.

3460 **APPLICATION USAGE**

3461        None.

3462 **RATIONALE**

3463        None.

3464 **FUTURE DIRECTIONS**

3465        None.

3466 **SEE ALSO**

3467        *islower()*, *toupper()*, the Base Definitions volume of IEEE Std 1003.1-200x, **<ctype.h>**, the Base  
3468        Definitions volume of IEEE Std 1003.1-200x, Chapter 7, Locale

3469 **CHANGE HISTORY**

3470        First released in Issue 1. Derived from Issue 1 of the SVID.

3471 **Issue 6**

3472        The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

3473 **NAME**

3474 a64l, l64a — convert between a 32-bit integer and a radix-64 ASCII string

3475 **SYNOPSIS**

```
3476 xSI #include <stdlib.h>
3477 long a64l(const char *s);
3478 char *l64a(long value);
3479
```

3480 **DESCRIPTION**

3481 These functions maintain numbers stored in radix-64 ASCII characters. This is a notation by  
 3482 which 32-bit integers can be represented by up to six characters; each character represents a digit  
 3483 in radix-64 notation. If the type **long** contains more than 32 bits, only the low-order 32 bits shall  
 3484 be used for these operations.

3485 The characters used to represent digits are '.' (dot) for 0, '/' for 1, '0' through '9' for [2,11],  
 3486 'A' through 'Z' for [12,37], and 'a' through 'z' for [38,63].

3487 The *a64l()* function shall take a pointer to a radix-64 representation, in which the first digit is the  
 3488 least significant, and return the corresponding **long** value. If the string pointed to by *s* contains  
 3489 more than six characters, *a64l()* shall use the first six. If the first six characters of the string  
 3490 contain a null terminator, *a64l()* shall use only characters preceding the null terminator. The  
 3491 *a64l()* function shall scan the character string from left to right with the least significant digit on  
 3492 the left, decoding each character as a 6-bit radix-64 number. If the type **long** contains more than  
 3493 32 bits, the resulting value is sign-extended. The behavior of *a64l()* is unspecified if *s* is a null  
 3494 pointer or the string pointed to by *s* was not generated by a previous call to *l64a()*.

3495 The *l64a()* function shall take a **long** argument and return a pointer to the corresponding radix-  
 3496 64 representation. The behavior of *l64a()* is unspecified if *value* is negative.

3497 The value returned by *l64a()* may be a pointer into a static buffer. Subsequent calls to *l64a()* may  
 3498 overwrite the buffer.

3499 The *l64a()* function need not be reentrant. A function that is not required to be reentrant is not  
 3500 required to be thread-safe.

3501 **RETURN VALUE**

3502 Upon successful completion, *a64l()* shall return the **long** value resulting from conversion of the  
 3503 input string. If a string pointed to by *s* is an empty string, *a64l()* shall return 0L.

3504 The *l64a()* function shall return a pointer to the radix-64 representation. If *value* is 0L, *l64a()* shall  
 3505 return a pointer to an empty string.

3506 **ERRORS**

3507 No errors are defined.

3508 **EXAMPLES**

3509 None.

3510 **APPLICATION USAGE**

3511 If the type **long** contains more than 32 bits, the result of *a64l(l64a(x))* is *x* in the low-order 32 bits.

3512 **RATIONALE**

3513 This is not the same encoding as used by either encoding variant of the *uuencode* utility.

3514 **FUTURE DIRECTIONS**

3515 None.

3516 **SEE ALSO**3517 *strtoul()*, the Base Definitions volume of IEEE Std 1003.1-200x, `<stdlib.h>`, the Shell and Utilities  
3518 volume of IEEE Std 1003.1-200x, *uuencode*3519 **CHANGE HISTORY**

3520 First released in Issue 4, Version 2.

3521 **Issue 5**

3522 Moved from X/OPEN UNIX extension to BASE.

3523 Normative text previously in the APPLICATION USAGE section moved to the DESCRIPTION.

3524 A note indicating that these functions need not be reentrant is added to the DESCRIPTION.

3525 **NAME**

3526 abort — generate an abnormal process abort

3527 **SYNOPSIS**

3528 #include &lt;stdlib.h&gt;

3529 void abort(void);

3530 **DESCRIPTION**

3531 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
3532 conflict between the requirements described here and the ISO C standard is unintentional. This  
3533 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

3534 The *abort()* function shall cause abnormal process termination to occur, unless the signal  
3535 SIGABRT is being caught and the signal handler does not return.

3536 CX The abnormal termination processing shall include at least the effect of *fclose()* on all open  
3537 streams and the default actions defined for SIGABRT.

3538 XSI On XSI-conformant systems, in addition the abnormal termination processing shall include the  
3539 effect of *fclose()* on message catalog descriptors.

3540 The SIGABRT signal shall be sent to the calling process as if by means of *raise()* with the  
3541 argument SIGABRT.

3542 CX The status made available to *wait()* or *waitpid()* by *abort()* shall be that of a process terminated  
3543 by the SIGABRT signal. The *abort()* function shall override blocking or ignoring the SIGABRT  
3544 signal.

3545 **RETURN VALUE**3546 The *abort()* function shall not return.3547 **ERRORS**

3548 No errors are defined.

3549 **EXAMPLES**

3550 None.

3551 **APPLICATION USAGE**

3552 Catching the signal is intended to provide the application writer with a portable means to abort  
3553 processing, free from possible interference from any implementation-defined functions.

3554 **RATIONALE**

3555 None.

3556 **FUTURE DIRECTIONS**

3557 None.

3558 **SEE ALSO**

3559 *exit()*, *kill()*, *raise()*, *signal()*, *wait()*, *waitpid()*, the Base Definitions volume of  
3560 IEEE Std 1003.1-200x, <stdlib.h>

3561 **CHANGE HISTORY**

3562 First released in Issue 1. Derived from Issue 1 of the SVID.

3563 **Issue 6**

3564 Extensions beyond the ISO C standard are now marked.

3565 **NAME**

3566       abs — return an integer absolute value

3567 **SYNOPSIS**

3568       #include &lt;stdlib.h&gt;

3569       int abs(int i);

3570 **DESCRIPTION**

3571 cx     The functionality described on this reference page is aligned with the ISO C standard. Any  
3572     conflict between the requirements described here and the ISO C standard is unintentional. This  
3573     volume of IEEE Std 1003.1-200x defers to the ISO C standard.

3574     The *abs()* function shall compute the absolute value of its integer operand, *i*. If the result cannot  
3575     be represented, the behavior is undefined.

3576 **RETURN VALUE**3577       The *abs()* function shall return the absolute value of its integer operand.3578 **ERRORS**

3579       No errors are defined.

3580 **EXAMPLES**

3581       None.

3582 **APPLICATION USAGE**

3583       In two's-complement representation, the absolute value of the negative integer with largest  
3584       magnitude {INT\_MIN} might not be representable.

3585 **RATIONALE**

3586       None.

3587 **FUTURE DIRECTIONS**

3588       None.

3589 **SEE ALSO**3590       *fabs()*, *labs()*, the Base Definitions volume of IEEE Std 1003.1-200x, <stdlib.h>3591 **CHANGE HISTORY**

3592       First released in Issue 1. Derived from Issue 1 of the SVID.

3593 **Issue 6**

3594       Extensions beyond the ISO C standard are now marked.

3595 **NAME**

3596           accept — accept a new connection on a socket

3597 **SYNOPSIS**

3598           #include &lt;sys/socket.h&gt;

3599           int accept(int *socket*, struct sockaddr \*restrict *address*,  
3600                      socklen\_t \*restrict *address\_len*);3601 **DESCRIPTION**3602           The *accept()* function shall extract the first connection on the queue of pending connections,  
3603           create a new socket with the same socket type protocol and address family as the specified  
3604           socket, and allocate a new file descriptor for that socket.3605           The *accept()* function takes the following arguments:3606           *socket*               Specifies a socket that was created with *socket()*, has been bound to an address  
3607                                   with *bind()*, and has issued a successful call to *listen()*.3608           *address*             Either a null pointer, or a pointer to a **sockaddr** structure where the address of  
3609                                   the connecting socket shall be returned.3610           *address\_len*       Points to a **socklen\_t** structure which on input specifies the length of the  
3611                                   supplied **sockaddr** structure, and on output specifies the length of the stored  
3612                                   address.3613           If *address* is not a null pointer, the address of the peer for the accepted connection shall be stored  
3614           in the **sockaddr** structure pointed to by *address*, and the length of this address shall be stored in  
3615           the object pointed to by *address\_len*.3616           If the actual length of the address is greater than the length of the supplied **sockaddr** structure,  
3617           the stored address shall be truncated.3618           If the protocol permits connections by unbound clients, and the peer is not bound, then the value  
3619           stored in the object pointed to by *address* is unspecified.3620           If the listen queue is empty of connection requests and O\_NONBLOCK is not set on the file  
3621           descriptor for the socket, *accept()* shall block until a connection is present. If the *listen()* queue is  
3622           empty of connection requests and O\_NONBLOCK is set on the file descriptor for the socket,  
3623           *accept()* shall fail and set *errno* to [EAGAIN] or [EWOULDBLOCK].3624           The accepted socket cannot itself accept more connections. The original socket remains open and  
3625           can accept more connections.3626 **RETURN VALUE**3627           Upon successful completion, *accept()* shall return the non-negative file descriptor of the accepted  
3628           socket. Otherwise, -1 shall be returned and *errno* set to indicate the error.3629 **ERRORS**3630           The *accept()* function shall fail if:

3631           [EAGAIN] or [EWOULDBLOCK]

3632                                   O\_NONBLOCK is set for the socket file descriptor and no connections are  
3633                                   present to be accepted.3634           [EBADF]               The *socket* argument is not a valid file descriptor.

3635           [ECONNABORTED]

3636                                   A connection has been aborted.

3637	[EINTR]	The <i>accept()</i> function was interrupted by a signal that was caught before a valid connection arrived.
3638		
3639	[EINVAL]	The <i>socket</i> is not accepting connections.
3640	[EMFILE]	{OPEN_MAX} file descriptors are currently open in the calling process.
3641	[ENFILE]	The maximum number of file descriptors in the system are already open.
3642	[ENOTSOCK]	The <i>socket</i> argument does not refer to a socket.
3643	[EOPNOTSUPP]	The socket type of the specified socket does not support accepting connections.
3644		
3645		The <i>accept()</i> function may fail if:
3646	[ENOBUFS]	No buffer space is available.
3647	[ENOMEM]	There was insufficient memory available to complete the operation.
3648	XSR [EPROTO]	A protocol error has occurred; for example, the STREAMS protocol stack has not been initialized.
3649		

**3650 EXAMPLES**

3651 None.

**3652 APPLICATION USAGE**

3653 When a connection is available, *select()* indicates that the file descriptor for the socket is ready  
3654 for reading.

**3655 RATIONALE**

3656 None.

**3657 FUTURE DIRECTIONS**

3658 None.

**3659 SEE ALSO**

3660 *bind()*, *connect()*, *listen()*, *socket()*, the Base Definitions volume of IEEE Std 1003.1-200x,  
3661 <sys/socket.h>

**3662 CHANGE HISTORY**

3663 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

3664 The **restrict** keyword is added to the *accept()* prototype for alignment with the  
3665 ISO/IEC 9899:1999 standard.

3666 **NAME**

3667           access — determine accessibility of a file

3668 **SYNOPSIS**

3669           #include &lt;unistd.h&gt;

3670           int access(const char \*path, int amode);

3671 **DESCRIPTION**

3672           The `access()` function shall check the file named by the pathname pointed to by the `path` |  
 3673           argument for accessibility according to the bit pattern contained in `amode`, using the real user ID |  
 3674           in place of the effective user ID and the real group ID in place of the effective group ID.

3675           The value of `amode` is either the bitwise-inclusive OR of the access permissions to be checked |  
 3676           (R\_OK, W\_OK, X\_OK) or the existence test (F\_OK).

3677           If any access permissions are checked, each shall be checked individually, as described in the |  
 3678           Base Definitions volume of IEEE Std 1003.1-200x, Chapter 3, Definitions. If the process has |  
 3679           appropriate privileges, an implementation may indicate success for X\_OK even if none of the |  
 3680           execute file permission bits are set.

3681 **RETURN VALUE**

3682           If the requested access is permitted, `access()` succeeds and shall return 0; otherwise, -1 shall be |  
 3683           returned and `errno` shall be set to indicate the error.

3684 **ERRORS**3685           The `access()` function shall fail if:

3686           [EACCES]           Permission bits of the file mode do not permit the requested access, or search |  
 3687           permission is denied on a component of the path prefix.

3688           [ELOOP]           A loop exists in symbolic links encountered during resolution of the `path` |  
 3689           argument.

3690           [ENAMETOOLONG]

3691                               The length of the `path` argument exceeds {PATH\_MAX} or a pathname |  
 3692                               component is longer than {NAME\_MAX}. |

3693           [ENOENT]           A component of `path` does not name an existing file or `path` is an empty string.

3694           [ENOTDIR]          A component of the path prefix is not a directory.

3695           [EROFS]           Write access is requested for a file on a read-only file system.

3696           The `access()` function may fail if:

3697           [EINVAL]           The value of the `amode` argument is invalid.

3698           [ELOOP]           More than {SYMLOOP\_MAX} symbolic links were encountered during |  
 3699           resolution of the `path` argument.

3700           [ENAMETOOLONG]

3701                               As a result of encountering a symbolic link in resolution of the `path` argument, |  
 3702                               the length of the substituted pathname string exceeded {PATH\_MAX}. |

3703           [ETXTBSY]          Write access is requested for a pure procedure (shared text) file that is being |  
 3704           executed.

3705 **EXAMPLES**3706 **Testing for the Existence of a File**

3707 The following example tests whether a file named **myfile** exists in the **/tmp** directory.

```
3708 #include <unistd.h>
3709 ...
3710 int result;
3711 const char *filename = "/tmp/myfile";
3712 result = access (filename, F_OK);
```

3713 **APPLICATION USAGE**

3714 Additional values of *amode* other than the set defined in the description may be valid; for  
3715 example, if a system has extended access controls.

3716 **RATIONALE**

3717 In early proposals, some inadequacies in the *access()* function led to the creation of an *eaccess()*  
3718 function because:

- 3719 1. Historical implementations of *access()* do not test file access correctly when the process'  
3720 real user ID is superuser. In particular, they always return zero when testing execute  
3721 permissions without regard to whether the file is executable.
- 3722 2. The superuser has complete access to all files on a system. As a consequence, programs  
3723 started by the superuser and switched to the effective user ID with lesser privileges cannot  
3724 use *access()* to test their file access permissions.

3725 However, the historical model of *eaccess()* does not resolve problem (1), so this volume of  
3726 IEEE Std 1003.1-200x now allows *access()* to behave in the desired way because several  
3727 implementations have corrected the problem. It was also argued that problem (2) is more easily  
3728 solved by using *open()*, *chdir()*, or one of the *exec* functions as appropriate and responding to the  
3729 error, rather than creating a new function that would not be as reliable. Therefore, *eaccess()* is not  
3730 included in this volume of IEEE Std 1003.1-200x.

3731 The sentence concerning appropriate privileges and execute permission bits reflects the two  
3732 possibilities implemented by historical implementations when checking superuser access for  
3733 *X\_OK*.

3734 New implementations are discouraged from returning *X\_OK* unless at least one execution  
3735 permission bit is set.

3736 **FUTURE DIRECTIONS**

3737 None.

3738 **SEE ALSO**

3739 *chmod()*, *stat()*, the Base Definitions volume of IEEE Std 1003.1-200x, **<unistd.h>**

3740 **CHANGE HISTORY**

3741 First released in Issue 1. Derived from Issue 1 of the SVID.

3742 **Issue 6**

3743 The following new requirements on POSIX implementations derive from alignment with the  
3744 Single UNIX Specification:

- 3745 • The [ELOOP] mandatory error condition is added.
- 3746 • A second [ENAMETOOLONG] is added as an optional error condition.

3747

- The [ETXTBSY] optional error condition is added.

3748

The following changes were made to align with the IEEE P1003.1a draft standard:

3749

- The [ELOOP] optional error condition is added.

3750 **NAME**

3751           acos, acosf, acosl — arc cosine functions

3752 **SYNOPSIS**

3753           #include &lt;math.h&gt;

3754           double acos(double x);

3755           float acosf(float x);

3756           long double acosl(long double x);

3757 **DESCRIPTION**

3758 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
 3759 conflict between the requirements described here and the ISO C standard is unintentional. This  
 3760 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

3761       These functions shall compute the principal value of the arc cosine of their argument *x*. The  
 3762 value of *x* should be in the range  $[-1,1]$ .

3763       An application wishing to check for error situations should set *errno* to zero and call  
 3764 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 3765 *fetetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 3766 zero, an error has occurred.

3767 **RETURN VALUE**

3768       Upon successful completion, these functions shall return the arc cosine of *x*, in the range  $[0,\pi]$   
 3769 radians.

3770 **MX**       For finite values of *x* not in the range  $[-1,1]$ , a domain error shall occur, and either a NaN (if  
 3771 supported), or an implementation-defined value shall be returned.

3772 **MX**       If *x* is NaN, a NaN shall be returned.

3773       If *x* is +1, +0 shall be returned.

3774       If *x* is  $\pm\text{Inf}$ , a domain error shall occur, and either a NaN (if supported), or an implementation-  
 3775 defined value shall be returned.

3776 **ERRORS**

3777       These functions shall fail if:

3778 **MX**       Domain Error    The *x* argument is finite and is not in the range  $[-1,1]$ , or is  $\pm\text{Inf}$ .

3779               If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero, |  
 3780 then *errno* shall be set to [EDOM]. If the integer expression (math\_errhandling |  
 3781 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception |  
 3782 shall be raised. |

3783 **EXAMPLES**

3784       None.

3785 **APPLICATION USAGE**

3786       On error, the expressions (math\_errhandling & MATH\_ERRNO) and (math\_errhandling &  
 3787 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

3788 **RATIONALE**

3789       None.

3790 **FUTURE DIRECTIONS**

3791 None.

3792 **SEE ALSO**

3793 *cos()*, *feclearexcept()*, *fetetestexcept()*, *isnan()*, the Base Definitions volume of IEEE Std 1003.1-200x, |  
3794 Section 4.18, Treatment of Error Conditions for Mathematical Functions, <math.h> |

3795 **CHANGE HISTORY**

3796 First released in Issue 1. Derived from Issue 1 of the SVID.

3797 **Issue 5**

3798 The DESCRIPTION is updated to indicate how an application should check for an error. This  
3799 text was previously published in the APPLICATION USAGE section.

3800 **Issue 6**3801 The *acosf()* and *acosl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

3802 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
3803 revised to align with the ISO/IEC 9899:1999 standard.

3804 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
3805 marked.

3806 **NAME**3807        **acosf** — arc cosine functions3808 **SYNOPSIS**

3809        #include &lt;math.h&gt;

3810        float acosf(float x);

3811 **DESCRIPTION**3812        Refer to *acos()*.

3813 **NAME**

3814 acosh, acoshf, acoshl, — inverse hyperbolic cosine functions

3815 **SYNOPSIS**

3816 #include &lt;math.h&gt;

3817 double acosh(double x);

3818 float acoshf(float x);

3819 long double acoshl(long double x);

3820 **DESCRIPTION**

3821 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 3822 conflict between the requirements described here and the ISO C standard is unintentional. This  
 3823 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

3824 These functions shall compute the inverse hyperbolic cosine of their argument *x*.

3825 An application wishing to check for error situations should set *errno* to zero and call  
 3826 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 3827 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 3828 zero, an error has occurred.

3829 **RETURN VALUE**3830 Upon successful completion, these functions shall return the inverse hyperbolic cosine of their  
3831 argument.

3832 **MX** For finite values of  $x < 1$ , a domain error shall occur, and either a NaN (if supported), or an  
 3833 implementation-defined value shall be returned.

3834 **MX** If *x* is NaN, a NaN shall be returned.3835 If *x* is +1, +0 shall be returned.3836 If *x* is +Inf, +Inf shall be returned.

3837 If *x* is -Inf, a domain error shall occur, and either a NaN (if supported), or an implementation-  
 3838 defined value shall be returned.

3839 **ERRORS**

3840 These functions shall fail if:

3841 **MX** Domain Error The *x* argument is finite and less than +1.0, or is -Inf.

3842 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero, |  
 3843 then *errno* shall be set to [EDOM]. If the integer expression (math\_errhandling |  
 3844 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception |  
 3845 shall be raised. |

3846 **EXAMPLES**

3847 None.

3848 **APPLICATION USAGE**

3849 On error, the expressions (math\_errhandling & MATH\_ERRNO) and (math\_errhandling &  
 3850 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

3851 **RATIONALE**

3852 None.

3853 **FUTURE DIRECTIONS**

3854 None.

3855 **SEE ALSO**

3856 *cosh()*, *feclearexcept()*, *fetetestexcept()*, the Base Definitions volume of IEEE Std 1003.1-200x, Section |  
3857 4.18, Treatment of Error Conditions for Mathematical Functions, <math.h> |

3858 **CHANGE HISTORY**

3859 First released in Issue 4, Version 2.

3860 **Issue 5**

3861 Moved from X/OPEN UNIX extension to BASE.

3862 **Issue 6**3863 The *acosh()* function is no longer marked as an extension.

3864 The *acoshf()*, and *acoshl()* functions are added for alignment with the ISO/IEC 9899:1999  
3865 standard.

3866 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
3867 revised to align with the ISO/IEC 9899:1999 standard.

3868 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
3869 marked.

3870 **NAME**

3871        *acosl* — arc cosine functions

3872 **SYNOPSIS**

3873        #include <math.h>

3874        long double *acosl*(long double *x*);

3875 **DESCRIPTION**

3876        Refer to *acos*().

3877 **NAME**3878 aio\_cancel — cancel an asynchronous I/O request (**REALTIME**)3879 **SYNOPSIS**

3880 AIO #include &lt;aio.h&gt;

3881 int aio\_cancel(int *fildev*, struct aiocb \**aiocbp*);

3882

3883 **DESCRIPTION**

3884 The *aio\_cancel()* function shall attempt to cancel one or more asynchronous I/O requests  
3885 currently outstanding against file descriptor *fildev*. The *aiocbp* argument points to the  
3886 asynchronous I/O control block for a particular request to be canceled. If *aiocbp* is NULL, then  
3887 all outstanding cancelable asynchronous I/O requests against *fildev* shall be canceled.

3888 Normal asynchronous notification shall occur for asynchronous I/O operations that are  
3889 successfully canceled. If there are requests that cannot be canceled, then the normal  
3890 asynchronous completion process shall take place for those requests when they are completed.

3891 For requested operations that are successfully canceled, the associated error status shall be set to  
3892 [ECANCELED] and the return status shall be -1. For requested operations that are not  
3893 successfully canceled, the *aiocbp* shall not be modified by *aio\_cancel()*.

3894 If *aiocbp* is not NULL, then if *fildev* does not have the same value as the file descriptor with which  
3895 the asynchronous operation was initiated, unspecified results occur.

3896 Which operations are cancelable is implementation-defined.

3897 **RETURN VALUE**

3898 The *aio\_cancel()* function shall return the value AIO\_CANCELED to the calling process if the  
3899 requested operation(s) were canceled. The value AIO\_NOTCANCELED shall be returned if at  
3900 least one of the requested operation(s) cannot be canceled because it is in progress. In this case,  
3901 the state of the other operations, if any, referenced in the call to *aio\_cancel()* is not indicated by  
3902 the return value of *aio\_cancel()*. The application may determine the state of affairs for these  
3903 operations by using *aio\_error()*. The value AIO\_ALLDONE is returned if all of the operations  
3904 have already completed. Otherwise, the function shall return -1 and set *errno* to indicate the  
3905 error.

3906 **ERRORS**

3907 The *aio\_cancel()* function shall fail if:

3908 [EBADF] The *fildev* argument is not a valid file descriptor.

3909 **EXAMPLES**

3910 None.

3911 **APPLICATION USAGE**

3912 The *aio\_cancel()* function is part of the Asynchronous Input and Output option and need not be  
3913 available on all implementations.

3914 **RATIONALE**

3915 None.

3916 **FUTURE DIRECTIONS**

3917 None.

3918 **SEE ALSO**

3919 *aio\_read()*, *aio\_write()*, the Base Definitions volume of IEEE Std 1003.1-200x, <**aio.h**>

3920 **CHANGE HISTORY**

3921 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

3922 **Issue 6**

3923 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
3924 implementation does not support the Asynchronous Input and Output option.

3925 The APPLICATION USAGE section is added.

3926 **NAME**

3927 aio\_error — retrieve errors status for an asynchronous I/O operation (**REALTIME**)

3928 **SYNOPSIS**

3929 AIO #include <aio.h>

3930 int aio\_error(const struct aiocb \*aiocbp);

3931

3932 **DESCRIPTION**

3933 The *aio\_error()* function shall return the error status associated with the **aiocb** structure  
3934 referenced by the *aiocbp* argument. The error status for an asynchronous I/O operation is the  
3935 SIO *errno* value that would be set by the corresponding *read()*, *write()*, *fdatasync()*, or *fsync()*  
3936 operation. If the operation has not yet completed, then the error status shall be equal to  
3937 [EINPROGRESS].

3938 **RETURN VALUE**

3939 If the asynchronous I/O operation has completed successfully, then 0 shall be returned. If the  
3940 asynchronous operation has completed unsuccessfully, then the error status, as described for  
3941 SIO *read()*, *write()*, *fdatasync()*, and *fsync()*, shall be returned. If the asynchronous I/O operation has  
3942 not yet completed, then [EINPROGRESS] shall be returned.

3943 **ERRORS**

3944 The *aio\_error()* function may fail if:

3945 [EINVAL] The *aiocbp* argument does not refer to an asynchronous operation whose  
3946 return status has not yet been retrieved.

3947 **EXAMPLES**

3948 None.

3949 **APPLICATION USAGE**

3950 The *aio\_error()* function is part of the Asynchronous Input and Output option and need not be  
3951 available on all implementations.

3952 **RATIONALE**

3953 None.

3954 **FUTURE DIRECTIONS**

3955 None.

3956 **SEE ALSO**

3957 *aio\_cancel()*, *aio\_fsync()*, *aio\_read()*, *aio\_return()*, *aio\_write()*, *close()*, *exec*, *exit()*, *fork()*, *lio\_listio()*,  
3958 *lseek()*, *read()*, the Base Definitions volume of IEEE Std 1003.1-200x, <**aio.h**>

3959 **CHANGE HISTORY**

3960 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

3961 **Issue 6**

3962 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
3963 implementation does not support the Asynchronous Input and Output option.

3964 The APPLICATION USAGE section is added.

3965 **NAME**3966 aio\_fsync — asynchronous file synchronization (**REALTIME**)3967 **SYNOPSIS**

3968 AIO #include &lt;aio.h&gt;

3969 int aio\_fsync(int op, struct aiocb \*aiocbp);

3970

3971 **DESCRIPTION**

3972 The *aio\_fsync()* function shall asynchronously force all I/O operations associated with the file |  
 3973 indicated by the file descriptor *aio\_fildes* member of the **aiocb** structure referenced by the *aiocbp* |  
 3974 argument and queued at the time of the call to *aio\_fsync()* to the synchronized I/O completion  
 3975 state. The function call shall return when the synchronization request has been initiated or  
 3976 queued to the file or device (even when the data cannot be synchronized immediately).

3977 If *op* is **O\_DSYNC**, all currently queued I/O operations shall be completed as if by a call to  
 3978 *fdatasync()*; that is, as defined for synchronized I/O data integrity completion. If *op* is **O\_SYNC**,  
 3979 all currently queued I/O operations shall be completed as if by a call to *fsync()*; that is, as  
 3980 defined for synchronized I/O file integrity completion. If the *aio\_fsync()* function fails, or if the  
 3981 operation queued by *aio\_fsync()* fails, then, as for *fsync()* and *fdatasync()*, outstanding I/O  
 3982 operations are not guaranteed to have been completed.

3983 If *aio\_fsync()* succeeds, then it is only the I/O that was queued at the time of the call to  
 3984 *aio\_fsync()* that is guaranteed to be forced to the relevant completion state. The completion of  
 3985 subsequent I/O on the file descriptor is not guaranteed to be completed in a synchronized  
 3986 fashion.

3987 The *aiocbp* argument refers to an asynchronous I/O control block. The *aiocbp* value may be used |  
 3988 as an argument to *aio\_error()* and *aio\_return()* in order to determine the error status and return  
 3989 status, respectively, of the asynchronous operation while it is proceeding. When the request is  
 3990 queued, the error status for the operation is **[EINPROGRESS]**. When all data has been  
 3991 successfully transferred, the error status shall be reset to reflect the success or failure of the  
 3992 operation. If the operation does not complete successfully, the error status for the operation shall  
 3993 be set to indicate the error. The *aio\_sigevent* member determines the asynchronous notification to  
 3994 occur as specified in Section 2.4.1 (on page 478) when all operations have achieved synchronized  
 3995 I/O completion. All other members of the structure referenced by *aiocbp* are ignored. If the  
 3996 control block referenced by *aiocbp* becomes an illegal address prior to asynchronous I/O  
 3997 completion, then the behavior is undefined.

3998 If the *aio\_fsync()* function fails or the *aiocbp* indicates an error condition, data is not guaranteed  
 3999 to have been successfully transferred.

4000 **RETURN VALUE**

4001 The *aio\_fsync()* function shall return the value 0 to the calling process if the I/O operation is  
 4002 successfully queued; otherwise, the function shall return the value -1 and set *errno* to indicate  
 4003 the error.

4004 **ERRORS**4005 The *aio\_fsync()* function shall fail if:

4006 **[EAGAIN]** The requested asynchronous operation was not queued due to temporary  
 4007 resource limitations.

4008 **[EBADF]** The *aio\_fildes* member of the **aiocb** structure referenced by the *aiocbp* argument  
 4009 is not a valid file descriptor open for writing.

- 4010 [EINVAL] This implementation does not support synchronized I/O for this file.
- 4011 [EINVAL] A value of *op* other than O\_DSYNC or O\_SYNC was specified.
- 4012 In the event that any of the queued I/O operations fail, *aio\_fsync()* shall return the error  
4013 condition defined for *read()* and *write()*. The error is returned in the error status for the  
4014 asynchronous *fsync()* operation, which can be retrieved using *aio\_error()*.
- 4015 **EXAMPLES**
- 4016 None.
- 4017 **APPLICATION USAGE**
- 4018 The *aio\_fsync()* function is part of the Asynchronous Input and Output option and need not be  
4019 available on all implementations.
- 4020 **RATIONALE**
- 4021 None.
- 4022 **FUTURE DIRECTIONS**
- 4023 None.
- 4024 **SEE ALSO**
- 4025 *fcntl()*, *fdatasync()*, *fsync()*, *open()*, *read()*, *write()*, the Base Definitions volume of  
4026 IEEE Std 1003.1-200x, <**aio.h**>
- 4027 **CHANGE HISTORY**
- 4028 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.
- 4029 **Issue 6**
- 4030 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
4031 implementation does not support the Asynchronous Input and Output option.
- 4032 The APPLICATION USAGE section is added.

4033 **NAME**4034 aio\_read — asynchronous read from a file (**REALTIME**)4035 **SYNOPSIS**

4036 AIO #include &lt;aio.h&gt;

4037 int aio\_read(struct aiocb \*aiocbp);

4038

4039 **DESCRIPTION**

4040 The *aio\_read()* function shall read *aiocbp->aio\_nbytes* from the file associated with *aiocbp->aio\_fildes* into the buffer pointed to by *aiocbp->aio\_buf*. The function call shall return when the read request has been initiated or queued to the file or device (even when the data cannot be delivered immediately).

4044 PIO If prioritized I/O is supported for this file, then the asynchronous operation shall be submitted at a priority equal to the scheduling priority of the process minus *aiocbp->aio\_reqprio*.

4046 The *aiocbp* value may be used as an argument to *aio\_error()* and *aio\_return()* in order to determine the error status and return status, respectively, of the asynchronous operation while it is proceeding. If an error condition is encountered during queuing, the function call shall return without having initiated or queued the request. The requested operation takes place at the absolute position in the file as given by *aio\_offset*, as if *lseek()* were called immediately prior to the operation with an *offset* equal to *aio\_offset* and a *whence* equal to *SEEK\_SET*. After a successful call to enqueue an asynchronous I/O operation, the value of the file offset for the file is unspecified.

4054 The *aiocbp->aio\_lio\_opcode* field shall be ignored by *aio\_read()*.

4055 The *aiocbp* argument points to an **aiocb** structure. If the buffer pointed to by *aiocbp->aio\_buf* or the control block pointed to by *aiocbp* becomes an illegal address prior to asynchronous I/O completion, then the behavior is undefined.

4058 Simultaneous asynchronous operations using the same *aiocbp* produce undefined results.

4059 SIO If synchronized I/O is enabled on the file associated with *aiocbp->aio\_fildes*, the behavior of this function shall be according to the definitions of synchronized I/O data integrity completion and synchronized I/O file integrity completion.

4062 For any system action that changes the process memory space while an asynchronous I/O is outstanding to the address range being changed, the result of that action is undefined.

4064 For regular files, no data transfer shall occur past the offset maximum established in the open file description associated with *aiocbp->aio\_fildes*.

4066 **RETURN VALUE**

4067 The *aio\_read()* function shall return the value zero to the calling process if the I/O operation is successfully queued; otherwise, the function shall return the value  $-1$  and set *errno* to indicate the error.

4070 **ERRORS**

4071 The *aio\_read()* function shall fail if:

4072 [EAGAIN] The requested asynchronous I/O operation was not queued due to system resource limitations.

4074 Each of the following conditions may be detected synchronously at the time of the call to *aio\_read()*, or asynchronously. If any of the conditions below are detected synchronously, the *aio\_read()* function shall return  $-1$  and set *errno* to the corresponding value. If any of the conditions below are detected asynchronously, the return status of the asynchronous operation

- 4078 is set to `-1`, and the error status of the asynchronous operation is set to the corresponding value.
- 4079 [EBADF] The `aiocbp->aio_fildes` argument is not a valid file descriptor open for reading.
- 4080 [EINVAL] The file offset value implied by `aiocbp->aio_offset` would be invalid, `aiocbp->aio_reqprio` is not a valid value, or `aiocbp->aio_nbytes` is an invalid value.
- 4081
- 4082 In the case that the `aio_read()` successfully queues the I/O operation but the operation is subsequently canceled or encounters an error, the return status of the asynchronous operation is one of the values normally returned by the `read()` function call. In addition, the error status of the asynchronous operation is set to one of the error statuses normally set by the `read()` function call, or one of the following values:
- 4083
- 4084
- 4085
- 4086
- 4087 [EBADF] The `aiocbp->aio_fildes` argument is not a valid file descriptor open for reading.
- 4088 [ECANCELED] The requested I/O was canceled before the I/O completed due to an explicit `aio_cancel()` request.
- 4089
- 4090 [EINVAL] The file offset value implied by `aiocbp->aio_offset` would be invalid.
- 4091 The following condition may be detected synchronously or asynchronously:
- 4092 [EOVERFLOW] The file is a regular file, `aiocbp->aio_nbytes` is greater than 0, and the starting offset in `aiocbp->aio_offset` is before the end-of-file and is at or beyond the offset maximum in the open file description associated with `aiocbp->aio_fildes`.
- 4093
- 4094

#### 4095 EXAMPLES

4096 None.

#### 4097 APPLICATION USAGE

4098 The `aio_read()` function is part of the Asynchronous Input and Output option and need not be available on all implementations.

4099

#### 4100 RATIONALE

4101 None.

#### 4102 FUTURE DIRECTIONS

4103 None.

#### 4104 SEE ALSO

4105 `aio_cancel()`, `aio_error()`, `lio_listio()`, `aio_return()`, `aio_write()`, `close()`, `exec`, `exit()`, `fork()`, `lseek()`, `read()`, the Base Definitions volume of IEEE Std 1003.1-200x, `<aio.h>`

4106

#### 4107 CHANGE HISTORY

4108 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

4109 Large File Summit extensions are added.

#### 4110 Issue 6

4111 The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Asynchronous Input and Output option.

4112

4113 The APPLICATION USAGE section is added. |

- 4114 The following new requirements on POSIX implementations derive from alignment with the |  
4115 Single UNIX Specification:
- 4116 • In the DESCRIPTION, text is added to indicate setting of the offset maximum in the open file  
4117 description. This change is to support large files.
  - 4118 • In the ERRORS section, the [Eoverflow] condition is added. This change is to support  
4119 large files.

4120 **NAME**4121 aio\_return — retrieve return status of an asynchronous I/O operation (**REALTIME**)4122 **SYNOPSIS**

4123 AIO #include &lt;aio.h&gt;

4124 ssize\_t aio\_return(struct aiocb \*aiocbp);

4125

4126 **DESCRIPTION**

4127 The *aio\_return()* function shall return the return status associated with the **aiocb** structure  
4128 referenced by the *aiocbp* argument. The return status for an asynchronous I/O operation is the  
4129 value that would be returned by the corresponding *read()*, *write()*, or *fsync()* function call. If the  
4130 error status for the operation is equal to [EINPROGRESS], then the return status for the  
4131 operation is undefined. The *aio\_return()* function may be called exactly once to retrieve the  
4132 return status of a given asynchronous operation; thereafter, if the same **aiocb** structure is used in  
4133 a call to *aio\_return()* or *aio\_error()*, an error may be returned. When the **aiocb** structure referred  
4134 to by *aiocbp* is used to submit another asynchronous operation, then *aio\_return()* may be  
4135 successfully used to retrieve the return status of that operation.

4136 **RETURN VALUE**

4137 If the asynchronous I/O operation has completed, then the return status, as described for *read()*,  
4138 *write()*, and *fsync()*, shall be returned. If the asynchronous I/O operation has not yet completed,  
4139 the results of *aio\_return()* are undefined.

4140 **ERRORS**4141 The *aio\_return()* function may fail if:

4142 [EINVAL] The *aiocbp* argument does not refer to an asynchronous operation whose  
4143 return status has not yet been retrieved.

4144 **EXAMPLES**

4145 None.

4146 **APPLICATION USAGE**

4147 The *aio\_return()* function is part of the Asynchronous Input and Output option and need not be  
4148 available on all implementations.

4149 **RATIONALE**

4150 None.

4151 **FUTURE DIRECTIONS**

4152 None.

4153 **SEE ALSO**

4154 *aio\_cancel()*, *aio\_error()*, *aio\_fsync()*, *aio\_read()*, *aio\_write()*, *close()*, *exec*, *exit()*, *fork()*, *lio\_listio()*,  
4155 *lseek()*, *read()*, the Base Definitions volume of IEEE Std 1003.1-200x, <**aio.h**>

4156 **CHANGE HISTORY**

4157 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

4158 **Issue 6**

4159 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
4160 implementation does not support the Asynchronous Input and Output option.

4161 The APPLICATION USAGE section is added.

4162 The [EINVAL] error condition is updated as a “may fail”. This is for consistency with the  
4163 DESCRIPTION.

4164 **NAME**4165 aio\_suspend — wait for an asynchronous I/O request (**REALTIME**)4166 **SYNOPSIS**4167 AIO `#include <aio.h>`4168 `int aio_suspend(const struct aiocb * const list[], int nent,`  
4169 `const struct timespec *timeout);`

4170

4171 **DESCRIPTION**

4172 The *aio\_suspend()* function shall suspend the calling thread until at least one of the asynchronous  
 4173 I/O operations referenced by the *list* argument has completed, until a signal interrupts the  
 4174 function, or, if *timeout* is not NULL, until the time interval specified by *timeout* has passed. If any  
 4175 of the **aiocb** structures in the list correspond to completed asynchronous I/O operations (that is,  
 4176 the error status for the operation is not equal to [EINPROGRESS]) at the time of the call, the  
 4177 function shall return without suspending the calling thread. The *list* argument is an array of  
 4178 pointers to asynchronous I/O control blocks. The *nent* argument indicates the number of  
 4179 elements in the array. Each **aiocb** structure pointed to has been used in initiating an  
 4180 asynchronous I/O request via *aio\_read()*, *aio\_write()*, or *lio\_listio()*. This array may contain  
 4181 NULL pointers, which are ignored. If this array contains pointers that refer to **aiocb** structures  
 4182 that have not been used in submitting asynchronous I/O, the effect is undefined.

4183 If the time interval indicated in the **timespec** structure pointed to by *timeout* passes before any of  
 4184 the I/O operations referenced by *list* are completed, then *aio\_suspend()* shall return with an  
 4185 error. If the Monotonic Clock option is supported, the clock that shall be used to measure this  
 4186 time interval shall be the CLOCK\_MONOTONIC clock.

4187 **RETURN VALUE**

4188 If the *aio\_suspend()* function returns after one or more asynchronous I/O operations have  
 4189 completed, the function shall return zero. Otherwise, the function shall return a value of -1 and  
 4190 set *errno* to indicate the error.

4191 The application may determine which asynchronous I/O completed by scanning the associated  
 4192 error and return status using *aio\_error()* and *aio\_return()*, respectively.

4193 **ERRORS**4194 The *aio\_suspend()* function shall fail if:

4195 [EAGAIN] No asynchronous I/O indicated in the list referenced by *list* completed in the  
 4196 time interval indicated by *timeout*.

4197 [EINTR] A signal interrupted the *aio\_suspend()* function. Note that, since each  
 4198 asynchronous I/O operation may possibly provoke a signal when it  
 4199 completes, this error return may be caused by the completion of one (or more)  
 4200 of the very I/O operations being awaited.

4201 **EXAMPLES**

4202 None.

4203 **APPLICATION USAGE**

4204 The *aio\_suspend()* function is part of the Asynchronous Input and Output option and need not  
 4205 be available on all implementations.

4206 **RATIONALE**

4207 None.

4208 **FUTURE DIRECTIONS**

4209 None.

4210 **SEE ALSO**

4211 *aio\_read()*, *aio\_write()*, *lio\_listio()*, the Base Definitions volume of IEEE Std 1003.1-200x, <**aio.h**>

4212 **CHANGE HISTORY**

4213 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

4214 **Issue 6**

4215 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
4216 implementation does not support the Asynchronous Input and Output option.

4217 The APPLICATION USAGE section is added.

4218 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that the  
4219 CLOCK\_MONOTONIC clock, if supported, is used.

4220 **NAME**4221 aio\_write — asynchronous write to a file (**REALTIME**)4222 **SYNOPSIS**

4223 AIO #include &lt;aio.h&gt;

4224 int aio\_write(struct aiocb \*aiocbp);

4225

4226 **DESCRIPTION**

4227 The *aio\_write()* function shall write *aiocbp->aio\_nbytes* to the file associated with *aiocbp->aio\_fildes* |  
 4228 from the buffer pointed to by *aiocbp->aio\_buf*. The function shall return when the write request |  
 4229 has been initiated or, at a minimum, queued to the file or device. |

4230 PIO If prioritized I/O is supported for this file, then the asynchronous operation shall be submitted |  
 4231 at a priority equal to the scheduling priority of the process minus *aiocbp->aio\_reqprio*.

4232 The *aiocbp* may be used as an argument to *aio\_error()* and *aio\_return()* in order to determine the |  
 4233 error status and return status, respectively, of the asynchronous operation while it is proceeding.

4234 The *aiocbp* argument points to an **aiocb** structure. If the buffer pointed to by *aiocbp->aio\_buf* or |  
 4235 the control block pointed to by *aiocbp* becomes an illegal address prior to asynchronous I/O |  
 4236 completion, then the behavior is undefined.

4237 If **O\_APPEND** is not set for the file descriptor *aio\_fildes*, then the requested operation shall take |  
 4238 place at the absolute position in the file as given by *aio\_offset*, as if *lseek()* were called |  
 4239 immediately prior to the operation with an *offset* equal to *aio\_offset* and a *whence* equal to |  
 4240 **SEEK\_SET**. If **O\_APPEND** is set for the file descriptor, write operations append to the file in the |  
 4241 same order as the calls were made. After a successful call to enqueue an asynchronous I/O |  
 4242 operation, the value of the file offset for the file is unspecified.

4243 The *aiocbp->aio\_lio\_opcode* field shall be ignored by *aio\_write()*.

4244 Simultaneous asynchronous operations using the same *aiocbp* produce undefined results.

4245 SIO If synchronized I/O is enabled on the file associated with *aiocbp->aio\_fildes*, the behavior of this |  
 4246 function shall be according to the definitions of synchronized I/O data integrity completion, and |  
 4247 synchronized I/O file integrity completion.

4248 For any system action that changes the process memory space while an asynchronous I/O is |  
 4249 outstanding to the address range being changed, the result of that action is undefined.

4250 For regular files, no data transfer shall occur past the offset maximum established in the open |  
 4251 file description associated with *aiocbp->aio\_fildes*.

4252 **RETURN VALUE**

4253 The *aio\_write()* function shall return the value zero to the calling process if the I/O operation is |  
 4254 successfully queued; otherwise, the function shall return the value  $-1$  and set *errno* to indicate |  
 4255 the error.

4256 **ERRORS**

4257 The *aio\_write()* function shall fail if:

4258 [EAGAIN] The requested asynchronous I/O operation was not queued due to system |  
 4259 resource limitations.

4260 Each of the following conditions may be detected synchronously at the time of the call to |  
 4261 *aio\_write()*, or asynchronously. If any of the conditions below are detected synchronously, the |  
 4262 *aio\_write()* function shall return  $-1$  and set *errno* to the corresponding value. If any of the |  
 4263 conditions below are detected asynchronously, the return status of the asynchronous operation

4264 shall be set to -1, and the error status of the asynchronous operation is set to the corresponding  
4265 value.

4266 [EBADF] The *aiocbp->aio\_fildes* argument is not a valid file descriptor open for writing.

4267 [EINVAL] The file offset value implied by *aiocbp->aio\_offset* would be invalid, *aiocbp->aio\_reqprio*  
4268 is not a valid value, or *aiocbp->aio\_nbytes* is an invalid value.

4269 In the case that the *aio\_write()* successfully queues the I/O operation, the return status of the  
4270 asynchronous operation shall be one of the values normally returned by the *write()* function call.  
4271 If the operation is successfully queued but is subsequently canceled or encounters an error, the  
4272 error status for the asynchronous operation contains one of the values normally set by the  
4273 *write()* function call, or one of the following:

4274 [EBADF] The *aiocbp->aio\_fildes* argument is not a valid file descriptor open for writing.

4275 [EINVAL] The file offset value implied by *aiocbp->aio\_offset* would be invalid.

4276 [ECANCELED] The requested I/O was canceled before the I/O completed due to an explicit  
4277 *aio\_cancel()* request.

4278 The following condition may be detected synchronously or asynchronously:

4279 [EFBIG] The file is a regular file, *aiocbp->aio\_nbytes* is greater than 0, and the starting  
4280 offset in *aiocbp->aio\_offset* is at or beyond the offset maximum in the open file  
4281 description associated with *aiocbp->aio\_fildes*.

#### 4282 EXAMPLES

4283 None.

#### 4284 APPLICATION USAGE

4285 The *aio\_write()* function is part of the Asynchronous Input and Output option and need not be  
4286 available on all implementations.

#### 4287 RATIONALE

4288 None.

#### 4289 FUTURE DIRECTIONS

4290 None.

#### 4291 SEE ALSO

4292 *aio\_cancel()*, *aio\_error()*, *aio\_read()*, *aio\_return()*, *close()*, *exec*, *exit()*, *fork()*, *lio\_listio()*, *lseek()*,  
4293 *write()*, the Base Definitions volume of IEEE Std 1003.1-200x, <**aio.h**>

#### 4294 CHANGE HISTORY

4295 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

4296 Large File Summit extensions are added.

#### 4297 Issue 6

4298 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
4299 implementation does not support the Asynchronous Input and Output option.

4300 The APPLICATION USAGE section is added.

4301 The following new requirements on POSIX implementations derive from alignment with the  
4302 Single UNIX Specification:

- 4303 • In the DESCRIPTION, text is added to indicate that for regular files no data transfer occurs  
4304 past the offset maximum established in the open file description associated with *aiocbp->aio\_fildes*.  
4305

4306

- The [EFBIG] error is added as part of the large file support extensions.

4307 **NAME**

4308 alarm — schedule an alarm signal

4309 **SYNOPSIS**

4310 #include &lt;unistd.h&gt;

4311 unsigned alarm(unsigned *seconds*);4312 **DESCRIPTION**

4313 The *alarm()* function shall cause the system to generate a SIGALRM signal for the process after  
4314 the number of realtime seconds specified by *seconds* have elapsed. Processor scheduling delays  
4315 may prevent the process from handling the signal as soon as it is generated.

4316 If *seconds* is 0, a pending alarm request, if any, is canceled.

4317 Alarm requests are not stacked; only one SIGALRM generation can be scheduled in this manner.  
4318 If the SIGALRM signal has not yet been generated, the call shall result in rescheduling the time  
4319 at which the SIGALRM signal is generated.

4320 XSI Interactions between *alarm()* and any of *setitimer()*, *ualarm()*, or *usleep()* are unspecified.

4321 **RETURN VALUE**

4322 If there is a previous *alarm()* request with time remaining, *alarm()* shall return a non-zero value  
4323 that is the number of seconds until the previous request would have generated a SIGALRM  
4324 signal. Otherwise, *alarm()* shall return 0.

4325 **ERRORS**

4326 The *alarm()* function is always successful, and no return value is reserved to indicate an error.

4327 **EXAMPLES**

4328 None.

4329 **APPLICATION USAGE**

4330 The *fork()* function clears pending alarms in the child process. A new process image created by  
4331 one of the *exec* functions inherits the time left to an alarm signal in the old process' image.

4332 Application writers should note that the type of the argument *seconds* and the return value of  
4333 *alarm()* is **unsigned**. That means that a Strictly Conforming POSIX System Interfaces  
4334 Application cannot pass a value greater than the minimum guaranteed value for {UINT\_MAX},  
4335 which the ISO C standard sets as 65 535, and any application passing a larger value is restricting  
4336 its portability. A different type was considered, but historical implementations, including those  
4337 with a 16-bit **int** type, consistently use either **unsigned** or **int**.

4338 Application writers should be aware of possible interactions when the same process uses both  
4339 the *alarm()* and *sleep()* functions.

4340 **RATIONALE**

4341 Many historical implementations (including Version 7 and System V) allow an alarm to occur up  
4342 to a second early. Other implementations allow alarms up to half a second or one clock tick  
4343 early or do not allow them to occur early at all. The latter is considered most appropriate, since it  
4344 gives the most predictable behavior, especially since the signal can always be delayed for an  
4345 indefinite amount of time due to scheduling. Applications can thus choose the *seconds* argument  
4346 as the minimum amount of time they wish to have elapse before the signal.

4347 The term *realtime* here and elsewhere (*sleep()*, *times()*) is intended to mean “wall clock” time as  
4348 common English usage, and has nothing to do with “realtime operating systems”. It is in  
4349 contrast to *virtual time*, which could be misinterpreted if just *time* were used.

4350 In some implementations, including 4.3 BSD, very large values of the *seconds* argument are  
4351 silently rounded down to an implementation-defined maximum value. This maximum is large

4352 enough (on the order of several months) that the effect is not noticeable.

4353 There were two possible choices for alarm generation in multi-threaded applications: generation  
4354 for the calling thread or generation for the process. The first option would not have been  
4355 particularly useful since the alarm state is maintained on a per-process basis and the alarm that  
4356 is established by the last invocation of *alarm()* is the only one that would be active.

4357 Furthermore, allowing generation of an asynchronous signal for a thread would have introduced  
4358 an exception to the overall signal model. This requires a compelling reason in order to be  
4359 justified.

4360 **FUTURE DIRECTIONS**

4361 None.

4362 **SEE ALSO**

4363 *alarm()*, *exec*, *fork()*, *getitimer()*, *pause()*, *sigaction()*, *sleep()*, *ualarm()*, *usleep()*, the Base  
4364 Definitions volume of IEEE Std 1003.1-200x, <signal.h>, <unistd.h>

4365 **CHANGE HISTORY**

4366 First released in Issue 1. Derived from Issue 1 of the SVID.

4367 **Issue 6**

4368 The following new requirements on POSIX implementations derive from alignment with the  
4369 Single UNIX Specification:

- 4370 • The DESCRIPTION is updated to indicate that interactions with the *setitimer()*, *ualarm()*, and  
4371 *usleep()* functions are unspecified.

## 4372 NAME

4373 asctime, asctime\_r — convert date and time to a string

## 4374 SYNOPSIS

4375 #include &lt;time.h&gt;

4376 char \*asctime(const struct tm \*timeptr);

4377 TSF char \*asctime\_r(const struct tm \*restrict tm, char \*restrict buf);

4378

## 4379 DESCRIPTION

4380 CX For *asctime()*: The functionality described on this reference page is aligned with the ISO C |  
 4381 standard. Any conflict between the requirements described here and the ISO C standard is  
 4382 unintentional. This volume of IEEE Std 1003.1-200x defers to the ISO C standard.

4383 The *asctime()* function shall convert the broken-down time in the structure pointed to by *timeptr*  
 4384 into a string in the form:

4385 Sun Sep 16 01:03:52 1973\n\0

4386 using the equivalent of the following algorithm:

```

4387 char *asctime(const struct tm *timeptr)
4388 {
4389     static char wday_name[7][3] = {
4390         "Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"
4391     };
4392     static char mon_name[12][3] = {
4393         "Jan", "Feb", "Mar", "Apr", "May", "Jun",
4394         "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"
4395     };
4396     static char result[26];
4397
4398     sprintf(result, "%.3s %.3s%3d %.2d:%.2d:%.2d %d\n",
4399             wday_name[timeptr->tm_wday],
4400             mon_name[timeptr->tm_mon],
4401             timeptr->tm_mday, timeptr->tm_hour,
4402             timeptr->tm_min, timeptr->tm_sec,
4403             1900 + timeptr->tm_year);
4404     return result;
4405 }
```

4405 The **tm** structure is defined in the <time.h> header. |

4406 CX The *asctime()*, *ctime()*, *gmtime()*, and *localtime()* functions shall return values in one of two static  
 4407 objects: a broken-down time structure and an array of type **char**. Execution of any of the  
 4408 functions may overwrite the information returned in either of these objects by any of the other  
 4409 functions.

4410 The *asctime()* function need not be reentrant. A function that is not required to be reentrant is not  
 4411 required to be thread-safe.

4412 TSF The *asctime\_r()* function shall convert the broken-down time in the structure pointed to by *tm*  
 4413 into a string (of the same form as that returned by *asctime()*) that is placed in the user-supplied  
 4414 buffer pointed to by *buf* (which shall contain at least 26 bytes) and then return *buf*.

4415 **RETURN VALUE**

4416 Upon successful completion, *asctime()* shall return a pointer to the string.

4417 TSF Upon successful completion, *asctime\_r()* shall return a pointer to a character string containing  
4418 the date and time. This string is pointed to by the argument *buf*. If the function is unsuccessful,  
4419 it shall return NULL.

4420 **ERRORS**

4421 No errors are defined.

4422 **EXAMPLES**

4423 None.

4424 **APPLICATION USAGE**

4425 Values for the broken-down time structure can be obtained by calling *gmtime()* or *localtime()*.  
4426 This function is included for compatibility with older implementations, and does not support  
4427 localized date and time formats. Applications should use *strptime()* to achieve maximum  
4428 portability.

4429 The *asctime\_r()* function is thread-safe and shall return values in a user-supplied buffer instead  
4430 of possibly using a static data area that may be overwritten by each call.

4431 **RATIONALE**

4432 None.

4433 **FUTURE DIRECTIONS**

4434 None.

4435 **SEE ALSO**

4436 *clock()*, *ctime()*, *difftime()*, *gmtime()*, *localtime()*, *mktime()*, *strptime()*, *strptime()*, *time()*, *utime()*,  
4437 the Base Definitions volume of IEEE Std 1003.1-200x, <**time.h**>

4438 **CHANGE HISTORY**

4439 First released in Issue 1. Derived from Issue 1 of the SVID.

4440 **Issue 5**

4441 Normative text previously in the APPLICATION USAGE section is moved to the  
4442 DESCRIPTION.

4443 The *asctime\_r()* function is included for alignment with the POSIX Threads Extension.

4444 A note indicating that the *asctime()* function need not be reentrant is added to the  
4445 DESCRIPTION.

4446 **Issue 6**

4447 The *asctime\_r()* function is marked as part of the Thread-Safe Functions option.

4448 Extensions beyond the ISO C standard are now marked.

4449 The APPLICATION USAGE section is updated to include a note on the thread-safe function and  
4450 its avoidance of possibly using a static data area.

4451 The DESCRIPTION of *asctime\_r()* is updated to describe the format of the string returned.

4452 The **restrict** keyword is added to the *asctime\_r()* prototype for alignment with the  
4453 ISO/IEC 9899:1999 standard.

4454 **NAME**

4455 asin, asinf, asinl — arc sine function

4456 **SYNOPSIS**

4457 #include &lt;math.h&gt;

4458 double asin(double x);

4459 float asinf(float x);

4460 long double asinl(long double x);

4461 **DESCRIPTION**

4462 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 4463 conflict between the requirements described here and the ISO C standard is unintentional. This  
 4464 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

4465 These functions shall compute the principal value of the arc sine of their argument *x*. The value  
 4466 of *x* should be in the range  $[-1,1]$ .

4467 An application wishing to check for error situations should set *errno* to zero and call  
 4468 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 4469 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 4470 zero, an error has occurred.

4471 **RETURN VALUE**

4472 Upon successful completion, these functions shall return the arc sine of *x*, in the range  
 4473  $[-\pi/2, \pi/2]$  radians.

4474 **MX** For finite values of *x* not in the range  $[-1,1]$ , a domain error shall occur, and either a NaN (if  
 4475 supported), or an implementation-defined value shall be returned.

4476 **MX** If *x* is NaN, a NaN shall be returned.

4477 If *x* is  $\pm 0$ , *x* shall be returned.

4478 If *x* is  $\pm \text{Inf}$ , a domain error shall occur, and either a NaN (if supported), or an implementation-  
 4479 defined value shall be returned.

4480 If *x* is subnormal, a range error may occur and *x* should be returned.

4481 **ERRORS**

4482 These functions shall fail if:

4483 **MX** **Domain Error** The *x* argument is finite and is not in the range  $[-1,1]$ , or is  $\pm \text{Inf}$ .

4484 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero, |  
 4485 then *errno* shall be set to [EDOM]. If the integer expression (math\_errhandling |  
 4486 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception |  
 4487 shall be raised. |

4488 These functions may fail if:

4489 **MX** **Range Error** The value of *x* is subnormal.

4490 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero, |  
 4491 then *errno* shall be set to [ERANGE]. If the integer expression |  
 4492 (math\_errhandling & MATH\_ERREXCEPT) is non-zero, then the underflow |  
 4493 floating-point exception shall be raised. |

4494 **EXAMPLES**

4495       None.

4496 **APPLICATION USAGE**

4497       On error, the expressions (`math_errhandling & MATH_ERRNO`) and (`math_errhandling &`  
4498       `MATH_ERREXCEPT`) are independent of each other, but at least one of them must be non-zero.

4499 **RATIONALE**

4500       None.

4501 **FUTURE DIRECTIONS**

4502       None.

4503 **SEE ALSO**

4504       *feclearexcept()*, *fetestexcept()*, *isnan()*, *sin()*, the Base Definitions volume of IEEE Std 1003.1-200x, |  
4505       Section 4.18, Treatment of Error Conditions for Mathematical Functions, <**math.h**> |

4506 **CHANGE HISTORY**

4507       First released in Issue 1. Derived from Issue 1 of the SVID.

4508 **Issue 5**

4509       The DESCRIPTION is updated to indicate how an application should check for an error. This  
4510       text was previously published in the APPLICATION USAGE section.

4511 **Issue 6**4512       The *asinf()* and *asinl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

4513       The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
4514       revised to align with the ISO/IEC 9899:1999 standard.

4515       IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
4516       marked.

4517 **NAME**4518        **asinf** — arc sine function4519 **SYNOPSIS**

4520        #include &lt;math.h&gt;

4521        float asinf(float x);

4522 **DESCRIPTION**4523        Refer to *asin()*.

4524 **NAME**

4525        asinhf, asinhl — inverse hyperbolic sine functions

4526 **SYNOPSIS**

4527        #include &lt;math.h&gt;

4528        float asinhf(float x);

4529        long double asinhl(long double x);

4530 **DESCRIPTION**4531        Refer to *asinh()*.

4532 **NAME**

4533 asinh, asinhf, asinhl — inverse hyperbolic sine functions

4534 **SYNOPSIS**

```
4535 #include <math.h>
4536 double asinh(double x);
4537 float asinhf(float x);
4538 long double asinhl(long double x);
```

4539 **DESCRIPTION**

4540 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 4541 conflict between the requirements described here and the ISO C standard is unintentional. This  
 4542 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

4543 These functions shall compute the inverse hyperbolic sine of their argument *x*.

4544 An application wishing to check for error situations should set *errno* to zero and call  
 4545 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 4546 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 4547 zero, an error has occurred.

4548 **RETURN VALUE**

4549 Upon successful completion, these functions shall return the inverse hyperbolic sine, of their  
 4550 argument.

4551 **MX** If *x* is NaN, a NaN shall be returned.

4552 If *x* is ±0, or ±Inf, *x* shall be returned.

4553 If *x* is subnormal, a range error may occur and *x* should be returned.

4554 **ERRORS**

4555 These functions may fail if:

4556 **MX** **Range Error** The value of *x* is subnormal.

4557 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero, |  
 4558 then *errno* shall be set to [ERANGE]. If the integer expression |  
 4559 (math\_errhandling & MATH\_ERREXCEPT) is non-zero, then the underflow |  
 4560 floating-point exception shall be raised. |

4561 **EXAMPLES**

4562 None.

4563 **APPLICATION USAGE**

4564 On error, the expressions (math\_errhandling & MATH\_ERRNO) and (math\_errhandling &  
 4565 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

4566 **RATIONALE**

4567 None.

4568 **FUTURE DIRECTIONS**

4569 None.

4570 **SEE ALSO**

4571 *feclearexcept*(), *fetestexcept*(), *sinh*(), the Base Definitions volume of IEEE Std 1003.1-200x, Section |  
 4572 4.18, Treatment of Error Conditions for Mathematical Functions, <math.h> |

4573 **CHANGE HISTORY**

4574 First released in Issue 4, Version 2.

4575 **Issue 5**

4576 Moved from X/OPEN UNIX extension to BASE.

4577 **Issue 6**4578 The *asinh()* function is no longer marked as an extension.4579 The *asinhf()*, and *asinhll()* functions are added for alignment with the ISO/IEC 9899:1999  
4580 standard.4581 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
4582 revised to align with the ISO/IEC 9899:1999 standard.4583 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
4584 marked.

4585 **NAME**4586        **asinl** — arc sine function4587 **SYNOPSIS**

4588        #include &lt;math.h&gt;

4589        long double asinl(long double x);

4590 **DESCRIPTION**4591        Refer to *asin()*.

4592 **NAME**

4593        assert — insert program diagnostics

4594 **SYNOPSIS**

4595        #include &lt;assert.h&gt;

4596        void assert(*scalar expression*);4597 **DESCRIPTION**

4598 **cx**     The functionality described on this reference page is aligned with the ISO C standard. Any  
4599 conflict between the requirements described here and the ISO C standard is unintentional. This  
4600 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

4601     The *assert()* macro shall insert diagnostics into programs; it shall expand to a **void** expression. |  
4602     When it is executed, if *expression* (which shall have a **scalar** type) is false (that is, compares equal  
4603     to 0), *assert()* shall write information about the particular call that failed on *stderr* and shall call  
4604     *abort()*.

4605     The information written about the call that failed shall include the text of the argument, the  
4606     name of the source file, the source file line number, and the name of the enclosing function, the  
4607     latter are, respectively, the values of the preprocessing macros `__FILE__` and `__LINE__` and of  
4608     the identifier `__func__`.

4609     Forcing a definition of the name `NDEBUG`, either from the compiler command line or with the  
4610     preprocessor control statement `#define NDEBUG` ahead of the `#include <assert.h>` statement,  
4611     shall stop assertions from being compiled into the program.

4612 **RETURN VALUE**4613        The *assert()* macro shall not return a value.4614 **ERRORS**

4615        No errors are defined.

4616 **EXAMPLES**

4617        None.

4618 **APPLICATION USAGE**

4619        None.

4620 **RATIONALE**

4621        None.

4622 **FUTURE DIRECTIONS**

4623        None.

4624 **SEE ALSO**4625        *abort()*, the Base Definitions volume of IEEE Std 1003.1-200x, `<assert.h>`, *stderr*4626 **CHANGE HISTORY**

4627        First released in Issue 1. Derived from Issue 1 of the SVID.

4628 **Issue 6**

4629     The prototype for the *expression* argument to *assert()* is changed from **int** to **scalar** for alignment  
4630     with the ISO/IEC 9899:1999 standard.

4631     The DESCRIPTION of *assert()* is updated for alignment with the ISO/IEC 9899:1999 standard.

4632 **NAME**

4633 atan, atanf, atanl — arc tangent function

4634 **SYNOPSIS**

4635 #include &lt;math.h&gt;

4636 double atan(double x);

4637 float atanf(float x);

4638 long double atanl(long double x);

4639 **DESCRIPTION**

4640 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 4641 conflict between the requirements described here and the ISO C standard is unintentional. This  
 4642 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

4643 These functions shall compute the principal value of the arc tangent of their argument  $x$ .

4644 An application wishing to check for error situations should set *errno* to zero and call  
 4645 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 4646 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 4647 zero, an error has occurred.

4648 **RETURN VALUE**4649 Upon successful completion, these functions shall return the arc tangent of  $x$  in the range  
4650  $[-\pi/2, \pi/2]$  radians.4651 **MX** If  $x$  is NaN, a NaN shall be returned.4652 If  $x$  is  $\pm 0$   $x$  shall be returned.4653 If  $x$  is  $\pm\text{Inf}$ ,  $\pm\pi/2$  shall be returned.4654 If  $x$  is subnormal, a range error may occur and  $x$  should be returned.4655 **ERRORS**

4656 These functions may fail if:

4657 **MX** **Range Error** The value of  $x$  is subnormal.

4658 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero, |  
 4659 then *errno* shall be set to [ERANGE]. If the integer expression |  
 4660 (math\_errhandling & MATH\_ERREXCEPT) is non-zero, then the underflow |  
 4661 floating-point exception shall be raised. |

4662 **EXAMPLES**

4663 None.

4664 **APPLICATION USAGE**4665 On error, the expressions (math\_errhandling & MATH\_ERRNO) and (math\_errhandling &  
4666 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.4667 **RATIONALE**

4668 None.

4669 **FUTURE DIRECTIONS**

4670 None.

4671 **SEE ALSO**

4672 *atan2*(), *feclearexcept*(), *fetestexcept*(), *isnan*(), *tan*(), the Base Definitions volume of |  
 4673 IEEE Std 1003.1-200x, Section 4.18, Treatment of Error Conditions for Mathematical Functions, |  
 4674 <math.h>

4675 **CHANGE HISTORY**

4676 First released in Issue 1. Derived from Issue 1 of the SVID.

4677 **Issue 5**

4678 The DESCRIPTION is updated to indicate how an application should check for an error. This  
4679 text was previously published in the APPLICATION USAGE section.

4680 **Issue 6**

4681 The *atanf()* and *atanl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

4682 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
4683 revised to align with the ISO/IEC 9899:1999 standard.

4684 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
4685 marked.

4686 **NAME**

4687         atan2, atan2f, atan2l — arc tangent functions

4688 **SYNOPSIS**

4689         #include &lt;math.h&gt;

4690         double atan2(double y, double x);

4691         float atan2f(float y, float x);

4692         long double atan2l(long double y, long double x);

4693 **DESCRIPTION**

4694 **CX**         The functionality described on this reference page is aligned with the ISO C standard. Any  
 4695 conflict between the requirements described here and the ISO C standard is unintentional. This  
 4696 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

4697         These functions shall compute the principal value of the arc tangent of  $y/x$ , using the signs of  
 4698 both arguments to determine the quadrant of the return value.

4699         An application wishing to check for error situations should set *errno* to zero and call  
 4700 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 4701 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 4702 zero, an error has occurred.

4703 **RETURN VALUE**

4704         Upon successful completion, these functions shall return the arc tangent of  $y/x$  in the range  
 4705  $[-\pi, \pi]$  radians.

4706         If  $y$  is  $\pm 0$  and  $x$  is  $< 0$ ,  $\pm\pi$  shall be returned.

4707         If  $y$  is  $\pm 0$  and  $x$  is  $> 0$ ,  $\pm 0$  shall be returned.

4708         If  $y$  is  $< 0$  and  $x$  is  $\pm 0$ ,  $-\pi/2$  shall be returned.

4709         If  $y$  is  $> 0$  and  $x$  is  $\pm 0$ ,  $\pi/2$  shall be returned.

4710         If  $x$  is 0, a pole error shall not occur.

4711 **MX**         If either  $x$  or  $y$  is NaN, a NaN shall be returned.

4712         If the result underflows, a range error may occur and  $y/x$  should be returned.

4713         If  $y$  is  $\pm 0$  and  $x$  is  $-0$ ,  $\pm\pi$  shall be returned.

4714         If  $y$  is  $\pm 0$  and  $x$  is  $+0$ ,  $\pm 0$  shall be returned.

4715         For finite values of  $\pm y > 0$ , if  $x$  is  $-\text{Inf}$ ,  $\pm\pi$  shall be returned.

4716         For finite values of  $\pm y > 0$ , if  $x$  is  $+\text{Inf}$ ,  $\pm 0$  shall be returned.

4717         For finite values of  $x$ , if  $y$  is  $\pm\text{Inf}$ ,  $\pm\pi/2$  shall be returned.

4718         If  $y$  is  $\pm\text{Inf}$  and  $x$  is  $-\text{Inf}$ ,  $\pm 3\pi/4$  shall be returned.

4719         If  $y$  is  $\pm\text{Inf}$  and  $x$  is  $+\text{Inf}$ ,  $\pm\pi/4$  shall be returned.

4720         If both arguments are 0, a domain error shall not occur.

4721 **ERRORS**

4722         These functions may fail if:

4723 **MX**         **Range Error**         The result underflows.

4724         If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero, |  
 4725 then *errno* shall be set to [ERANGE]. If the integer expression |

4726 (math\_errhandling & MATH\_ERREXCEPT) is non-zero, then the underflow |  
4727 floating-point exception shall be raised. |

4728 **EXAMPLES**

4729 None.

4730 **APPLICATION USAGE**

4731 On error, the expressions (math\_errhandling & MATH\_ERRNO) and (math\_errhandling &  
4732 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

4733 **RATIONALE**

4734 None.

4735 **FUTURE DIRECTIONS**

4736 None.

4737 **SEE ALSO**

4738 *atan()*, *feclearexcept()*, *fetestexcept()*, *isnan()*, *tan()*, the Base Definitions volume of |  
4739 IEEE Std 1003.1-200x, Section 4.18, Treatment of Error Conditions for Mathematical Functions, |  
4740 <math.h>

4741 **CHANGE HISTORY**

4742 First released in Issue 1. Derived from Issue 1 of the SVID.

4743 **Issue 5**

4744 The DESCRIPTION is updated to indicate how an application should check for an error. This  
4745 text was previously published in the APPLICATION USAGE section.

4746 **Issue 6**

4747 The *atan2f()* and *atan2l()* functions are added for alignment with the ISO/IEC 9899:1999  
4748 standard.

4749 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
4750 revised to align with the ISO/IEC 9899:1999 standard.

4751 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
4752 marked.

4753 **NAME**4754        **atanf** — arc tangent function4755 **SYNOPSIS**

4756        #include &lt;math.h&gt;

4757        float atanf(float x);

4758 **DESCRIPTION**4759        Refer to *atan()*.

4760 **NAME**

4761           atanh, atanhf, atanh1 — inverse hyperbolic tangent functions

4762 **SYNOPSIS**

4763           #include &lt;math.h&gt;

4764           double atanh(double x);

4765           float atanhf(float x);

4766           long double atanh1(long double x);

4767 **DESCRIPTION**

4768 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
 4769 conflict between the requirements described here and the ISO C standard is unintentional. This  
 4770 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

4771       These functions shall compute the inverse hyperbolic tangent of their argument  $x$ .

4772       An application wishing to check for error situations should set *errno* to zero and call  
 4773 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 4774 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 4775 zero, an error has occurred.

4776 **RETURN VALUE**4777       Upon successful completion, these functions shall return the inverse hyperbolic tangent of their  
4778 argument.

4779       If  $x$  is  $\pm 1$ , a pole error shall occur, and *atanh*( $x$ ), *atanhf*( $x$ ), and *atanh1*( $x$ ) shall return the value of the  
 4780 macro HUGE\_VAL, HUGE\_VALF, and HUGE\_VALL, respectively, with the same sign as the  
 4781 correct value of the function.

4782 **MX**       For finite  $|x| > 1$ , a domain error shall occur, and either a NaN (if supported), or an  
 4783 implementation-defined value shall be returned.

4784 **MX**       If  $x$  is NaN, a NaN shall be returned.4785       If  $x$  is  $\pm 0$ ,  $x$  shall be returned.

4786       If  $x$  is  $\pm \text{Inf}$ , a domain error shall occur, and either a NaN (if supported), or an implementation-  
 4787 defined value shall be returned.

4788       If  $x$  is subnormal, a range error may occur and  $x$  should be returned.4789 **ERRORS**

4790       These functions shall fail if:

4791 **MX**       Domain Error    The  $x$  argument is finite and not in the range  $[-1, 1]$ , or is  $\pm \text{Inf}$ .

4792               If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero, |  
 4793 then *errno* shall be set to [EDOM]. If the integer expression (math\_errhandling |  
 4794 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception |  
 4795 shall be raised. |

4796       Pole Error        The  $x$  argument is  $\pm 1$ .

4797               If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero, |  
 4798 then *errno* shall be set to [ERANGE]. If the integer expression |  
 4799 (math\_errhandling & MATH\_ERREXCEPT) is non-zero, then the divide-by- |  
 4800 zero floating-point exception shall be raised. |

4801       These functions may fail if:



4831 **NAME**

4832       atanl — arc tangent function

4833 **SYNOPSIS**

4834       #include <math.h>

4835       long double atanl(long double x);

4836 **DESCRIPTION**

4837       Refer to *atan()*.

4838 **NAME**

4839         atexit — register a function to run at process termination

4840 **SYNOPSIS**

4841         #include &lt;stdlib.h&gt;

4842         int atexit(void (\*func)(void));

4843 **DESCRIPTION**4844 **CX**         The functionality described on this reference page is aligned with the ISO C standard. Any  
4845 conflict between the requirements described here and the ISO C standard is unintentional. This  
4846 volume of IEEE Std 1003.1-200x defers to the ISO C standard.4847         The *atexit()* function shall register the function pointed to by *func*, to be called without  
4848 arguments at normal program termination. At normal program termination, all functions  
4849 registered by the *atexit()* function shall be called, in the reverse order of their registration, except  
4850 that a function is called after any previously registered functions that had already been called at  
4851 the time it was registered. Normal termination occurs either by a call to *exit()* or a return from  
4852 *main()*.4853         At least 32 functions can be registered with *atexit()*.4854 **CX**         After a successful call to any of the *exec* functions, any functions previously registered by *atexit()*  
4855 shall no longer be registered.4856 **RETURN VALUE**4857         Upon successful completion, *atexit()* shall return 0; otherwise, it shall return a non-zero value.4858 **ERRORS**

4859         No errors are defined.

4860 **EXAMPLES**

4861         None.

4862 **APPLICATION USAGE**4863         The functions registered by a call to *atexit()* must return to ensure that all registered functions  
4864 are called.4865         The application should call *sysconf()* to obtain the value of {ATEXIT\_MAX}, the number of  
4866 functions that can be registered. There is no way for an application to tell how many functions  
4867 have already been registered with *atexit()*.4868 **RATIONALE**

4869         None.

4870 **FUTURE DIRECTIONS**

4871         None.

4872 **SEE ALSO**4873         *exit()*, *sysconf()*, the Base Definitions volume of IEEE Std 1003.1-200x, <stdlib.h>4874 **CHANGE HISTORY**

4875         First released in Issue 4. Derived from the ANSI C standard.

4876 **Issue 6**

4877         Extensions beyond the ISO C standard are now marked.

4878         The DESCRIPTION is updated for alignment with the ISO/IEC 9899:1999 standard.

4879 **NAME**

4880        atof — convert a string to double-precision number

4881 **SYNOPSIS**

4882        #include &lt;stdlib.h&gt;

4883        double atof(const char \*str);

4884 **DESCRIPTION**

4885 cx       The functionality described on this reference page is aligned with the ISO C standard. Any  
4886 conflict between the requirements described here and the ISO C standard is unintentional. This  
4887 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

4888        The call *atof(str)* shall be equivalent to:4889        *strtod(str, (char \*\*)NULL),*

4890        except that the handling of errors may differ. If the value cannot be represented, the behavior is  
4891 undefined.

4892 **RETURN VALUE**4893        The *atof()* function shall return the converted value if the value can be represented.4894 **ERRORS**

4895        No errors are defined.

4896 **EXAMPLES**

4897        None.

4898 **APPLICATION USAGE**

4899        The *atof()* function is subsumed by *strtod()* but is retained because it is used extensively in  
4900 existing code. If the number is not known to be in range, *strtod()* should be used because *atof()* is  
4901 not required to perform any error checking.

4902 **RATIONALE**

4903        None.

4904 **FUTURE DIRECTIONS**

4905        None.

4906 **SEE ALSO**4907        *strtod()*, the Base Definitions volume of IEEE Std 1003.1-200x, <stdlib.h>4908 **CHANGE HISTORY**

4909        First released in Issue 1. Derived from Issue 1 of the SVID.

4910 **NAME**

4911         atoi — convert a string to an integer

4912 **SYNOPSIS**

4913         #include &lt;stdlib.h&gt;

4914         int atoi(const char \*str);

4915 **DESCRIPTION**

4916 cx       The functionality described on this reference page is aligned with the ISO C standard. Any  
4917 conflict between the requirements described here and the ISO C standard is unintentional. This  
4918 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

4919         The call *atoi(str)* shall be equivalent to:

4920         (int) strtol(str, (char \*\*)NULL, 10)

4921         except that the handling of errors may differ. If the value cannot be represented, the behavior is  
4922 undefined.4923 **RETURN VALUE**4924         The *atoi()* function shall return the converted value if the value can be represented.4925 **ERRORS**

4926         No errors are defined.

4927 **EXAMPLES**4928         **Converting an Argument**4929         The following example checks for proper usage of the program. If there is an argument and the  
4930 decimal conversion of this argument (obtained using *atoi()*) is greater than 0, then the program  
4931 has a valid number of minutes to wait for an event.

```
4932         #include <stdlib.h>
4933         #include <stdio.h>
4934         ...
4935         int minutes_to_event;
4936         ...
4937         if (argc < 2 || ((minutes_to_event = atoi (argv[1]))) <= 0) {
4938             fprintf(stderr, "Usage: %s minutes\n", argv[0]); exit(1);
4939         }
4940         ...
```

4941 **APPLICATION USAGE**4942         The *atoi()* function is subsumed by *strtol()* but is retained because it is used extensively in  
4943 existing code. If the number is not known to be in range, *strtol()* should be used because *atoi()* is  
4944 not required to perform any error checking.4945 **RATIONALE**

4946         None.

4947 **FUTURE DIRECTIONS**

4948         None.

4949 **SEE ALSO**4950         *strtol()*, the Base Definitions volume of IEEE Std 1003.1-200x, <stdlib.h>

4951 **CHANGE HISTORY**

4952 First released in Issue 1. Derived from Issue 1 of the SVID.

4953 **NAME**

4954        atol, atoll — convert a string to a long integer

4955 **SYNOPSIS**

4956        #include &lt;stdlib.h&gt;

4957        long atol(const char \*str);

4958        long long atoll(const char \*nptr);

4959 **DESCRIPTION**4960 cx       The functionality described on this reference page is aligned with the ISO C standard. Any  
4961 conflict between the requirements described here and the ISO C standard is unintentional. This  
4962 volume of IEEE Std 1003.1-200x defers to the ISO C standard.4963        The call *atol(str)* shall be equivalent to:

4964        strtol(str, (char \*\*)NULL, 10)

4965        The call *atoll(str)* shall be equivalent to:

4966        strtoll(nptr, (char \*\*)NULL, 10)

4967        except that the handling of errors may differ. If the value cannot be represented, the behavior is  
4968 undefined.4969 **RETURN VALUE**

4970        These functions shall return the converted value if the value can be represented.

4971 **ERRORS**

4972        No errors are defined.

4973 **EXAMPLES**

4974        None.

4975 **APPLICATION USAGE**4976        The *atol()* function is subsumed by *strtol()* but is retained because it is used extensively in  
4977 existing code. If the number is not known to be in range, *strtol()* should be used because *atol()* is  
4978 not required to perform any error checking.4979 **RATIONALE**

4980        None.

4981 **FUTURE DIRECTIONS**

4982        None.

4983 **SEE ALSO**4984        *strtol()*, the Base Definitions volume of IEEE Std 1003.1-200x, <stdlib.h>4985 **CHANGE HISTORY**

4986        First released in Issue 1. Derived from Issue 1 of the SVID.

4987 **Issue 6**4988        The *atoll()* function is added for alignment with the ISO/IEC 9899:1999 standard.

4989 **NAME**4990 `basename` — return the last component of a pathname |4991 **SYNOPSIS**4992 `XSI` `#include <libgen.h>`4993 `char *basename(char *path);`

4994

4995 **DESCRIPTION**4996 The `basename()` function shall take the pathname pointed to by `path` and return a pointer to the |  
4997 final component of the pathname, deleting any trailing `'/'` characters. |4998 If the string consists entirely of the `'/'` character, `basename()` shall return a pointer to the string  
4999 `"/"`. If the string is exactly `"/"`, it is implementation-defined whether `'/'` or `"/"` is  
5000 returned.5001 If `path` is a null pointer or points to an empty string, `basename()` shall return a pointer to the  
5002 string  `"."`.5003 The `basename()` function may modify the string pointed to by `path`, and may return a pointer to  
5004 static storage that may then be overwritten by a subsequent call to `basename()`.5005 The `basename()` function need not be reentrant. A function that is not required to be reentrant is  
5006 not required to be thread-safe.5007 **RETURN VALUE**5008 The `basename()` function shall return a pointer to the final component of `path`.5009 **ERRORS**

5010 No errors are defined.

5011 **EXAMPLES**5012 **Using `basename()`**5013 The following program fragment returns a pointer to the value `lib`, which is the base name of  
5014 `/usr/lib`.5015 `#include <libgen.h>`  
5016 `...`  
5017 `char *name = "/usr/lib";`  
5018 `char *base;`  
5019 `base = basename(name);`  
5020 `...`5021 **Sample Input and Output Strings for `basename()`**5022 In the following table, the input string is the value pointed to by `path`, and the output string is  
5023 the return value of the `basename()` function.

5024

5025

5026

5027

Input String	Output String
<code>"/usr/lib"</code>	<code>"lib"</code>
<code>"/usr/"</code>	<code>"usr"</code>
<code>"/"</code>	<code>"/"</code>

5028 **APPLICATION USAGE**

5029       None.

5030 **RATIONALE**

5031       None.

5032 **FUTURE DIRECTIONS**

5033       None.

5034 **SEE ALSO**5035       *dirname()*, the Base Definitions volume of IEEE Std 1003.1-200x, <**libgen.h**>, the Shell and  
5036       Utilities volume of IEEE Std 1003.1-200x, *basename*5037 **CHANGE HISTORY**

5038       First released in Issue 4, Version 2.

5039 **Issue 5**

5040       Moved from X/OPEN UNIX extension to BASE.

5041       Normative text previously in the APPLICATION USAGE section is moved to the  
5042       DESCRIPTION.

5043       A note indicating that this function need not be reentrant is added to the DESCRIPTION.

5044 **Issue 6**

5045       In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

5046 **NAME**5047       bcmp — memory operations (**LEGACY**)5048 **SYNOPSIS**

5049 XSI       #include &lt;strings.h&gt;

5050       int bcmp(const void \*s1, const void \*s2, size\_t n);

5051

5052 **DESCRIPTION**5053       The *bcmp()* function shall compare the first *n* bytes of the area pointed to by *s1* with the area  
5054       pointed to by *s2*.5055 **RETURN VALUE**5056       The *bcmp()* function shall return 0 if *s1* and *s2* are identical; otherwise, it shall return non-zero.  
5057       Both areas are assumed to be *n* bytes long. If the value of *n* is 0, *bcmp()* shall return 0.5058 **ERRORS**

5059       No errors are defined.

5060 **EXAMPLES**

5061       None.

5062 **APPLICATION USAGE**5063       *memcmp()* is preferred over this function.5064       For maximum portability, it is recommended to replace the function call to *bcmp()* as follows:

5065       #define bcmp(b1,b2,len) memcmp((b1), (b2), (size\_t)(len))

5066 **RATIONALE**

5067       None.

5068 **FUTURE DIRECTIONS**

5069       This function may be withdrawn in a future version.

5070 **SEE ALSO**5071       *memcmp()*, the Base Definitions volume of IEEE Std 1003.1-200x, <strings.h>5072 **CHANGE HISTORY**

5073       First released in Issue 4, Version 2.

5074 **Issue 5**

5075       Moved from X/OPEN UNIX extension to BASE.

5076 **Issue 6**

5077       This function is marked LEGACY.

5078 **NAME**5079 `bcopy` — memory operations (**LEGACY**)5080 **SYNOPSIS**5081 XSI 

```
#include <strings.h>
```

5082 

```
void bcopy(const void *s1, void *s2, size_t n);
```

5083

5084 **DESCRIPTION**5085 The *bcopy()* function shall copy *n* bytes from the area pointed to by *s1* to the area pointed to by  
5086 *s2*.5087 The bytes are copied correctly even if the area pointed to by *s1* overlaps the area pointed to by  
5088 *s2*.5089 **RETURN VALUE**5090 The *bcopy()* function shall not return a value.5091 **ERRORS**

5092 No errors are defined.

5093 **EXAMPLES**

5094 None.

5095 **APPLICATION USAGE**5096 *memmove()* is preferred over this function.

5097 The following are approximately equivalent (note the order of the arguments):

5098 `bcopy(s1,s2,n) ~ memmove(s2,s1,n)`5099 For maximum portability, it is recommended to replace the function call to *bcopy()* as follows:5100 

```
#define bcopy(b1,b2,len) (memmove((b2), (b1), (len)), (void) 0)
```

5101 **RATIONALE**

5102 None.

5103 **FUTURE DIRECTIONS**

5104 This function may be withdrawn in a future version.

5105 **SEE ALSO**5106 *memmove()*, the Base Definitions volume of IEEE Std 1003.1-200x, `<strings.h>`5107 **CHANGE HISTORY**

5108 First released in Issue 4, Version 2.

5109 **Issue 5**

5110 Moved from X/OPEN UNIX extension to BASE.

5111 **Issue 6**

5112 This function is marked LEGACY.

5113 **NAME**

5114        bind — bind a name to a socket

5115 **SYNOPSIS**

5116        #include &lt;sys/socket.h&gt;

5117        int bind(int *socket*, const struct sockaddr \**address*,  
5118                socklen\_t *address\_len*);5119 **DESCRIPTION**5120        The *bind()* function shall assign a local socket address *address* to a socket identified by descriptor  
5121        *socket* that has no local socket address assigned. Sockets created with the *socket()* function are  
5122        initially unnamed; they are identified only by their address family.5123        The *bind()* function takes the following arguments:5124        *socket*                Specifies the file descriptor of the socket to be bound.5125        *address*               Points to a **sockaddr** structure containing the address to be bound to the  
5126        socket. The length and format of the address depend on the address family of  
5127        the socket.5128        *address\_len*         Specifies the length of the **sockaddr** structure pointed to by the *address*  
5129        argument.5130        The socket specified by *socket* may require the process to have appropriate privileges to use the  
5131        *bind()* function.5132 **RETURN VALUE**5133        Upon successful completion, *bind()* shall return 0; otherwise,  $-1$  shall be returned and *errno* set  
5134        to indicate the error.5135 **ERRORS**5136        The *bind()* function shall fail if:

5137        [EADDRINUSE]

5138                The specified address is already in use.

5139        [EADDRNOTAVAIL]

5140                The specified address is not available from the local machine.

5141        [EAFNOSUPPORT]

5142                The specified address is not a valid address for the address family of the  
5143        specified socket.5144        [EBADF]                The *socket* argument is not a valid file descriptor.5145        [EINVAL]               The socket is already bound to an address, and the protocol does not support  
5146        binding to a new address; or the socket has been shut down.5147        [ENOTSOCK]            The *socket* argument does not refer to a socket.5148        [EOPNOTSUPP]         The socket type of the specified socket does not support binding to an  
5149        address.5150        If the address family of the socket is AF\_UNIX, then *bind()* shall fail if:5151        [EACCES]               A component of the path prefix denies search permission, or the requested  
5152        name requires writing in a directory with a mode that denies write  
5153        permission.

5154	[EDESTADDRREQ] or [EISDIR]	
5155		The <i>address</i> argument is a null pointer.
5156	[EIO]	An I/O error occurred.
5157	[ELOOP]	A loop exists in symbolic links encountered during resolution of the pathname
5158		in <i>address</i> .
5159	[ENAMETOOLONG]	
5160		A component of a pathname exceeded {NAME_MAX} characters, or an entire
5161		pathname exceeded {PATH_MAX} characters.
5162	[ENOENT]	A component of the pathname does not name an existing file or the pathname
5163		is an empty string.
5164	[ENOTDIR]	A component of the path prefix of the pathname in <i>address</i> is not a directory.
5165	[EROFS]	The name would reside on a read-only file system.
5166		The <i>bind()</i> function may fail if:
5167	[EACCES]	The specified address is protected and the current user does not have
5168		permission to bind to it.
5169	[EINVAL]	The <i>address_len</i> argument is not a valid length for the address family.
5170	[EISCONN]	The socket is already connected.
5171	[ELOOP]	More than {SYMLOOP_MAX} symbolic links were encountered during
5172		resolution of the pathname in <i>address</i> .
5173	[ENAMETOOLONG]	
5174		Pathname resolution of a symbolic link produced an intermediate result
5175		whose length exceeds {PATH_MAX}.
5176	[ENOBUFS]	Insufficient resources were available to complete the call.
5177	<b>EXAMPLES</b>	
5178		None.
5179	<b>APPLICATION USAGE</b>	
5180		An application program can retrieve the assigned socket name with the <i>getsockname()</i> function.
5181	<b>RATIONALE</b>	
5182		None.
5183	<b>FUTURE DIRECTIONS</b>	
5184		None.
5185	<b>SEE ALSO</b>	
5186		<i>connect()</i> , <i>getsockname()</i> , <i>listen()</i> , <i>socket()</i> , the Base Definitions volume of IEEE Std 1003.1-200x,
5187		<sys/socket.h>
5188	<b>CHANGE HISTORY</b>	
5189		First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

5190 **NAME**

5191        bsd\_signal — simplified signal facilities

5192 **SYNOPSIS**

5193 OB XSI   #include &lt;signal.h&gt;

5194        void (\*bsd\_signal(int sig, void (\*func)(int)))(int);

5195

5196 **DESCRIPTION**5197        The *bsd\_signal()* function provides a partially compatible interface for programs written to  
5198        historical system interfaces (see APPLICATION USAGE).5199        The function call *bsd\_signal(sig, func)* shall be equivalent to the following:

5200        void (\*bsd\_signal(int sig, void (\*func)(int)))(int)

5201        {

5202            struct sigaction act, oact;

5203            act.sa\_handler = *func*;

5204            act.sa\_flags = SA\_RESTART;

5205            sigemptyset(&amp;act.sa\_mask);

5206            sigaddset(&act.sa\_mask, *sig*);5207            if (sigaction(*sig*, &act, &oact) == -1)

5208                return(SIG\_ERR);

5209            return(oact.sa\_handler);

5210        }

5211        The handler function should be declared:

5212        void handler(int *sig*);5213        where *sig* is the signal number. The behavior is undefined if *func* is a function that takes more  
5214        than one argument, or an argument of a different type.5215 **RETURN VALUE**5216        Upon successful completion, *bsd\_signal()* shall return the previous action for *sig*. Otherwise,  
5217        SIG\_ERR shall be returned and *errno* shall be set to indicate the error.5218 **ERRORS**5219        Refer to *sigaction()*.5220 **EXAMPLES**

5221        None.

5222 **APPLICATION USAGE**5223        This function is a direct replacement for the BSD *signal()* function for simple applications that  
5224        are installing a single-argument signal handler function. If a BSD signal handler function is being  
5225        installed that expects more than one argument, the application has to be modified to use  
5226        *sigaction()*. The *bsd\_signal()* function differs from *signal()* in that the SA\_RESTART flag is set  
5227        and the SA\_RESETHAND is clear when *bsd\_signal()* is used. The state of these flags is not  
5228        specified for *signal()*.5229        It is recommended that new applications use the *sigaction()* function.5230 **RATIONALE**

5231        None.

5232 **FUTURE DIRECTIONS**

5233       None.

5234 **SEE ALSO**5235       *sigaction()*, *sigaddset()*, *sigemptyset()*, *signal()*, the Base Definitions volume of  
5236       IEEE Std 1003.1-200x, <**signal.h**>5237 **CHANGE HISTORY**

5238       First released in Issue 4, Version 2.

5239 **Issue 5**

5240       Moved from X/OPEN UNIX extension to BASE.

5241 **Issue 6**

5242       This function is marked obsolescent.

5243 **NAME**5244       **bsearch** — binary search a sorted table5245 **SYNOPSIS**

5246       #include &lt;stdlib.h&gt;

5247       void \*bsearch(const void \*key, const void \*base, size\_t nel,  
5248                   size\_t width, int (\*compar)(const void \*, const void \*));5249 **DESCRIPTION**5250 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
5251 conflict between the requirements described here and the ISO C standard is unintentional. This  
5252 volume of IEEE Std 1003.1-200x defers to the ISO C standard.5253       The *bsearch()* function shall search an array of *nel* objects, the initial element of which is pointed  
5254 to by *base*, for an element that matches the object pointed to by *key*. The size of each element in  
5255 the array is specified by *width*.5256       The comparison function pointed to by *compar* shall be called with two arguments that point to  
5257 the *key* object and to an array element, in that order.5258       The application shall ensure that the function returns an integer less than, equal to, or greater  
5259 than 0 if the *key* object is considered, respectively, to be less than, to match, or to be greater than  
5260 the array element. The application shall ensure that the array consists of all the elements that  
5261 compare less than, all the elements that compare equal to, and all the elements that compare  
5262 greater than the *key* object, in that order.5263 **RETURN VALUE**5264       The *bsearch()* function shall return a pointer to a matching member of the array, or a null pointer  
5265 if no match is found. If two or more members compare equal, which member is returned is  
5266 unspecified.5267 **ERRORS**

5268       No errors are defined.

5269 **EXAMPLES**5270       The example below searches a table containing pointers to nodes consisting of a string and its  
5271 length. The table is ordered alphabetically on the string in the node pointed to by each entry.5272       The code fragment below reads in strings and either finds the corresponding node and prints out  
5273 the string and its length, or prints an error message.5274       #include <stdio.h>  
5275       #include <stdlib.h>  
5276       #include <string.h>  
  
5277       #define TABSIZE       1000  
  
5278       struct node {   /\* These are stored in the table. \*/  
5279           char \*string;  
5280           int length;  
5281       };  
5282       struct node table[TABSIZE];       /\* Table to be searched. \*/  
5283       .  
5284       .  
5285       .  
5286       {  
5287           struct node \*node\_ptr, node;  
5288           /\* routine to compare 2 nodes \*/

```

5289     int node_compare(const void *, const void *);
5290     char str_space[20]; /* Space to read string into. */
5291     .
5292     .
5293     .
5294     node.string = str_space;
5295     while (scanf("%s", node.string) != EOF) {
5296         node_ptr = (struct node *)bsearch((void *)&node,
5297             (void *)table, TABSIZE,
5298             sizeof(struct node), node_compare);
5299         if (node_ptr != NULL) {
5300             (void)printf("string = %20s, length = %d\n",
5301                 node_ptr->string, node_ptr->length);
5302         } else {
5303             (void)printf("not found: %s\n", node.string);
5304         }
5305     }
5306 }
5307 /*
5308     This routine compares two nodes based on an
5309     alphabetical ordering of the string field.
5310 */
5311 int
5312 node_compare(const void *node1, const void *node2)
5313 {
5314     return strcoll(((const struct node *)node1)->string,
5315         ((const struct node *)node2)->string);
5316 }

```

**5317 APPLICATION USAGE**

5318 The pointers to the key and the element at the base of the table should be of type pointer-to-  
5319 element.

5320 The comparison function need not compare every byte, so arbitrary data may be contained in  
5321 the elements in addition to the values being compared.

5322 In practice, the array is usually sorted according to the comparison function.

**5323 RATIONALE**

5324 None.

**5325 FUTURE DIRECTIONS**

5326 None.

**5327 SEE ALSO**

5328 *hcreate()*, *lsearch()*, *qsort()*, *tsearch()*, the Base Definitions volume of IEEE Std 1003.1-200x,  
5329 <stdlib.h>

**5330 CHANGE HISTORY**

5331 First released in Issue 1. Derived from Issue 1 of the SVID.

**5332 Issue 6**

5333 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

5334 **NAME**

5335       btowc — single byte to wide character conversion

5336 **SYNOPSIS**

5337       #include &lt;stdio.h&gt;

5338       #include &lt;wchar.h&gt;

5339       wint\_t btowc(int c);

5340 **DESCRIPTION**

5341 cx     The functionality described on this reference page is aligned with the ISO C standard. Any  
5342     conflict between the requirements described here and the ISO C standard is unintentional. This  
5343     volume of IEEE Std 1003.1-200x defers to the ISO C standard.

5344     The *btowc()* function shall determine whether *c* constitutes a valid (one-byte) character in the  
5345     initial shift state.

5346     The behavior of this function shall be affected by the *LC\_CTYPE* category of the current locale.

5347 **RETURN VALUE**

5348     The *btowc()* function shall return WEOF if *c* has the value EOF or if (**unsigned char**) *c* does not  
5349     constitute a valid (one-byte) character in the initial shift state. Otherwise, it shall return the  
5350     wide-character representation of that character.

5351 **ERRORS**

5352       No errors are defined.

5353 **EXAMPLES**

5354       None.

5355 **APPLICATION USAGE**

5356       None.

5357 **RATIONALE**

5358       None.

5359 **FUTURE DIRECTIONS**

5360       None.

5361 **SEE ALSO**5362       *wctob()*, the Base Definitions volume of IEEE Std 1003.1-200x, <wchar.h>5363 **CHANGE HISTORY**

5364       First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995  
5365       (E).

5366 **NAME**5367       **bzero** — memory operations (**LEGACY**)5368 **SYNOPSIS**

5369 XSI       #include &lt;strings.h&gt;

5370       void bzero(void \*s, size\_t n);

5371

5372 **DESCRIPTION**5373       The *bzero()* function shall place *n* zero-valued bytes in the area pointed to by *s*.5374 **RETURN VALUE**5375       The *bzero()* function shall not return a value.5376 **ERRORS**

5377       No errors are defined.

5378 **EXAMPLES**

5379       None.

5380 **APPLICATION USAGE**5381       *memset()* is preferred over this function.5382       For maximum portability, it is recommended to replace the function call to *bzero()* as follows:

5383       #define bzero(b,len) (memset((b), '\0', (len)), (void) 0)

5384 **RATIONALE**

5385       None.

5386 **FUTURE DIRECTIONS**

5387       This function may be withdrawn in a future version.

5388 **SEE ALSO**5389       *memset()*, the Base Definitions volume of IEEE Std 1003.1-200x, <**strings.h**>5390 **CHANGE HISTORY**

5391       First released in Issue 4, Version 2.

5392 **Issue 5**

5393       Moved from X/OPEN UNIX extension to BASE.

5394 **Issue 6**

5395       This function is marked LEGACY.

5396 **NAME**

5397       cabs, cabsf, cabsl — return a complex absolute value

5398 **SYNOPSIS**

5399       #include &lt;complex.h&gt;

5400       double cabs(double complex *z*);5401       float cabsf(float complex *z*);5402       long double cabsl(long double complex *z*);5403 **DESCRIPTION**5404 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
5405 conflict between the requirements described here and the ISO C standard is unintentional. This  
5406 volume of IEEE Std 1003.1-200x defers to the ISO C standard.5407       These functions shall compute the complex absolute value (also called norm, modulus, or  
5408 magnitude) of *z*.5409 **RETURN VALUE**

5410       These functions shall return the complex absolute value.

5411 **ERRORS**

5412       No errors are defined.

5413 **EXAMPLES**

5414       None.

5415 **APPLICATION USAGE**

5416       None.

5417 **RATIONALE**

5418       None.

5419 **FUTURE DIRECTIONS**

5420       None.

5421 **SEE ALSO**

5422       The Base Definitions volume of IEEE Std 1003.1-200x, &lt;complex.h&gt;

5423 **CHANGE HISTORY**

5424       First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

5425 **NAME**

5426 cacos, cacosf, cacosl — complex arc cosine functions

5427 **SYNOPSIS**

5428 #include &lt;complex.h&gt;

5429 double complex cacos(double complex z);

5430 float complex cacosf(float complex z);

5431 long double complex cacosl(long double complex z);

5432 **DESCRIPTION**

5433 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
5434 conflict between the requirements described here and the ISO C standard is unintentional. This  
5435 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

5436 These functions shall compute the complex arc cosine of  $z$ , with branch cuts outside the interval  
5437  $[-1, +1]$  along the real axis.

5438 **RETURN VALUE**

5439 These functions shall return the complex arc cosine value, in the range of a strip mathematically  
5440 unbounded along the imaginary axis and in the interval  $[0, \pi]$  along the real axis.

5441 **ERRORS**

5442 No errors are defined.

5443 **EXAMPLES**

5444 None.

5445 **APPLICATION USAGE**

5446 None.

5447 **RATIONALE**

5448 None.

5449 **FUTURE DIRECTIONS**

5450 None.

5451 **SEE ALSO**5452 *ccos()*, the Base Definitions volume of IEEE Std 1003.1-200x, <complex.h>5453 **CHANGE HISTORY**

5454 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

5455 **NAME**

5456       cacosf — complex arc cosine functions

5457 **SYNOPSIS**

5458       #include &lt;complex.h&gt;

5459       float complex cacosf(float complex z);

5460 **DESCRIPTION**5461       Refer to *cacos()*.

5462 **NAME**

5463 cacosh, cacoshf, cacoshl — complex arc hyperbolic cosine functions

5464 **SYNOPSIS**

5465 #include &lt;complex.h&gt;

5466 double complex cacosh(double complex z);

5467 float complex cacoshf(float complex z);

5468 long double complex cacoshl(long double complex z);

5469 **DESCRIPTION**

5470 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
5471 conflict between the requirements described here and the ISO C standard is unintentional. This  
5472 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

5473 These functions shall compute the complex arc hyperbolic cosine of  $z$ , with a branch cut at  
5474 values less than 1 along the real axis.

5475 **RETURN VALUE**

5476 These functions shall return the complex arc hyperbolic cosine value, in the range of a half-strip  
5477 of non-negative values along the real axis and in the interval  $[-i\pi, +i\pi]$  along the imaginary axis.

5478 **ERRORS**

5479 No errors are defined.

5480 **EXAMPLES**

5481 None.

5482 **APPLICATION USAGE**

5483 None.

5484 **RATIONALE**

5485 None.

5486 **FUTURE DIRECTIONS**

5487 None.

5488 **SEE ALSO**5489 *ccosh()*, the Base Definitions volume of IEEE Std 1003.1-200x, <complex.h>5490 **CHANGE HISTORY**

5491 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

5492 **NAME**

5493       cacosl — complex arc cosine functions

5494 **SYNOPSIS**

5495       #include &lt;complex.h&gt;

5496       long double complex cacosl(long double complex z);

5497 **DESCRIPTION**5498       Refer to *acos()*.

5499 **NAME**

5500        calloc — a memory allocator

5501 **SYNOPSIS**

5502        #include &lt;stdlib.h&gt;

5503        void \*calloc(size\_t *nelem*, size\_t *elsize*);5504 **DESCRIPTION**

5505 **CX**        The functionality described on this reference page is aligned with the ISO C standard. Any  
 5506 conflict between the requirements described here and the ISO C standard is unintentional. This  
 5507 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

5508        The *calloc()* function shall allocate unused space for an array of *nelem* elements each of whose  
 5509 size in bytes is *elsize*. The space shall be initialized to all bits 0.

5510        The order and contiguity of storage allocated by successive calls to *calloc()* is unspecified. The  
 5511 pointer returned if the allocation succeeds shall be suitably aligned so that it may be assigned to  
 5512 a pointer to any type of object and then used to access such an object or an array of such objects  
 5513 in the space allocated (until the space is explicitly freed or reallocated). Each such allocation  
 5514 shall yield a pointer to an object disjoint from any other object. The pointer returned shall point  
 5515 to the start (lowest byte address) of the allocated space. If the space cannot be allocated, a null  
 5516 pointer shall be returned. If the size of the space requested is 0, the behavior is implementation-  
 5517 defined: the value returned shall be either a null pointer or a unique pointer.

5518 **RETURN VALUE**

5519        Upon successful completion with both *nelem* and *elsize* non-zero, *calloc()* shall return a pointer to  
 5520 the allocated space. If either *nelem* or *elsize* is 0, then either a null pointer or a unique pointer  
 5521 value that can be successfully passed to *free()* shall be returned. Otherwise, it shall return a null  
 5522 **CX** pointer and set *errno* to indicate the error.

5523 **ERRORS**5524        The *calloc()* function shall fail if:5525 **CX**        [ENOMEM]        Insufficient memory is available.5526 **EXAMPLES**

5527        None.

5528 **APPLICATION USAGE**

5529        There is now no requirement for the implementation to support the inclusion of &lt;malloc.h&gt;.

5530 **RATIONALE**

5531        None.

5532 **FUTURE DIRECTIONS**

5533        None.

5534 **SEE ALSO**5535        *free()*, *malloc()*, *realloc()*, the Base Definitions volume of IEEE Std 1003.1-200x, <stdlib.h>5536 **CHANGE HISTORY**

5537        First released in Issue 1. Derived from Issue 1 of the SVID.

5538 **Issue 6**

5539        Extensions beyond the ISO C standard are now marked.

5540        The following new requirements on POSIX implementations derive from alignment with the  
 5541 Single UNIX Specification:

5542  
5543

- The setting of *errno* and the [ENOMEM] error condition are mandatory if an insufficient memory condition occurs.

5544 **NAME**5545 `carg`, `cargf`, `cargl` — complex argument functions5546 **SYNOPSIS**5547 `#include <complex.h>`5548 `double carg(double complex z);`5549 `float cargf(float complex z);`5550 `long double cargl(long double complex z);`5551 **DESCRIPTION**

5552 `CX` The functionality described on this reference page is aligned with the ISO C standard. Any  
5553 conflict between the requirements described here and the ISO C standard is unintentional. This  
5554 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

5555 These functions shall compute the argument (also called phase angle) of  $z$ , with a branch cut  
5556 along the negative real axis.

5557 **RETURN VALUE**5558 These functions shall return the value of the argument in the interval  $[-\pi, +\pi]$ .5559 **ERRORS**

5560 No errors are defined.

5561 **EXAMPLES**

5562 None.

5563 **APPLICATION USAGE**

5564 None.

5565 **RATIONALE**

5566 None.

5567 **FUTURE DIRECTIONS**

5568 None.

5569 **SEE ALSO**5570 `cimag()`, `conj()`, `cproj()`, the Base Definitions volume of IEEE Std 1003.1-200x, `<complex.h>`5571 **CHANGE HISTORY**

5572 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

5573 **NAME**

5574            casin, casinf, casinl — complex arc sine functions

5575 **SYNOPSIS**

5576            #include &lt;complex.h&gt;

5577            double complex casin(double complex z);

5578            float complex casinf(float complex z);

5579            long double complex casinl(long double complex z);

5580 **DESCRIPTION**

5581 cx        The functionality described on this reference page is aligned with the ISO C standard. Any  
5582            conflict between the requirements described here and the ISO C standard is unintentional. This  
5583            volume of IEEE Std 1003.1-200x defers to the ISO C standard.

5584            These functions shall compute the complex arc sine of  $z$ , with branch cuts outside the interval  
5585             $[-1, +1]$  along the real axis.

5586 **RETURN VALUE**

5587            These functions shall return the complex arc sine value, in the range of a strip mathematically  
5588            unbounded along the imaginary axis and in the interval  $[-\pi/2, +\pi/2]$  along the real axis.

5589 **ERRORS**

5590            No errors are defined.

5591 **EXAMPLES**

5592            None.

5593 **APPLICATION USAGE**

5594            None.

5595 **RATIONALE**

5596            None.

5597 **FUTURE DIRECTIONS**

5598            None.

5599 **SEE ALSO**5600            `csin()`, the Base Definitions volume of IEEE Std 1003.1-200x, <complex.h>5601 **CHANGE HISTORY**

5602            First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

5603 **NAME**5604 `casinf` — complex arc sine functions5605 **SYNOPSIS**5606 `#include <complex.h>`5607 `float complex casinf(float complex z);`5608 **DESCRIPTION**5609 Refer to *casin()*.

5610 **NAME**

5611           casinh, casinhf, casinhl — complex arc hyperbolic sine functions

5612 **SYNOPSIS**

5613           #include &lt;complex.h&gt;

5614           double complex casinh(double complex *z*);5615           float complex casinhf(float complex *z*);5616           long double complex casinhl(long double complex *z*);5617 **DESCRIPTION**

5618 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
5619           conflict between the requirements described here and the ISO C standard is unintentional. This  
5620           volume of IEEE Std 1003.1-200x defers to the ISO C standard.

5621           These functions shall compute the complex arc hyperbolic sine of *z*, with branch cuts outside the  
5622           interval  $[-i, +i]$  along the imaginary axis.

5623 **RETURN VALUE**

5624           These functions shall return the complex arc hyperbolic sine value, in the range of a strip  
5625           mathematically unbounded along the real axis and in the interval  $[-i\pi/2, +i\pi/2]$  along the  
5626           imaginary axis.

5627 **ERRORS**

5628           No errors are defined.

5629 **EXAMPLES**

5630           None.

5631 **APPLICATION USAGE**

5632           None.

5633 **RATIONALE**

5634           None.

5635 **FUTURE DIRECTIONS**

5636           None.

5637 **SEE ALSO**5638           `csinh()`, the Base Definitions volume of IEEE Std 1003.1-200x, <complex.h>5639 **CHANGE HISTORY**

5640           First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

5641 **NAME**5642        **casinl** — complex arc sine functions5643 **SYNOPSIS**

5644        #include &lt;complex.h&gt;

5645        long double complex casinl(long double complex *z*);5646 **DESCRIPTION**5647        Refer to *casin()*.

5648 **NAME**

5649           catan, catanf, catanl — complex arc tangent functions

5650 **SYNOPSIS**

5651           #include &lt;complex.h&gt;

5652           double complex catan(double complex z);

5653           float complex catanf(float complex z);

5654           long double complex catanl(long double complex z);

5655 **DESCRIPTION**

5656 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
5657 conflict between the requirements described here and the ISO C standard is unintentional. This  
5658 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

5659       These functions shall compute the complex arc tangent of  $z$ , with branch cuts outside the  
5660 interval  $[-i, +i]$  along the imaginary axis.

5661 **RETURN VALUE**

5662       These functions shall return the complex arc tangent value, in the range of a strip  
5663 mathematically unbounded along the imaginary axis and in the interval  $[-\pi/2, +\pi/2]$  along the  
5664 real axis.

5665 **ERRORS**

5666       No errors are defined.

5667 **EXAMPLES**

5668       None.

5669 **APPLICATION USAGE**

5670       None.

5671 **RATIONALE**

5672       None.

5673 **FUTURE DIRECTIONS**

5674       None.

5675 **SEE ALSO**

5676       ctan(), the Base Definitions volume of IEEE Std 1003.1-200x, &lt;complex.h&gt;

5677 **CHANGE HISTORY**

5678       First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

5679 **NAME**

5680       catanf — complex arc tangent functions

5681 **SYNOPSIS**

5682       #include &lt;complex.h&gt;

5683       float complex catanf(float complex z);

5684 **DESCRIPTION**5685       Refer to *catan()*.

5686 **NAME**

5687         catanh, catanhf, catanhl — complex arc hyperbolic tangent functions

5688 **SYNOPSIS**

5689         #include &lt;complex.h&gt;

5690         double complex catanh(double complex z);

5691         float complex catanhf(float complex z);

5692         long double complex catanhl(long double complex z);

5693 **DESCRIPTION**5694 **CX**         The functionality described on this reference page is aligned with the ISO C standard. Any  
5695 conflict between the requirements described here and the ISO C standard is unintentional. This  
5696 volume of IEEE Std 1003.1-200x defers to the ISO C standard.5697         These functions shall compute the complex arc hyperbolic tangent of  $z$ , with branch cuts outside  
5698 the interval  $[-1, +1]$  along the real axis.5699 **RETURN VALUE**5700         These functions shall return the complex arc hyperbolic tangent value, in the range of a strip  
5701 mathematically unbounded along the real axis and in the interval  $[-i\pi/2, +i\pi/2]$  along the  
5702 imaginary axis.5703 **ERRORS**

5704         No errors are defined.

5705 **EXAMPLES**

5706         None.

5707 **APPLICATION USAGE**

5708         None.

5709 **RATIONALE**

5710         None.

5711 **FUTURE DIRECTIONS**

5712         None.

5713 **SEE ALSO**5714         `ctanh()`, the Base Definitions volume of IEEE Std 1003.1-200x, <complex.h>5715 **CHANGE HISTORY**

5716         First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

5717 **NAME**5718        **catanl** — complex arc tangent functions5719 **SYNOPSIS**5720        `#include <complex.h>`5721        `long double complex catanl(long double complex z);`5722 **DESCRIPTION**5723        Refer to *catan()*.

5724 **NAME**

5725           catclose — close a message catalog descriptor

5726 **SYNOPSIS**

5727 XSI       #include &lt;nl\_types.h&gt;

5728       int catclose(nl\_catd catd);

5729

5730 **DESCRIPTION**5731       The *catclose()* function shall close the message catalog identified by *catd*. If a file descriptor is  
5732       used to implement the type **nl\_catd**, that file descriptor shall be closed.5733 **RETURN VALUE**5734       Upon successful completion, *catclose()* shall return 0; otherwise, -1 shall be returned, and *errno*  
5735       set to indicate the error.5736 **ERRORS**5737       The *catclose()* function may fail if:

5738       [EBADF]           The catalog descriptor is not valid.

5739       [EINTR]           The *catclose()* function was interrupted by a signal.5740 **EXAMPLES**

5741       None.

5742 **APPLICATION USAGE**

5743       None.

5744 **RATIONALE**

5745       None.

5746 **FUTURE DIRECTIONS**

5747       None.

5748 **SEE ALSO**5749       *catgets()*, *catopen()*, the Base Definitions volume of IEEE Std 1003.1-200x, <nl\_types.h>5750 **CHANGE HISTORY**

5751       First released in Issue 2.

5752 **NAME**

5753           catgets — read a program message

5754 **SYNOPSIS**

5755 XSI       #include &lt;nl\_types.h&gt;

5756       char \*catgets(nl\_catd *catd*, int *set\_id*, int *msg\_id*, const char \**s*);

5757

5758 **DESCRIPTION**

5759       The *catgets()* function shall attempt to read message *msg\_id*, in set *set\_id*, from the message  
 5760       catalog identified by *catd*. The *catd* argument is a message catalog descriptor returned from an  
 5761       earlier call to *catopen()*. The *s* argument points to a default message string which shall be  
 5762       returned by *catgets()* if it cannot retrieve the identified message.

5763       The *catgets()* function need not be reentrant. A function that is not required to be reentrant is not  
 5764       required to be thread-safe.

5765 **RETURN VALUE**

5766       If the identified message is retrieved successfully, *catgets()* shall return a pointer to an internal  
 5767       buffer area containing the null-terminated message string. If the call is unsuccessful for any  
 5768       reason, *s* shall be returned and *errno* may be set to indicate the error.

5769 **ERRORS**5770       The *catgets()* function may fail if:

- |      |           |  |
|------|-----------|--|
| 5771 | [EBADF]   | The <i>catd</i> argument is not a valid message catalog descriptor open for reading.   |
| 5772 | [EBADMSG] | The message identified by <i>set_id</i> and <i>msg_id</i> in the specified message catalog did not satisfy implementation-defined security criteria. |
| 5773 |           |  |
| 5774 | [EINTR]   | The read operation was terminated due to the receipt of a signal, and no data was transferred.   |
| 5775 |           |  |
| 5776 | [EINVAL]  | The message catalog identified by <i>catd</i> is corrupted.  |
| 5777 | [ENOMSG]  | The message identified by <i>set_id</i> and <i>msg_id</i> is not in the message catalog.   |

5778 **EXAMPLES**

5779       None.

5780 **APPLICATION USAGE**

5781       None.

5782 **RATIONALE**

5783       None.

5784 **FUTURE DIRECTIONS**

5785       None.

5786 **SEE ALSO**5787       *catclose()*, *catopen()*, the Base Definitions volume of IEEE Std 1003.1-200x, <nl\_types.h>5788 **CHANGE HISTORY**

5789       First released in Issue 2.

5790 **Issue 5**

5791       A note indicating that this function need not be reentrant is added to the DESCRIPTION.

5792 **Issue 6**

5793 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

## 5794 NAME

5795 catopen — open a message catalog

## 5796 SYNOPSIS

5797 xSI #include &lt;nl\_types.h&gt;

5798 nl\_catd catopen(const char \*name, int oflag);

5799

## 5800 DESCRIPTION

5801 The *catopen()* function shall open a message catalog and return a message catalog descriptor.  
 5802 The *name* argument specifies the name of the message catalog to be opened. If *name* contains a  
 5803 `'/'`, then *name* specifies a complete name for the message catalog. Otherwise, the environment  
 5804 variable *NLSPATH* is used with *name* substituted for the `%N` conversion specification (see the  
 5805 Base Definitions volume of IEEE Std 1003.1-200x, Chapter 8, Environment Variables). If  
 5806 *NLSPATH* exists in the environment when the process starts, then if the process has appropriate  
 5807 privileges, the behavior of *catopen()* is undefined. If *NLSPATH* does not exist in the environment,  
 5808 or if a message catalog cannot be found in any of the components specified by *NLSPATH*, then  
 5809 an implementation-defined default path shall be used. This default may be affected by the  
 5810 setting of *LC\_MESSAGES* if the value of *oflag* is *NL\_CAT\_LOCALE*, or the *LANG* environment  
 5811 variable if *oflag* is 0.

5812 A message catalog descriptor shall remain valid in a process until that process closes it, or a  
 5813 successful call to one of the *exec* functions. A change in the setting of the *LC\_MESSAGES*  
 5814 category may invalidate existing open catalogs.

5815 If a file descriptor is used to implement message catalog descriptors, the *FD\_CLOEXEC* flag  
 5816 shall be set; see <*fcntl.h*>.

5817 If the value of the *oflag* argument is 0, the *LANG* environment variable is used to locate the  
 5818 catalog without regard to the *LC\_MESSAGES* category. If the *oflag* argument is  
 5819 *NL\_CAT\_LOCALE*, the *LC\_MESSAGES* category is used to locate the message catalog (see the  
 5820 Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2, Internationalization Variables).

## 5821 RETURN VALUE

5822 Upon successful completion, *catopen()* shall return a message catalog descriptor for use on  
 5823 subsequent calls to *catgets()* and *catclose()*. Otherwise, *catopen()* shall return (*nl\_catd*) -1 and set  
 5824 *errno* to indicate the error.

## 5825 ERRORS

5826 The *catopen()* function may fail if:

5827 [EACCES] Search permission is denied for the component of the path prefix of the  
 5828 message catalog or read permission is denied for the message catalog.

5829 [EMFILE] {OPEN\_MAX} file descriptors are currently open in the calling process.

5830 [ENAMETOOLONG]

5831 The length of a pathname of the message catalog exceeds {PATH\_MAX} or a  
 5832 pathname component is longer than {NAME\_MAX}.

5833 [ENAMETOOLONG]

5834 Pathname resolution of a symbolic link produced an intermediate result  
 5835 whose length exceeds {PATH\_MAX}.

5836 [ENFILE] Too many files are currently open in the system.

5837 [ENOENT] The message catalog does not exist or the *name* argument points to an empty  
 5838 string.

- 5839 [ENOMEM] Insufficient storage space is available.  
5840 [ENOTDIR] A component of the path prefix of the message catalog is not a directory.

**5841 EXAMPLES**

5842 None.

**5843 APPLICATION USAGE**

5844 Some implementations of *catopen()* use *malloc()* to allocate space for internal buffer areas. The  
5845 *catopen()* function may fail if there is insufficient storage space available to accommodate these  
5846 buffers.

5847 Conforming applications must assume that message catalog descriptors are not valid after a call  
5848 to one of the *exec* functions.

5849 Application writers should be aware that guidelines for the location of message catalogs have  
5850 not yet been developed. Therefore they should take care to avoid conflicting with catalogs used  
5851 by other applications and the standard utilities.

**5852 RATIONALE**

5853 None.

**5854 FUTURE DIRECTIONS**

5855 None.

**5856 SEE ALSO**

5857 *catclose()*, *catgets()*, the Base Definitions volume of IEEE Std 1003.1-200x, `<fcntl.h>`,  
5858 `<nl_types.h>`, the Shell and Utilities volume of IEEE Std 1003.1-200x

**5859 CHANGE HISTORY**

5860 First released in Issue 2.

5861 **NAME**

5862           cbrt, cbrtf, cbrtl — cube root functions

5863 **SYNOPSIS**

5864           #include &lt;math.h&gt;

5865           double cbrt(double x);

5866           float cbrtf(float x);

5867           long double cbrtl(long double x);

5868 **DESCRIPTION**

5869 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
5870 conflict between the requirements described here and the ISO C standard is unintentional. This  
5871 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

5872       These functions shall compute the real cube root of their argument *x*.5873 **RETURN VALUE**5874       Upon successful completion, these functions shall return the cube root of *x*.5875 **MX**       If *x* is NaN, a NaN shall be returned.5876       If *x* is  $\pm 0$ , or  $\pm \text{Inf}$ , *x* shall be returned.5877 **ERRORS**

5878       No errors are defined.

5879 **EXAMPLES**

5880       None.

5881 **APPLICATION USAGE**

5882       None.

5883 **RATIONALE**

5884       For some applications, a true cube root function, which returns negative results for negative  
5885 arguments, is more appropriate than *pow(x, 1.0/3.0)*, which returns a NaN for *x* less than 0.

5886 **FUTURE DIRECTIONS**

5887       None.

5888 **SEE ALSO**

5889       The Base Definitions volume of IEEE Std 1003.1-200x, &lt;math.h&gt;

5890 **CHANGE HISTORY**

5891       First released in Issue 4, Version 2.

5892 **Issue 5**

5893       Moved from X/OPEN UNIX extension to BASE.

5894 **Issue 6**5895       The *cbrt()* function is no longer marked as an extension.5896       The *cbrtf()* and *cbrtl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

5897       The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
5898 revised to align with the ISO/IEC 9899:1999 standard.

5899       IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
5900 marked.

5901 **NAME**

5902           ccos, ccosf, ccosl — complex cosine functions

5903 **SYNOPSIS**

5904           #include &lt;complex.h&gt;

5905           double complex ccos(double complex z);

5906           float complex ccosf(float complex z);

5907           long double complex ccosl(long double complex z);

5908 **DESCRIPTION**5909 cx       The functionality described on this reference page is aligned with the ISO C standard. Any  
5910       conflict between the requirements described here and the ISO C standard is unintentional. This  
5911       volume of IEEE Std 1003.1-200x defers to the ISO C standard.5912       These functions shall compute the complex cosine of *z*.5913 **RETURN VALUE**

5914       These functions shall return the complex cosine value.

5915 **ERRORS**

5916       No errors are defined.

5917 **EXAMPLES**

5918       None.

5919 **APPLICATION USAGE**

5920       None.

5921 **RATIONALE**

5922       None.

5923 **FUTURE DIRECTIONS**

5924       None.

5925 **SEE ALSO**

5926       ccos(), the Base Definitions volume of IEEE Std 1003.1-200x, &lt;complex.h&gt;

5927 **CHANGE HISTORY**

5928       First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

5929 **NAME**5930 `ccosf` — complex cosine functions5931 **SYNOPSIS**5932 `#include <complex.h>`5933 `float complex ccosf(float complex z);`5934 **DESCRIPTION**5935 Refer to `ccos()`.

5936 **NAME**

5937 ccosh, ccoshf, ccoshl — complex hyperbolic cosine functions

5938 **SYNOPSIS**

5939 #include &lt;complex.h&gt;

5940 double complex ccosh(double complex z);

5941 float complex ccoshf(float complex z);

5942 long double complex ccoshl(long double complex z);

5943 **DESCRIPTION**

5944 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
5945 conflict between the requirements described here and the ISO C standard is unintentional. This  
5946 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

5947 These functions shall compute the complex hyperbolic cosine of  $z$ .5948 **RETURN VALUE**

5949 These functions shall return the complex hyperbolic cosine value.

5950 **ERRORS**

5951 No errors are defined.

5952 **EXAMPLES**

5953 None.

5954 **APPLICATION USAGE**

5955 None.

5956 **RATIONALE**

5957 None.

5958 **FUTURE DIRECTIONS**

5959 None.

5960 **SEE ALSO**5961 *cacosh()*, the Base Definitions volume of IEEE Std 1003.1-200x, <complex.h>5962 **CHANGE HISTORY**

5963 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

5964 **NAME**

5965       ccosl — complex cosine functions

5966 **SYNOPSIS**

5967       #include &lt;complex.h&gt;

5968       long double complex ccosl(long double complex z);

5969 **DESCRIPTION**5970       Refer to *ccos()*.

5971 **NAME**

5972       ceil, ceilf, ceill — ceiling value function

5973 **SYNOPSIS**

5974       #include &lt;math.h&gt;

5975       double ceil(double x);

5976       float ceilf(float x);

5977       long double ceill(long double x);

5978 **DESCRIPTION**5979 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
5980 conflict between the requirements described here and the ISO C standard is unintentional. This  
5981 volume of IEEE Std 1003.1-200x defers to the ISO C standard.5982       These functions shall compute the smallest integral value not less than *x*.5983       An application wishing to check for error situations should set *errno* to zero and call  
5984 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
5985 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
5986 zero, an error has occurred.5987 **RETURN VALUE**5988       Upon successful completion, *ceil()*, *ceilf()*, and *ceill()* shall return the smallest integral value not  
5989 less than *x*, expressed as a type **double**, **float**, or **long double**, respectively.5990 **MX**       If *x* is NaN, a NaN shall be returned.5991       If *x* is  $\pm 0$ , or  $\pm \text{Inf}$ , *x* shall be returned.5992 **XSI**       If the correct value would cause overflow, a range error shall occur and *ceil()*, *ceilf()*, and *ceill()*  
5993 shall return the value of the macro HUGE\_VAL, HUGE\_VALF, and HUGE\_VALL, respectively.5994 **ERRORS**

5995       These functions shall fail if:

5996 **XSI**       **Range Error**       The result overflows.5997       If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero, |  
5998 then *errno* shall be set to [ERANGE]. If the integer expression |  
5999 (math\_errhandling & MATH\_ERREXCEPT) is non-zero, then the overflow |  
6000 floating-point exception shall be raised. |6001 **EXAMPLES**

6002       None.

6003 **APPLICATION USAGE**6004       The integral value returned by these functions need not be expressible as an **int** or **long**. The  
6005 return value should be tested before assigning it to an integer type to avoid the undefined results  
6006 of an integer overflow.6007       The *ceil()* function can only overflow when the floating-point representation has  
6008 DBL\_MANT\_DIG > DBL\_MAX\_EXP.6009       On error, the expressions (math\_errhandling & MATH\_ERRNO) and (math\_errhandling &  
6010 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

6011 **RATIONALE**

6012       None.

6013 **FUTURE DIRECTIONS**

6014       None.

6015 **SEE ALSO**6016       *feclearexcept()*, *fetestexcept()*, *floor()*, *isnan()*, the Base Definitions volume of IEEE Std 1003.1-200x, |

6017       Section 4.18, Treatment of Error Conditions for Mathematical Functions, &lt;math.h&gt; |

6018 **CHANGE HISTORY**

6019       First released in Issue 1. Derived from Issue 1 of the SVID.

6020 **Issue 5**6021       The DESCRIPTION is updated to indicate how an application should check for an error. This  
6022       text was previously published in the APPLICATION USAGE section.6023 **Issue 6**6024       The *ceilf()* and *ceilll()* functions are added for alignment with the ISO/IEC 9899:1999 standard.6025       The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
6026       revised to align with the ISO/IEC 9899:1999 standard.6027       IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
6028       marked.

6029 **NAME**

6030 cexp, cexpf, cexpl — complex exponential functions

6031 **SYNOPSIS**

6032 #include &lt;complex.h&gt;

6033 double complex cexp(double complex z);

6034 float complex cexpf(float complex z);

6035 long double complex cexpl(long double complex z);

6036 **DESCRIPTION**6037 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
6038 conflict between the requirements described here and the ISO C standard is unintentional. This  
6039 volume of IEEE Std 1003.1-200x defers to the ISO C standard.6040 These functions shall compute the complex exponent of  $z$ , defined as  $e^z$ .6041 **RETURN VALUE**6042 These functions shall return the complex exponential value of  $z$ .6043 **ERRORS**

6044 No errors are defined.

6045 **EXAMPLES**

6046 None.

6047 **APPLICATION USAGE**

6048 None.

6049 **RATIONALE**

6050 None.

6051 **FUTURE DIRECTIONS**

6052 None.

6053 **SEE ALSO**6054 *clog()*, the Base Definitions volume of IEEE Std 1003.1-200x, <complex.h>6055 **CHANGE HISTORY**

6056 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

6057 **NAME**

6058 cfgetispeed — get input baud rate

6059 **SYNOPSIS**

6060 #include &lt;termios.h&gt;

6061 speed\_t cfgetispeed(const struct termios \*termios\_p);

6062 **DESCRIPTION**6063 The *cfgetispeed()* function shall extract the input baud rate from the **termios** structure to which  
6064 the *termios\_p* argument points.6065 This function shall return exactly the value in the **termios** data structure, without interpretation.6066 **RETURN VALUE**6067 Upon successful completion, *cfgetispeed()* shall return a value of type **speed\_t** representing the  
6068 input baud rate.6069 **ERRORS**

6070 No errors are defined.

6071 **EXAMPLES**

6072 None.

6073 **APPLICATION USAGE**

6074 None.

6075 **RATIONALE**6076 The term *baud* is used historically here, but is not technically correct. This is properly “bits per  
6077 second”, which may not be the same as baud. However, the term is used because of the  
6078 historical usage and understanding.6079 The *cfgetospeed()*, *cfgetispeed()*, *cfsetospeed()*, and *cfsetispeed()* functions do not take arguments as  
6080 numbers, but rather as symbolic names. There are two reasons for this:

- 6081 1. Historically, numbers were not used because of the way the rate was stored in the data
- 
- 6082 structure. This is retained even though a function is now used.
- 
- 6083 2. More importantly, only a limited set of possible rates is at all portable, and this constrains
- 
- 6084 the application to that set.

6085 There is nothing to prevent an implementation to accept, as an extension, a number (such as 126)  
6086 if it wished, and because the encoding of the Bxxx symbols is not specified, this can be done so  
6087 no ambiguity is introduced.6088 Setting the input baud rate to zero was a mechanism to allow for split baud rates. Clarifications  
6089 in this volume of IEEE Std 1003.1-200x have made it possible to determine whether split rates are  
6090 supported and to support them without having to treat zero as a special case. Since this  
6091 functionality is also confusing, it has been declared obsolescent. The 0 argument referred to is  
6092 the literal constant 0, not the symbolic constant B0. This volume of IEEE Std 1003.1-200x does  
6093 not preclude B0 from being defined as the value 0; in fact, implementations would likely benefit  
6094 from the two being equivalent. This volume of IEEE Std 1003.1-200x does not fully specify  
6095 whether the previous *cfsetispeed()* value is retained after a *tcgetattr()* as the actual value or as  
6096 zero. Therefore, conforming applications should always set both the input speed and output  
6097 speed when setting either.6098 In historical implementations, the baud rate information is traditionally kept in **c\_cflag**.  
6099 Applications should be written to presume that this might be the case (and thus not blindly copy  
6100 **c\_cflag**), but not to rely on it in case it is in some other field of the structure. Setting the **c\_cflag**  
6101 field absolutely after setting a baud rate is a non-portable action because of this. In general, the

6102 unused parts of the flag fields might be used by the implementation and should not be blindly  
6103 copied from the descriptions of one terminal device to another.

6104 **FUTURE DIRECTIONS**

6105 None.

6106 **SEE ALSO**

6107 *cfgetospeed()*, *cfsetispeed()*, *cfsetospeed()*, *tcgetattr()*, the Base Definitions volume of  
6108 IEEE Std 1003.1-200x, <**termios.h**>, the Base Definitions volume of IEEE Std 1003.1-200x,  
6109 Chapter 11, General Terminal Interface

6110 **CHANGE HISTORY**

6111 First released in Issue 3.

6112 Entry included for alignment with the POSIX.1-1988 standard.

6113 **NAME**

6114 cfgetospeed — get output baud rate

6115 **SYNOPSIS**

6116 #include &lt;termios.h&gt;

6117 speed\_t cfgetospeed(const struct termios \*termios\_p);

6118 **DESCRIPTION**6119 The *cfgetospeed()* function shall extract the output baud rate from the **termios** structure to which  
6120 the *termios\_p* argument points.6121 This function shall return exactly the value in the **termios** data structure, without interpretation.6122 **RETURN VALUE**6123 Upon successful completion, *cfgetospeed()* shall return a value of type **speed\_t** representing the  
6124 output baud rate.6125 **ERRORS**

6126 No errors are defined.

6127 **EXAMPLES**

6128 None.

6129 **APPLICATION USAGE**

6130 None.

6131 **RATIONALE**6132 Refer to *cfgetispeed()*.6133 **FUTURE DIRECTIONS**

6134 None.

6135 **SEE ALSO**6136 *cfgetispeed()*, *cfsetispeed()*, *cfsetospeed()*, *tcgetattr()*, the Base Definitions volume of  
6137 IEEE Std 1003.1-200x, <**termios.h**>, the Base Definitions volume of IEEE Std 1003.1-200x,  
6138 Chapter 11, General Terminal Interface6139 **CHANGE HISTORY**

6140 First released in Issue 3.

6141 Entry included for alignment with the POSIX.1-1988 standard.

6142 **NAME**

6143 cfsetispeed — set input baud rate

6144 **SYNOPSIS**

6145 #include &lt;termios.h&gt;

6146 int cfsetispeed(struct termios \*termios\_p, speed\_t speed);

6147 **DESCRIPTION**6148 The *cfsetispeed()* function shall set the input baud rate stored in the structure pointed to by  
6149 *termios\_p* to *speed*.6150 There shall be no effect on the baud rates set in the hardware until a subsequent successful call  
6151 to *tcsetattr()* with the same **termios** structure. Similarly, errors resulting from attempts to set  
6152 baud rates not supported by the terminal device need not be detected until the *tcsetattr()*  
6153 function is called.6154 **RETURN VALUE**6155 Upon successful completion, *cfsetispeed()* shall return 0; otherwise, -1 shall be returned, and  
6156 *errno* may be set to indicate the error.6157 **ERRORS**6158 The *cfsetispeed()* function may fail if:6159 [EINVAL] The *speed* value is not a valid baud rate.6160 [EINVAL] The value of *speed* is outside the range of possible speed values as specified in  
6161 <**termios.h**>.6162 **EXAMPLES**

6163 None.

6164 **APPLICATION USAGE**

6165 None.

6166 **RATIONALE**6167 Refer to *cfgetispeed()*.6168 **FUTURE DIRECTIONS**

6169 None.

6170 **SEE ALSO**6171 *cfgetispeed()*, *cfgetospeed()*, *cfsetospeed()*, *tcsetattr()*, the Base Definitions volume of  
6172 IEEE Std 1003.1-200x, <**termios.h**>, the Base Definitions volume of IEEE Std 1003.1-200x,  
6173 Chapter 11, General Terminal Interface6174 **CHANGE HISTORY**

6175 First released in Issue 3.

6176 Entry included for alignment with the POSIX.1-1988 standard.

6177 **Issue 6**6178 The following new requirements on POSIX implementations derive from alignment with the  
6179 Single UNIX Specification:

- 6180
- The optional setting of *errno* and the [EINVAL] error conditions are added.

6181 **NAME**

6182 cfsetospeed — set output baud rate

6183 **SYNOPSIS**

6184 #include &lt;termios.h&gt;

6185 int cfsetospeed(struct termios \*termios\_p, speed\_t speed);

6186 **DESCRIPTION**6187 The *cfsetospeed()* function shall set the output baud rate stored in the structure pointed to by  
6188 *termios\_p* to *speed*.6189 There shall be no effect on the baud rates set in the hardware until a subsequent successful call  
6190 to *tcsetattr()* with the same **termios** structure. Similarly, errors resulting from attempts to set  
6191 baud rates not supported by the terminal device need not be detected until the *tcsetattr()*  
6192 function is called.6193 **RETURN VALUE**6194 Upon successful completion, *cfsetospeed()* shall return 0; otherwise, it shall return -1 and *errno*  
6195 may be set to indicate the error.6196 **ERRORS**6197 The *cfsetospeed()* function may fail if:6198 [EINVAL] The *speed* value is not a valid baud rate.6199 [EINVAL] The value of *speed* is outside the range of possible speed values as specified in  
6200 <**termios.h**>.6201 **EXAMPLES**

6202 None.

6203 **APPLICATION USAGE**

6204 None.

6205 **RATIONALE**6206 Refer to *cfgetispeed()*.6207 **FUTURE DIRECTIONS**

6208 None.

6209 **SEE ALSO**6210 *cfgetispeed()*, *cfgetospeed()*, *cfsetispeed()*, *tcsetattr()*, the Base Definitions volume of  
6211 IEEE Std 1003.1-200x, <**termios.h**>, the Base Definitions volume of IEEE Std 1003.1-200x,  
6212 Chapter 11, General Terminal Interface6213 **CHANGE HISTORY**

6214 First released in Issue 3.

6215 Entry included for alignment with the POSIX.1-1988 standard.

6216 **Issue 6**6217 The following new requirements on POSIX implementations derive from alignment with the  
6218 Single UNIX Specification:

- 6219
- The optional setting of *errno* and the [EINVAL] error conditions are added.

6220 **NAME**

6221           chdir — change working directory

6222 **SYNOPSIS**

6223           #include &lt;unistd.h&gt;

6224           int chdir(const char \*path);

6225 **DESCRIPTION**

6226           The *chdir()* function shall cause the directory named by the pathname pointed to by the *path* |  
 6227           argument to become the current working directory; that is, the starting point for path searches |  
 6228           for pathnames not beginning with '/'. |

6229 **RETURN VALUE**

6230           Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned, the current |  
 6231           working directory shall remain unchanged, and *errno* shall be set to indicate the error.

6232 **ERRORS**6233           The *chdir()* function shall fail if:

6234           [EACCES]           Search permission is denied for any component of the pathname. |

6235           [ELOOP]           A loop exists in symbolic links encountered during resolution of the *path* |  
6236           argument.

6237           [ENAMETOOLONG]

6238           The length of the *path* argument exceeds {PATH\_MAX} or a pathname |  
6239           component is longer than {NAME\_MAX}.6240           [ENOENT]           A component of *path* does not name an existing directory or *path* is an empty |  
6241           string.

6242           [ENOTDIR]           A component of the pathname is not a directory. |

6243           The *chdir()* function may fail if:6244           [ELOOP]           More than {SYMLOOP\_MAX} symbolic links were encountered during |  
6245           resolution of the *path* argument.

6246           [ENAMETOOLONG]

6247           As a result of encountering a symbolic link in resolution of the *path* argument, |  
6248           the length of the substituted pathname string exceeded {PATH\_MAX}.6249 **EXAMPLES**6250           **Changing the Current Working Directory**6251           The following example makes the value pointed to by **directory**, **/tmp**, the current working |  
6252           directory.

6253           #include &lt;unistd.h&gt;

6254           ...

6255           char \*directory = "/tmp";

6256           int ret;

6257           ret = chdir (directory);

6258 **APPLICATION USAGE**

6259 None.

6260 **RATIONALE**6261 The *chdir()* function only affects the working directory of the current process.6262 **FUTURE DIRECTIONS**

6263 None.

6264 **SEE ALSO**6265 *getcwd()*, the Base Definitions volume of IEEE Std 1003.1-200x, <**unistd.h**>6266 **CHANGE HISTORY**

6267 First released in Issue 1. Derived from Issue 1 of the SVID.

6268 **Issue 6**

6269 The APPLICATION USAGE section is added.

6270 The following new requirements on POSIX implementations derive from alignment with the |  
6271 Single UNIX Specification:

- 6272
- The [ELOOP] mandatory error condition is added.
  - A second [ENAMETOOLONG] is added as an optional error condition.

6274 The following changes were made to align with the IEEE P1003.1a draft standard:

- 6275
- The [ELOOP] optional error condition is added.

6276 **NAME**

6277 chmod — change mode of a file

6278 **SYNOPSIS**

6279 #include &lt;sys/stat.h&gt;

6280 int chmod(const char \*path, mode\_t mode);

6281 **DESCRIPTION**

6282 XSI The *chmod()* function shall change S\_ISUID, S\_ISGID, S\_ISVTX, and the file permission bits of |  
 6283 the file named by the pathname pointed to by the *path* argument to the corresponding bits in the |  
 6284 *mode* argument. The application shall ensure that the effective user ID of the process matches the |  
 6285 owner of the file or the process has appropriate privileges in order to do this.

6286 XSI S\_ISUID, S\_ISGID, S\_ISVTX, and the file permission bits are described in &lt;sys/stat.h&gt;.

6287 If the calling process does not have appropriate privileges, and if the group ID of the file does |  
 6288 not match the effective group ID or one of the supplementary group IDs and if the file is a |  
 6289 regular file, bit S\_ISGID (set-group-ID on execution) in the file's mode shall be cleared upon |  
 6290 successful return from *chmod()*.

6291 Additional implementation-defined restrictions may cause the S\_ISUID and S\_ISGID bits in |  
 6292 *mode* to be ignored.

6293 The effect on file descriptors for files open at the time of a call to *chmod()* is implementation- |  
 6294 defined.

6295 Upon successful completion, *chmod()* shall mark for update the *st\_ctime* field of the file.6296 **RETURN VALUE**

6297 Upon successful completion, 0 shall be returned; otherwise, -1 shall be returned and *errno* set to |  
 6298 indicate the error. If -1 is returned, no change to the file mode occurs.

6299 **ERRORS**6300 The *chmod()* function shall fail if:

6301 [EACCES] Search permission is denied on a component of the path prefix.

6302 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path* |  
 6303 argument.

6304 [ENAMETOOLONG]

6305 The length of the *path* argument exceeds {PATH\_MAX} or a pathname |  
 6306 component is longer than {NAME\_MAX}.

6307 [ENOTDIR] A component of the path prefix is not a directory.

6308 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.6309 [EPERM] The effective user ID does not match the owner of the file and the process |  
 6310 does not have appropriate privileges.

6311 [EROFS] The named file resides on a read-only file system.

6312 The *chmod()* function may fail if:

6313 [EINTR] A signal was caught during execution of the function.

6314 [EINVAL] The value of the *mode* argument is invalid.6315 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during |  
 6316 resolution of the *path* argument.

6317 [ENAMETOOLONG]  
6318 As a result of encountering a symbolic link in resolution of the *path* argument, |  
6319 the length of the substituted pathname strings exceeded {PATH\_MAX}. |

## 6320 EXAMPLES

### 6321 **Setting Read Permissions for User, Group, and Others**

6322 The following example sets read permissions for the owner, group, and others.

```
6323 #include <sys/stat.h>
6324 const char *path;
6325 ...
6326 chmod(path, S_IRUSR|S_IRGRP|S_IROTH);
```

### 6327 **Setting Read, Write, and Execute Permissions for the Owner Only**

6328 The following example sets read, write, and execute permissions for the owner, and no  
6329 permissions for group and others.

```
6330 #include <sys/stat.h>
6331 const char *path;
6332 ...
6333 chmod(path, S_IRWXU);
```

### 6334 **Setting Different Permissions for Owner, Group, and Other**

6335 The following example sets owner permissions for `CHANGEFILE` to read, write, and execute,  
6336 group permissions to read and execute, and other permissions to read.

```
6337 #include <sys/stat.h>
6338 #define CHANGEFILE "/etc/myfile"
6339 ...
6340 chmod(CHANGEFILE, S_IRWXU|S_IRGRP|S_IXGRP|S_IROTH);
```

### 6341 **Setting and Checking File Permissions**

6342 The following example sets the file permission bits for a file named `/home/cnd/mod1`, then calls  
6343 the `stat()` function to verify the permissions.

```
6344 #include <sys/types.h>
6345 #include <sys/stat.h>
6346 int status;
6347 struct stat buffer
6348 ...
6349 chmod("home/cnd/mod1", S_IRWXU|S_IRWXG|S_IROTH|S_IWOTH);
6350 status = stat("home/cnd/mod1", &buffer);
```

## 6351 APPLICATION USAGE

6352 In order to ensure that the `S_ISUID` and `S_ISGID` bits are set, an application requiring this should  
6353 use `stat()` after a successful `chmod()` to verify this.

6354 Any file descriptors currently open by any process on the file could possibly become invalid if  
6355 the mode of the file is changed to a value which would deny access to that process. One

6356 situation where this could occur is on a stateless file system. This behavior will not occur in a  
6357 conforming environment.

6358 **RATIONALE**

6359 This volume of IEEE Std 1003.1-200x specifies that the S\_ISGID bit is cleared by *chmod()* on a  
6360 regular file under certain conditions. This is specified on the assumption that regular files may  
6361 be executed, and the system should prevent users from making executable *setgid()* files perform  
6362 with privileges that the caller does not have. On implementations that support execution of  
6363 other file types, the S\_ISGID bit should be cleared for those file types under the same  
6364 circumstances.

6365 Implementations that use the S\_ISUID bit to indicate some other function (for example,  
6366 mandatory record locking) on non-executable files need not clear this bit on writing. They  
6367 should clear the bit for executable files and any other cases where the bit grants special powers  
6368 to processes that change the file contents. Similar comments apply to the S\_ISGID bit.

6369 **FUTURE DIRECTIONS**

6370 None.

6371 **SEE ALSO**

6372 *chown()*, *mkdir()*, *mkfifo()*, *open()*, *stat()*, *statvfs()*, the Base Definitions volume of  
6373 IEEE Std 1003.1-200x, <sys/stat.h>, <sys/types.h>

6374 **CHANGE HISTORY**

6375 First released in Issue 1. Derived from Issue 1 of the SVID.

6376 **Issue 6**

6377 The following new requirements on POSIX implementations derive from alignment with the |  
6378 Single UNIX Specification:

6379 • The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was  
6380 required for conforming implementations of previous POSIX specifications, it was not  
6381 required for UNIX applications.

6382 • The [EINVAL] and [EINTR] optional error conditions are added.

6383 • A second [ENAMETOOLONG] is added as an optional error condition.

6384 The following changes were made to align with the IEEE P1003.1a draft standard:

6385 • The [ELOOP] optional error condition is added.

6386 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

6387 **NAME**

6388 chown — change owner and group of a file

6389 **SYNOPSIS**

6390 #include &lt;unistd.h&gt;

6391 int chown(const char \*path, uid\_t owner, gid\_t group);

6392 **DESCRIPTION**6393 The *chown()* function shall change the user and group ownership of a file. |6394 The *path* argument points to a pathname naming a file. The user ID and group ID of the named |  
6395 file shall be set to the numeric values contained in *owner* and *group*, respectively. |6396 Only processes with an effective user ID equal to the user ID of the file or with appropriate |  
6397 privileges may change the ownership of a file. If `_POSIX_CHOWN_RESTRICTED` is in effect for |  
6398 *path*:

- 6399
- Changing the user ID is restricted to processes with appropriate privileges.
  - Changing the group ID is permitted to a process with an effective user ID equal to the user |  
6400 ID of the file, but without appropriate privileges, if and only if *owner* is equal to the file's user |  
6401 ID or `(uid_t)-1` and *group* is equal either to the calling process' effective group ID or to one of |  
6402 its supplementary group IDs.

6403  
6404 If the specified file is a regular file, one or more of the `S_IXUSR`, `S_IXGRP`, or `S_IXOTH` bits of |  
6405 the file mode are set, and the process does not have appropriate privileges, the set-user-ID |  
6406 (`S_ISUID`) and set-group-ID (`S_ISGID`) bits of the file mode shall be cleared upon successful |  
6407 return from *chown()*. If the specified file is a regular file, one or more of the `S_IXUSR`, `S_IXGRP`, |  
6408 or `S_IXOTH` bits of the file mode are set, and the process has appropriate privileges, it is |  
6409 implementation-defined whether the set-user-ID and set-group-ID bits are altered. If the *chown()* |  
6410 function is successfully invoked on a file that is not a regular file and one or more of the |  
6411 `S_IXUSR`, `S_IXGRP`, or `S_IXOTH` bits of the file mode are set, the set-user-ID and set-group-ID |  
6412 bits may be cleared.6413 If *owner* or *group* is specified as `(uid_t)-1` or `(gid_t)-1`, respectively, the corresponding ID of the |  
6414 file shall not be changed. If both owner and group are `-1`, the times need not be updated. |6415 Upon successful completion, *chown()* shall mark for update the *st\_ctime* field of the file.6416 **RETURN VALUE**6417 Upon successful completion, 0 shall be returned; otherwise, `-1` shall be returned and *errno* set to |  
6418 indicate the error. If `-1` is returned, no changes are made in the user ID and group ID of the file.6419 **ERRORS**6420 The *chown()* function shall fail if:

- 6421 [EACCES] Search permission is denied on a component of the path prefix.
- 
- 6422 [ELOOP] A loop exists in symbolic links encountered during resolution of the
- path*
- |
- 
- 6423 argument.
- 
- 6424 [ENAMETOOLONG] |
- 
- 6425 The length of the
- path*
- argument exceeds
- `{PATH_MAX}`
- or a pathname |
- 
- 6426 component is longer than
- `{NAME_MAX}`
- . |
- 
- 6427 [ENOTDIR] A component of the path prefix is not a directory.
- 
- 6428 [ENOENT] A component of
- path*
- does not name an existing file or
- path*
- is an empty string.

6429 [EPERM] The effective user ID does not match the owner of the file, or the calling  
 6430 process does not have appropriate privileges and  
 6431 `_POSIX_CHOWN_RESTRICTED` indicates that such privilege is required.

6432 [EROFS] The named file resides on a read-only file system.

6433 The `chown()` function may fail if:

6434 [EIO] An I/O error occurred while reading or writing to the file system.

6435 [EINTR] The `chown()` function was interrupted by a signal which was caught.

6436 [EINVAL] The owner or group ID supplied is not a value supported by the  
 6437 implementation.

6438 [ELOOP] More than `{SYMLOOP_MAX}` symbolic links were encountered during  
 6439 resolution of the *path* argument.

6440 [ENAMETOOLONG]

6441 As a result of encountering a symbolic link in resolution of the *path* argument, |  
 6442 the length of the substituted pathname string exceeded `{PATH_MAX}`. |

#### 6443 EXAMPLES

6444 None.

#### 6445 APPLICATION USAGE

6446 Although `chown()` can be used on some implementations by the file owner to change the owner |  
 6447 and group to any desired values, the only portable use of this function is to change the group of |  
 6448 a file to the effective GID of the calling process or to a member of its group set. |

#### 6449 RATIONALE

6450 System III and System V allow a user to give away files; that is, the owner of a file may change  
 6451 its user ID to anything. This is a serious problem for implementations that are intended to meet  
 6452 government security regulations. Version 7 and 4.3 BSD permit only the superuser to change the  
 6453 user ID of a file. Some government agencies (usually not ones concerned directly with security)  
 6454 find this limitation too confining. This volume of IEEE Std 1003.1-200x uses *may* to permit secure  
 6455 implementations while not disallowing System V.

6456 System III and System V allow the owner of a file to change the group ID to anything. Version 7  
 6457 permits only the superuser to change the group ID of a file. 4.3 BSD permits the owner to  
 6458 change the group ID of a file to its effective group ID or to any of the groups in the list of  
 6459 supplementary group IDs, but to no others.

6460 The POSIX.1-1990 standard requires that the `chown()` function invoked by a non-appropriate  
 6461 privileged process clear the `S_ISGID` and the `S_ISUID` bits for regular files, and permits them to  
 6462 be cleared for other types of files. This is so that changes in accessibility do not accidentally  
 6463 cause files to become security holes. Unfortunately, requiring these bits to be cleared on non-  
 6464 executable data files also clears the mandatory file locking bit (shared with `S_ISGID`), which is  
 6465 an extension on many implementations (it first appeared in System V). These bits should only be  
 6466 required to be cleared on regular files that have one or more of their execute bits set.

#### 6467 FUTURE DIRECTIONS

6468 None.

#### 6469 SEE ALSO

6470 `chmod()`, `pathconf()`, the Base Definitions volume of IEEE Std 1003.1-200x, `<sys/types.h>`,  
 6471 `<unistd.h>`

6472 **CHANGE HISTORY**

6473 First released in Issue 1. Derived from Issue 1 of the SVID.

6474 **Issue 6**

6475 The following changes are made for alignment with the ISO POSIX-1: 1996 standard:

- 6476 • The wording describing the optional dependency on `_POSIX_CHOWN_RESTRICTED` is  
6477 restored.
- 6478 • The [EPERM] error is restored as an error dependent on `_POSIX_CHOWN_RESTRICTED`. |  
6479 This is since its operand is a pathname and applications should be aware that the error may |  
6480 not occur for that pathname if the file system does not support |  
6481 `_POSIX_CHOWN_RESTRICTED`. |

6482 The following new requirements on POSIX implementations derive from alignment with the  
6483 Single UNIX Specification:

- 6484 • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was  
6485 required for conforming implementations of previous POSIX specifications, it was not  
6486 required for UNIX applications.
- 6487 • The value for *owner* of `(uid_t)-1` allows the use of `-1` by the owner of a file to change the  
6488 group ID only. A corresponding change is made for group. |
- 6489 • The [ELOOP] mandatory error condition is added.
- 6490 • The [EIO] and [EINTR] optional error conditions are added.
- 6491 • A second [ENAMETOOLONG] is added as an optional error condition.

6492 The following changes were made to align with the IEEE P1003.1a draft standard:

- 6493 • Clarification is added that the `S_ISUID` and `S_ISGID` bits do not need to be cleared when the  
6494 process has appropriate privileges.
- 6495 • The [ELOOP] optional error condition is added.

6496 **NAME**

6497        cimag, cimagf, cimagl — complex imaginary functions

6498 **SYNOPSIS**

6499        #include &lt;complex.h&gt;

6500        double cimag(double complex z);

6501        float cimagf(float complex z);

6502        long double cimagl(long double complex z);

6503 **DESCRIPTION**6504 **CX**        The functionality described on this reference page is aligned with the ISO C standard. Any  
6505 conflict between the requirements described here and the ISO C standard is unintentional. This  
6506 volume of IEEE Std 1003.1-200x defers to the ISO C standard.6507        These functions shall compute the imaginary part of *z*.6508 **RETURN VALUE**

6509        These functions shall return the imaginary part value (as a real).

6510 **ERRORS**

6511        No errors are defined.

6512 **EXAMPLES**

6513        None.

6514 **APPLICATION USAGE**6515        For a variable *z* of complex type:6516        `z == creal(z) + cimag(z)*I`6517 **RATIONALE**

6518        None.

6519 **FUTURE DIRECTIONS**

6520        None.

6521 **SEE ALSO**6522        *carg()*, *conj()*, *cproj()*, *creal()*, the Base Definitions volume of IEEE Std 1003.1-200x, <complex.h>6523 **CHANGE HISTORY**

6524        First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

6525 **NAME**

6526 clearerr — clear indicators on a stream

6527 **SYNOPSIS**

6528 #include <stdio.h>

6529 void clearerr(FILE \*stream);

6530 **DESCRIPTION**

6531 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
6532 conflict between the requirements described here and the ISO C standard is unintentional. This  
6533 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

6534 The *clearerr()* function shall clear the end-of-file and error indicators for the stream to which  
6535 *stream* points.

6536 **RETURN VALUE**

6537 The *clearerr()* function shall not return a value.

6538 **ERRORS**

6539 No errors are defined.

6540 **EXAMPLES**

6541 None.

6542 **APPLICATION USAGE**

6543 None.

6544 **RATIONALE**

6545 None.

6546 **FUTURE DIRECTIONS**

6547 None.

6548 **SEE ALSO**

6549 The Base Definitions volume of IEEE Std 1003.1-200x, <stdio.h>

6550 **CHANGE HISTORY**

6551 First released in Issue 1. Derived from Issue 1 of the SVID.

6552 **NAME**

6553 clock — report CPU time used

6554 **SYNOPSIS**

6555 #include &lt;time.h&gt;

6556 clock\_t clock(void);

6557 **DESCRIPTION**

6558 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
6559 conflict between the requirements described here and the ISO C standard is unintentional. This  
6560 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

6561 The *clock()* function shall return the implementation's best approximation to the processor time  
6562 used by the process since the beginning of an implementation-defined era related only to the  
6563 process invocation.

6564 **RETURN VALUE**

6565 To determine the time in seconds, the value returned by *clock()* should be divided by the value  
6566 **XSI** of the macro `CLOCKS_PER_SEC`. `CLOCKS_PER_SEC` is defined to be one million in `<time.h>`.  
6567 If the processor time used is not available or its value cannot be represented, the function shall  
6568 return the value `(clock_t)-1`.

6569 **ERRORS**

6570 No errors are defined.

6571 **EXAMPLES**

6572 None.

6573 **APPLICATION USAGE**

6574 In order to measure the time spent in a program, *clock()* should be called at the start of the  
6575 program and its return value subtracted from the value returned by subsequent calls. The value  
6576 returned by *clock()* is defined for compatibility across systems that have clocks with different  
6577 resolutions. The resolution on any particular system need not be to microsecond accuracy.

6578 The value returned by *clock()* may wrap around on some implementations. For example, on a  
6579 machine with 32-bit values for `clock_t`, it wraps after 2 147 seconds or 36 minutes.

6580 **RATIONALE**

6581 None.

6582 **FUTURE DIRECTIONS**

6583 None.

6584 **SEE ALSO**

6585 *asctime()*, *ctime()*, *difftime()*, *gmtime()*, *localtime()*, *mktime()*, *strftime()*, *strptime()*, *time()*, *utime()*,  
6586 the Base Definitions volume of IEEE Std 1003.1-200x, `<time.h>`

6587 **CHANGE HISTORY**

6588 First released in Issue 1. Derived from Issue 1 of the SVID.

6589 **NAME**6590 clock\_getcpuclockid — access a process CPU-time clock (**ADVANCED REALTIME**)6591 **SYNOPSIS**

6592 CPT #include &lt;time.h&gt;

6593 int clock\_getcpuclockid(pid\_t pid, clockid\_t \*clock\_id);

6594

6595 **DESCRIPTION**6596 The *clock\_getcpuclockid()* function shall return the clock ID of the CPU-time clock of the process  
6597 specified by *pid*. If the process described by *pid* exists and the calling process has permission,  
6598 the clock ID of this clock shall be returned in *clock\_id*.6599 If *pid* is zero, the *clock\_getcpuclockid()* function shall return the clock ID of the CPU-time clock of  
6600 the process making the call, in *clock\_id*.6601 The conditions under which one process has permission to obtain the CPU-time clock ID of  
6602 other processes are implementation-defined.6603 **RETURN VALUE**6604 Upon successful completion, *clock\_getcpuclockid()* shall return zero; otherwise, an error number  
6605 shall be returned to indicate the error.6606 **ERRORS**6607 The *clock\_getcpuclockid()* function shall fail if:6608 [EPERM] The requesting process does not have permission to access the CPU-time  
6609 clock for the process.6610 The *clock\_getcpuclockid()* function may fail if:6611 [ESRCH] No process can be found corresponding to the process specified by *pid*.6612 **EXAMPLES**

6613 None.

6614 **APPLICATION USAGE**6615 The *clock\_getcpuclockid()* function is part of the Process CPU-Time Clocks option and need not  
6616 be provided on all implementations.6617 **RATIONALE**

6618 None.

6619 **FUTURE DIRECTIONS**

6620 None.

6621 **SEE ALSO**6622 *clock\_getres()*, *timer\_create()*, the Base Definitions volume of IEEE Std 1003.1-200x, <time.h>6623 **CHANGE HISTORY**

6624 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

6625 In the SYNOPSIS, the inclusion of &lt;sys/types.h&gt; is no longer required.

## 6626 NAME

6627 clock\_getres, clock\_gettime, clock\_settime — clock and timer functions (REALTIME)

## 6628 SYNOPSIS

6629 TMR #include &lt;time.h&gt;

```
6630 int clock_getres(clockid_t clock_id, struct timespec *res);
6631 int clock_gettime(clockid_t clock_id, struct timespec *tp);
6632 int clock_settime(clockid_t clock_id, const struct timespec *tp);
6633
```

## 6634 DESCRIPTION

6635 The *clock\_getres()* function shall return the resolution of any clock. Clock resolutions are  
 6636 implementation-defined and cannot be set by a process. If the argument *res* is not NULL, the  
 6637 resolution of the specified clock shall be stored in the location pointed to by *res*. If *res* is NULL,  
 6638 the clock resolution is not returned. If the *time* argument of *clock\_settime()* is not a multiple of *res*,  
 6639 then the value is truncated to a multiple of *res*.

6640 The *clock\_gettime()* function shall return the current value *tp* for the specified clock, *clock\_id*.

6641 The *clock\_settime()* function shall set the specified clock, *clock\_id*, to the value specified by *tp*.  
 6642 Time values that are between two consecutive non-negative integer multiples of the resolution  
 6643 of the specified clock shall be truncated down to the smaller multiple of the resolution.

6644 A clock may be system-wide (that is, visible to all processes) or per-process (measuring time that  
 6645 is meaningful only within a process). All implementations shall support a *clock\_id* of  
 6646 CLOCK\_REALTIME as defined in <time.h>. This clock represents the realtime clock for the  
 6647 system. For this clock, the values returned by *clock\_gettime()* and specified by *clock\_settime()*  
 6648 represent the amount of time (in seconds and nanoseconds) since the Epoch. An implementation  
 6649 may also support additional clocks. The interpretation of time values for these clocks is  
 6650 unspecified.

6651 If the value of the CLOCK\_REALTIME clock is set via *clock\_settime()*, the new value of the clock  
 6652 shall be used to determine the time of expiration for absolute time services based upon the  
 6653 CLOCK\_REALTIME clock. This applies to the time at which armed absolute timers expire. If the  
 6654 absolute time requested at the invocation of such a time service is before the new value of the  
 6655 clock, the time service shall expire immediately as if the clock had reached the requested time  
 6656 normally.

6657 Setting the value of the CLOCK\_REALTIME clock via *clock\_settime()* shall have no effect on  
 6658 threads that are blocked waiting for a relative time service based upon this clock, including the  
 6659 *nanosleep()* function; nor on the expiration of relative timers based upon this clock.  
 6660 Consequently, these time services shall expire when the requested relative interval elapses,  
 6661 independently of the new or old value of the clock.

6662 MON If the Monotonic Clock option is supported, all implementations shall support a *clock\_id* of  
 6663 CLOCK\_MONOTONIC defined in <time.h>. This clock represents the monotonic clock for the  
 6664 system. For this clock, the value returned by *clock\_gettime()* represents the amount of time (in  
 6665 seconds and nanoseconds) since an unspecified point in the past (for example, system start-up  
 6666 time, or the Epoch). This point does not change after system start-up time. The value of the  
 6667 CLOCK\_MONOTONIC clock cannot be set via *clock\_settime()*. This function shall fail if it is  
 6668 invoked with a *clock\_id* argument of CLOCK\_MONOTONIC.

6669 The effect of setting a clock via *clock\_settime()* on armed per-process timers associated with a  
 6670 clock other than CLOCK\_REALTIME is implementation-defined.

6671 CS If the value of the CLOCK\_REALTIME clock is set via *clock\_settime()*, the new value of the clock  
 6672 shall be used to determine the time at which the system shall awaken a thread blocked on an

6673 absolute *clock\_nanosleep()* call based upon the CLOCK\_REALTIME clock. If the absolute time  
 6674 requested at the invocation of such a time service is before the new value of the clock, the call  
 6675 shall return immediately as if the clock had reached the requested time normally.

6676 Setting the value of the CLOCK\_REALTIME clock via *clock\_settime()* shall have no effect on any  
 6677 thread that is blocked on a relative *clock\_nanosleep()* call. Consequently, the call shall return  
 6678 when the requested relative interval elapses, independently of the new or old value of the clock.

6679 The appropriate privilege to set a particular clock is implementation-defined.

6680 CPT If `_POSIX_CPUTIME` is defined, implementations shall support clock ID values obtained by  
 6681 invoking *clock\_getcpuclockid()*, which represent the CPU-time clock of a given process.  
 6682 Implementations shall also support the special `clockid_t` value  
 6683 `CLOCK_PROCESS_CPUTIME_ID`, which represents the CPU-time clock of the calling process  
 6684 when invoking one of the *clock\_\**() or *timer\_\**() functions. For these clock IDs, the values  
 6685 returned by *clock\_gettime()* and specified by *clock\_settime()* represent the amount of execution  
 6686 time of the process associated with the clock. Changing the value of a CPU-time clock via  
 6687 *clock\_settime()* shall have no effect on the behavior of the sporadic server scheduling policy (see  
 6688 **Scheduling Policies** (on page 494)).

6689 TCT If `_POSIX_THREAD_CPUTIME` is defined, implementations shall support clock ID values  
 6690 obtained by invoking *pthread\_getcpuclockid()*, which represent the CPU-time clock of a given  
 6691 thread. Implementations shall also support the special `clockid_t` value  
 6692 `CLOCK_THREAD_CPUTIME_ID`, which represents the CPU-time clock of the calling thread  
 6693 when invoking one of the *clock\_\**() or *timer\_\**() functions. For these clock IDs, the values  
 6694 returned by *clock\_gettime()* and specified by *clock\_settime()* shall represent the amount of  
 6695 execution time of the thread associated with the clock. Changing the value of a CPU-time clock  
 6696 via *clock\_settime()* shall have no effect on the behavior of the sporadic server scheduling policy  
 6697 (see **Scheduling Policies** (on page 494)).

#### 6698 RETURN VALUE

6699 A return value of 0 shall indicate that the call succeeded. A return value of -1 shall indicate that  
 6700 an error occurred, and *errno* shall be set to indicate the error.

#### 6701 ERRORS

6702 The *clock\_getres()*, *clock\_gettime()*, and *clock\_settime()* functions shall fail if:

6703 [EINVAL] The *clock\_id* argument does not specify a known clock.

6704 The *clock\_settime()* function shall fail if:

6705 [EINVAL] The *tp* argument to *clock\_settime()* is outside the range for the given clock ID.

6706 [EINVAL] The *tp* argument specified a nanosecond value less than zero or greater than  
 6707 or equal to 1 000 million.

6708 MON [EINVAL] The value of the *clock\_id* argument is `CLOCK_MONOTONIC`.

6709 The *clock\_settime()* function may fail if:

6710 [EPERM] The requesting process does not have the appropriate privilege to set the  
 6711 specified clock.

6712 **EXAMPLES**

6713 None.

6714 **APPLICATION USAGE**

6715 These functions are part of the Timers option and need not be available on all implementations.

6716 Note that the absolute value of the monotonic clock is meaningless (because its origin is  
 6717 arbitrary), and thus there is no need to set it. Furthermore, realtime applications can rely on the  
 6718 fact that the value of this clock is never set and, therefore, that time intervals measured with this  
 6719 clock will not be affected by calls to *clock\_settime()*.

6720 **RATIONALE**

6721 None.

6722 **FUTURE DIRECTIONS**

6723 None.

6724 **SEE ALSO**

6725 *clock\_getcpuclockid()*, *clock\_nanosleep()*, *ctime()*, *mq\_timedreceive()*, *mq\_timedsend()*, *nanosleep()*,  
 6726 *pthread\_mutex\_timedlock()*, *sem\_timedwait()*, *time()*, *timer\_create()*, *timer\_getoverrun()*, the Base  
 6727 Definitions volume of IEEE Std 1003.1-200x, <**time.h**>

6728 **CHANGE HISTORY**

6729 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

6730 **Issue 6**

6731 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
 6732 implementation does not support the Timers option.

6733 The APPLICATION USAGE section is added.

6734 The following changes were made to align with the IEEE P1003.1a draft standard:

6735 • Clarification is added of the effect of resetting the clock resolution.

6736 CPU-time clocks and the *clock\_getcpuclockid()* function are added for alignment with  
 6737 IEEE Std 1003.1d-1999.

6738 The following changes are added for alignment with IEEE Std 1003.1j-2000:

6739 • The DESCRIPTION is updated as follows:

6740 — The value returned by *clock\_gettime()* for CLOCK\_MONOTONIC is specified.6741 — *clock\_settime()* failing for CLOCK\_MONOTONIC is specified.

6742 — The effects of *clock\_settime()* on the *clock\_nanosleep()* function with respect to  
 6743 CLOCK\_REALTIME is specified.

6744 • An [EINVAL] error is added to the ERRORS section, indicating that *clock\_settime()* fails for  
 6745 CLOCK\_MONOTONIC.

6746 • The APPLICATION USAGE section notes that the CLOCK\_MONOTONIC clock need not  
 6747 and shall not be set by *clock\_settime()* since the absolute value of the CLOCK\_MONOTONIC  
 6748 clock is meaningless.

6749 • The *clock\_nanosleep()*, *mq\_timedreceive()*, *mq\_timedsend()*, *pthread\_mutex\_timedlock()*,  
 6750 *sem\_timedwait()*, *timer\_create()*, and *timer\_settime()* functions are added to the SEE ALSO  
 6751 section.

## 6752 NAME

6753 clock\_nanosleep — high resolution sleep with specifiable clock (**ADVANCED REALTIME**)

## 6754 SYNOPSIS

6755 cs #include &lt;time.h&gt;

```
6756 int clock_nanosleep(clockid_t clock_id, int flags,  
6757     const struct timespec *rqtp, struct timespec *rmtp);  
6758
```

## 6759 DESCRIPTION

6760 If the flag `TIMER_ABSTIME` is not set in the *flags* argument, the `clock_nanosleep()` function shall  
6761 cause the current thread to be suspended from execution until either the time interval specified  
6762 by the *rqtp* argument has elapsed, or a signal is delivered to the calling thread and its action is to  
6763 invoke a signal-catching function, or the process is terminated. The clock used to measure the  
6764 time shall be the clock specified by *clock\_id*.

6765 If the flag `TIMER_ABSTIME` is set in the *flags* argument, the `clock_nanosleep()` function shall  
6766 cause the current thread to be suspended from execution until either the time value of the clock  
6767 specified by *clock\_id* reaches the absolute time specified by the *rqtp* argument, or a signal is  
6768 delivered to the calling thread and its action is to invoke a signal-catching function, or the  
6769 process is terminated. If, at the time of the call, the time value specified by *rqtp* is less than or  
6770 equal to the time value of the specified clock, then `clock_nanosleep()` shall return immediately  
6771 and the calling process shall not be suspended.

6772 The suspension time caused by this function may be longer than requested because the  
6773 argument value is rounded up to an integer multiple of the sleep resolution, or because of the  
6774 scheduling of other activity by the system. But, except for the case of being interrupted by a  
6775 signal, the suspension time for the relative `clock_nanosleep()` function (that is, with the  
6776 `TIMER_ABSTIME` flag not set) shall not be less than the time interval specified by *rqtp*, as  
6777 measured by the corresponding clock. The suspension for the absolute `clock_nanosleep()` function  
6778 (that is, with the `TIMER_ABSTIME` flag set) shall be in effect at least until the value of the  
6779 corresponding clock reaches the absolute time specified by *rqtp*, except for the case of being  
6780 interrupted by a signal.

6781 The use of the `clock_nanosleep()` function shall have no effect on the action or blockage of any  
6782 signal.

6783 The `clock_nanosleep()` function shall fail if the *clock\_id* argument refers to the CPU-time clock of  
6784 the calling thread. It is unspecified if *clock\_id* values of other CPU-time clocks are allowed.

## 6785 RETURN VALUE

6786 If the `clock_nanosleep()` function returns because the requested time has elapsed, its return value  
6787 shall be zero.

6788 If the `clock_nanosleep()` function returns because it has been interrupted by a signal, it shall return  
6789 the corresponding error value. For the relative `clock_nanosleep()` function, if the *rmtp* argument is  
6790 non-NULL, the **timespec** structure referenced by it shall be updated to contain the amount of  
6791 time remaining in the interval (the requested time minus the time actually slept). If the *rmtp*  
6792 argument is NULL, the remaining time is not returned. The absolute `clock_nanosleep()` function  
6793 has no effect on the structure referenced by *rmtp*.

6794 If `clock_nanosleep()` fails, it shall return the corresponding error value.

6795 **ERRORS**6796 The *clock\_nanosleep()* function shall fail if:6797 [EINTR] The *clock\_nanosleep()* function was interrupted by a signal.6798 [EINVAL] The *rntp* argument specified a nanosecond value less than zero or greater than  
6799 or equal to 1 000 million; or the `TIMER_ABSTIME` flag was specified in *flags*  
6800 and the *rntp* argument is outside the range for the clock specified by *clock\_id*;  
6801 or the *clock\_id* argument does not specify a known clock, or specifies the  
6802 CPU-time clock of the calling thread.6803 [ENOTSUP] The *clock\_id* argument specifies a clock for which *clock\_nanosleep()* is not  
6804 supported, such as a CPU-time clock.6805 **EXAMPLES**

6806 None.

6807 **APPLICATION USAGE**6808 Calling *clock\_nanosleep()* with the value `TIMER_ABSTIME` not set in the *flags* argument and with  
6809 a *clock\_id* of `CLOCK_REALTIME` is equivalent to calling *nanosleep()* with the same *rntp* and *rntp*  
6810 arguments.6811 **RATIONALE**6812 The *nanosleep()* function specifies that the system-wide clock `CLOCK_REALTIME` is used to  
6813 measure the elapsed time for this time service. However, with the introduction of the monotonic  
6814 clock `CLOCK_MONOTONIC` a new relative sleep function is needed to allow an application to  
6815 take advantage of the special characteristics of this clock.6816 There are many applications in which a process needs to be suspended and then activated  
6817 multiple times in a periodic way; for example, to poll the status of a non-interrupting device or  
6818 to refresh a display device. For these cases, it is known that precise periodic activation cannot be  
6819 achieved with a relative *sleep()* or *nanosleep()* function call. Suppose, for example, a periodic  
6820 process that is activated at time  $T_0$ , executes for a while, and then wants to suspend itself until  
6821 time  $T_0+T$ , the period being  $T$ . If this process wants to use the *nanosleep()* function, it must first  
6822 call *clock\_gettime()* to get the current time, then calculate the difference between the current time  
6823 and  $T_0+T$  and, finally, call *nanosleep()* using the computed interval. However, the process could  
6824 be preempted by a different process between the two function calls, and in this case the interval  
6825 computed would be wrong; the process would wake up later than desired. This problem would  
6826 not occur with the absolute *clock\_nanosleep()* function, since only one function call would be  
6827 necessary to suspend the process until the desired time. In other cases, however, a relative sleep  
6828 is needed, and that is why both functionalities are required.6829 Although it is possible to implement periodic processes using the timers interface, this  
6830 implementation would require the use of signals, and the reservation of some signal numbers. In  
6831 this regard, the reasons for including an absolute version of the *clock\_nanosleep()* function in  
6832 IEEE Std 1003.1-200x are the same as for the inclusion of the relative *nanosleep()*.6833 It is also possible to implement precise periodic processes using *pthread\_cond\_timedwait()*, in  
6834 which an absolute timeout is specified that takes effect if the condition variable involved is  
6835 never signaled. However, the use of this interface is unnatural, and involves performing other  
6836 operations on mutexes and condition variables that imply an unnecessary overhead.  
6837 Furthermore, *pthread\_cond\_timedwait()* is not available in implementations that do not support  
6838 threads.6839 Although the interface of the relative and absolute versions of the new high resolution sleep  
6840 service is the same *clock\_nanosleep()* function, the *rntp* argument is only used in the relative  
6841 sleep. This argument is needed in the relative *clock\_nanosleep()* function to reissue the function

6842 call if it is interrupted by a signal, but it is not needed in the absolute *clock\_nanosleep()* function  
6843 call; if the call is interrupted by a signal, the absolute *clock\_nanosleep()* function can be invoked  
6844 again with the same *rqt* argument used in the interrupted call.

6845 **FUTURE DIRECTIONS**

6846 None.

6847 **SEE ALSO**

6848 *clock\_getres()*, *nanosleep()*, *pthread\_cond\_timedwait()*, *sleep()*, the Base Definitions volume of  
6849 IEEE Std 1003.1-200x, <**time.h**>

6850 **CHANGE HISTORY**

6851 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

6852 **NAME**6853 clock\_settime — clock and timer functions (**REALTIME**)6854 **SYNOPSIS**6855 TMR `#include <time.h>`6856 `int clock_settime(clockid_t clock_id, const struct timespec *tp);`

6857

6858 **DESCRIPTION**6859 Refer to *clock\_getres()*.

6860 **NAME**6861 `clog, clogf, clogl` — complex natural logarithm functions6862 **SYNOPSIS**6863 `#include <complex.h>`6864 `double complex clog(double complex z);`6865 `float complex clogf(float complex z);`6866 `long double complex clogl(long double complex z);`6867 **DESCRIPTION**

6868 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
6869 conflict between the requirements described here and the ISO C standard is unintentional. This  
6870 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

6871 These functions shall compute the complex natural (base  $e$ ) logarithm of  $z$ , with a branch cut  
6872 along the negative real axis.

6873 **RETURN VALUE**

6874 These functions shall return the complex natural logarithm value, in the range of a strip  
6875 mathematically unbounded along the real axis and in the interval  $[-i\pi, +i\pi]$  along the imaginary  
6876 axis.

6877 **ERRORS**

6878 No errors are defined.

6879 **EXAMPLES**

6880 None.

6881 **APPLICATION USAGE**

6882 None.

6883 **RATIONALE**

6884 None.

6885 **FUTURE DIRECTIONS**

6886 None.

6887 **SEE ALSO**6888 `cexp()`, the Base Definitions volume of IEEE Std 1003.1-200x, `<complex.h>`6889 **CHANGE HISTORY**

6890 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

6891 **NAME**

6892           close — close a file descriptor

6893 **SYNOPSIS**

6894           #include &lt;unistd.h&gt;

6895           int close(int *fildes*);6896 **DESCRIPTION**

6897       The *close()* function shall deallocate the file descriptor indicated by *fildes*. To deallocate means  
 6898       to make the file descriptor available for return by subsequent calls to *open()* or other functions  
 6899       that allocate file descriptors. All outstanding record locks owned by the process on the file  
 6900       associated with the file descriptor shall be removed (that is, unlocked).

6901       If *close()* is interrupted by a signal that is to be caught, it shall return  $-1$  with *errno* set to [EINTR]  
 6902       and the state of *fildes* is unspecified. If an I/O error occurred while reading from or writing to the  
 6903       file system during *close()*, it may return  $-1$  with *errno* set to [EIO]; if this error is returned, the  
 6904       state of *fildes* is unspecified.

6905       When all file descriptors associated with a pipe or FIFO special file are closed, any data  
 6906       remaining in the pipe or FIFO shall be discarded.

6907       When all file descriptors associated with an open file description have been closed the open file  
 6908       description shall be freed.

6909       If the link count of the file is 0, when all file descriptors associated with the file are closed, the  
 6910       space occupied by the file shall be freed and the file shall no longer be accessible.

6911 **XSR**       If a STREAMS-based *fildes* is closed and the calling process was previously registered to receive  
 6912       a SIGPOLL signal for events associated with that STREAM, the calling process shall be  
 6913       unregistered for events associated with the STREAM. The last *close()* for a STREAM shall cause  
 6914       the STREAM associated with *fildes* to be dismantled. If O\_NONBLOCK is not set and there have  
 6915       been no signals posted for the STREAM, and if there is data on the module's write queue, *close()*  
 6916       shall wait for an unspecified time (for each module and driver) for any output to drain before  
 6917       dismantling the STREAM. The time delay can be changed via an I\_SETCLTIME *ioctl()* request. If  
 6918       the O\_NONBLOCK flag is set, or if there are any pending signals, *close()* shall not wait for  
 6919       output to drain, and shall dismantle the STREAM immediately.

6920       If the implementation supports STREAMS-based pipes, and *fildes* is associated with one end of a  
 6921       pipe, the last *close()* shall cause a hangup to occur on the other end of the pipe. In addition, if the  
 6922       other end of the pipe has been named by *fattach()*, then the last *close()* shall force the named end  
 6923       to be detached by *fdetach()*. If the named end has no open file descriptors associated with it and  
 6924       gets detached, the STREAM associated with that end shall also be dismantled.

6925 **XSI**       If *fildes* refers to the master side of a pseudo-terminal, and this is the last close, a SIGHUP signal  
 6926       shall be sent to the process group, if any, for which the slave side of the pseudo-terminal is the  
 6927       controlling terminal. It is unspecified whether closing the master side of the pseudo-terminal  
 6928       flushes all queued input and output.

6929 **XSR**       If *fildes* refers to the slave side of a STREAMS-based pseudo-terminal, a zero-length message  
 6930       may be sent to the master.

6931 **AIO**       When there is an outstanding cancelable asynchronous I/O operation against *fildes* when *close()*  
 6932       is called, that I/O operation may be canceled. An I/O operation that is not canceled completes  
 6933       as if the *close()* operation had not yet occurred. All operations that are not canceled shall  
 6934       complete as if the *close()* blocked until the operations completed. The *close()* operation itself  
 6935       need not block awaiting such I/O completion. Whether any I/O operation is canceled, and  
 6936       which I/O operation may be canceled upon *close()*, is implementation-defined.

6937 MF|SHM If a shared memory object or a memory mapped file remains referenced at the last close (that is,  
6938 a process has it mapped), then the entire contents of the memory object shall persist until the  
6939 memory object becomes unreferenced. If this is the last close of a shared memory object or a  
6940 memory mapped file and the close results in the memory object becoming unreferenced, and the  
6941 memory object has been unlinked, then the memory object shall be removed.

6942 If *fdes* refers to a socket, *close()* shall cause the socket to be destroyed. If the socket is in  
6943 connection-mode, and the `SO_LINGER` option is set for the socket with non-zero linger time,  
6944 and the socket has untransmitted data, then *close()* shall block for up to the current linger  
6945 interval until all data is transmitted.

#### 6946 RETURN VALUE

6947 Upon successful completion, 0 shall be returned; otherwise, -1 shall be returned and *errno* set to  
6948 indicate the error.

#### 6949 ERRORS

6950 The *close()* function shall fail if:

6951 [EBADF] The *fdes* argument is not a valid file descriptor.

6952 [EINTR] The *close()* function was interrupted by a signal.

6953 The *close()* function may fail if:

6954 [EIO] An I/O error occurred while reading from or writing to the file system.

#### 6955 EXAMPLES

##### 6956 Reassigning a File Descriptor

6957 The following example closes the file descriptor associated with standard output for the current  
6958 process, re-assigns standard output to a new file descriptor, and closes the original file  
6959 descriptor to clean up. This example assumes that the file descriptor 0 (which is the descriptor  
6960 for standard input) is not closed.

```
6961 #include <unistd.h>  
6962 ...  
6963 int pfd;  
6964 ...  
6965 close(1);  
6966 dup(pfd);  
6967 close(pfd);  
6968 ...
```

6969 Incidentally, this is exactly what could be achieved using:

```
6970 dup2(pfd, 1);  
6971 close(pfd);
```

##### 6972 Closing a File Descriptor

6973 In the following example, *close()* is used to close a file descriptor after an unsuccessful attempt is  
6974 made to associate that file descriptor with a stream.

```
6975 #include <stdio.h>  
6976 #include <unistd.h>  
6977 #include <stdlib.h>
```

```

6978     #define LOCKFILE "/etc/ptmp"
6979     ...
6980     int pfd;
6981     FILE *fpfd;
6982     ...
6983     if ((fpfd = fdopen (pfd, "w")) == NULL) {
6984         close(pfd);
6985         unlink(LOCKFILE);
6986         exit(1);
6987     }
6988     ...

```

#### 6989 APPLICATION USAGE

6990 An application that had used the *stdio* routine *fopen()* to open a file should use the  
 6991 corresponding *fclose()* routine rather than *close()*. Once a file is closed, the file descriptor no  
 6992 longer exists, since the integer corresponding to it no longer refers to a file.

#### 6993 RATIONALE

6994 The use of interruptible device close routines should be discouraged to avoid problems with the  
 6995 implicit closes of file descriptors by *exec* and *exit()*. This volume of IEEE Std 1003.1-200x only  
 6996 intends to permit such behavior by specifying the [EINTR] error condition.

#### 6997 FUTURE DIRECTIONS

6998 None.

#### 6999 SEE ALSO

7000 *fattach()*, *fclose()*, *fdetach()*, *fopen()*, *ioctl()*, *open()*, the Base Definitions volume of  
 7001 IEEE Std 1003.1-200x, <*unistd.h*>, Section 2.6 (on page 488)

#### 7002 CHANGE HISTORY

7003 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 7004 Issue 5

7005 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension.

#### 7006 Issue 6

7007 The DESCRIPTION related to a STREAMS-based file or pseudo-terminal is marked as part of the  
 7008 XSI STREAMS Option Group.

7009 The following new requirements on POSIX implementations derive from alignment with the  
 7010 Single UNIX Specification:

- 7011 • The [EIO] error condition is added as an optional error.
- 7012 • The DESCRIPTION is updated to describe the state of the *fildev* file descriptor as unspecified  
 7013 if an I/O error occurs and an [EIO] error condition is returned.

7014 Text referring to sockets is added to the DESCRIPTION.

7015 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that  
 7016 shared memory objects and memory mapped files (and not typed memory objects) are the types  
 7017 of memory objects to which the paragraph on last closes applies.

7018 **NAME**

7019       closedir — close a directory stream

7020 **SYNOPSIS**

7021       #include &lt;dirent.h&gt;

7022       int closedir(DIR \*dirp);

7023 **DESCRIPTION**

7024       The *closedir()* function shall close the directory stream referred to by the argument *dirp*. Upon  
7025       return, the value of *dirp* may no longer point to an accessible object of the type **DIR**. If a file  
7026       descriptor is used to implement type **DIR**, that file descriptor shall be closed.

7027 **RETURN VALUE**

7028       Upon successful completion, *closedir()* shall return 0; otherwise, -1 shall be returned and *errno*  
7029       set to indicate the error.

7030 **ERRORS**7031       The *closedir()* function may fail if:7032       [EBADF]       The *dirp* argument does not refer to an open directory stream.7033       [EINTR]       The *closedir()* function was interrupted by a signal.7034 **EXAMPLES**7035       **Closing a Directory Stream**7036       The following program fragment demonstrates how the *closedir()* function is used.

```
7037       ...  
7038       DIR *dir;  
7039       struct dirent *dp;  
7040       ...  
7041       if ((dir = opendir(".")) == NULL) {  
7042       ...  
7043       }  
7044       while ((dp = readdir(dir)) != NULL) {  
7045       ...  
7046       }  
7047       closedir(dir);  
7048       ...
```

7049 **APPLICATION USAGE**

7050       None.

7051 **RATIONALE**

7052       None.

7053 **FUTURE DIRECTIONS**

7054       None.

7055 **SEE ALSO**7056       *opendir()*, the Base Definitions volume of IEEE Std 1003.1-200x, <dirent.h>

7057 **CHANGE HISTORY**

7058 First released in Issue 2.

7059 **Issue 6**7060 In the SYNOPSIS, the optional include of the `<sys/types.h>` header is removed.7061 The following new requirements on POSIX implementations derive from alignment with the  
7062 Single UNIX Specification:

- 7063 • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was  
7064 required for conforming implementations of previous POSIX specifications, it was not  
7065 required for UNIX applications.
- 7066 • The [EINTR] error condition is added as an optional error condition.

7067 **NAME**

7068 closelog, openlog, setlogmask, syslog — control system log

7069 **SYNOPSIS**

```
7070 xSI #include <syslog.h>
7071
7072 void closelog(void);
7073 void openlog(const char *ident, int logopt, int facility);
7074 int setlogmask(int maskpri);
7075 void syslog(int priority, const char *message, ... /* arguments */);
```

7076 **DESCRIPTION**

7077 The *syslog()* function shall send a message to an implementation-defined logging facility, which  
 7078 may log it in an implementation-defined system log, write it to the system console, forward it to  
 7079 a list of users, or forward it to the logging facility on another host over the network. The logged  
 7080 message shall include a message header and a message body. The message header contains at  
 7081 least a timestamp and a tag string.

7082 The message body is generated from the *message* and following arguments in the same manner  
 7083 as if these were arguments to *printf()*, except that the additional conversion specification *%m*  
 7084 shall be recognized; it shall convert no arguments, shall cause the output of the error message  
 7085 string associated with the value of *errno* on entry to *syslog()*, and may be mixed with argument  
 7086 specifications of the "*%n\$*" form. If a complete conversion specification with the *m* conversion  
 7087 specifier character is not just *%m*, the behavior is undefined. A trailing *<newline>* may be added  
 7088 if needed.

7089 Values of the *priority* argument are formed by OR'ing together a severity level value and an  
 7090 optional facility value. If no facility value is specified, the current default facility value is used.

7091 Possible values of severity level include:

7092	LOG_EMERG	A panic condition.
7093	LOG_ALERT	A condition that should be corrected immediately, such as a corrupted system database.
7094		
7095	LOG_CRIT	Critical conditions, such as hard device errors.
7096	LOG_ERR	Errors.
7097	LOG_WARNING	
7098		Warning messages.
7099	LOG_NOTICE	Conditions that are not error conditions, but that may require special handling.
7100		
7101	LOG_INFO	Informational messages.
7102	LOG_DEBUG	Messages that contain information normally of use only when debugging a program.
7103		

7104 The facility indicates the application or system component generating the message. Possible  
 7105 facility values include:

7106	LOG_USER	Messages generated by arbitrary processes. This is the default facility identifier if none is specified.
7107		
7108	LOG_LOCAL0	Reserved for local use.

- 7109 LOG\_LOCAL1 Reserved for local use.
- 7110 LOG\_LOCAL2 Reserved for local use.
- 7111 LOG\_LOCAL3 Reserved for local use.
- 7112 LOG\_LOCAL4 Reserved for local use.
- 7113 LOG\_LOCAL5 Reserved for local use.
- 7114 LOG\_LOCAL6 Reserved for local use.
- 7115 LOG\_LOCAL7 Reserved for local use.
- 7116 The *openlog()* function shall set process attributes that affect subsequent calls to *syslog()*. The
- 7117 *ident* argument is a string that is prepended to every message. The *logopt* argument indicates
- 7118 logging options. Values for *logopt* are constructed by a bitwise-inclusive OR of zero or more of
- 7119 the following:
- 7120 LOG\_PID Log the process ID with each message. This is useful for identifying specific
- 7121 processes.
- 7122 LOG\_CONS Write messages to the system console if they cannot be sent to the logging
- 7123 facility. The *syslog()* function ensures that the process does not acquire the
- 7124 console as a controlling terminal in the process of writing the message.
- 7125 LOG\_NDELAY Open the connection to the logging facility immediately. Normally the open is
- 7126 delayed until the first message is logged. This is useful for programs that need
- 7127 to manage the order in which file descriptors are allocated.
- 7128 LOG\_ODELAY Delay open until *syslog()* is called.
- 7129 LOG\_NOWAIT Do not wait for child processes that may have been created during the course
- 7130 of logging the message. This option should be used by processes that enable
- 7131 notification of child termination using SIGCHLD, since *syslog()* may
- 7132 otherwise block waiting for a child whose exit status has already been
- 7133 collected.
- 7134 The *facility* argument encodes a default facility to be assigned to all messages that do not have
- 7135 an explicit facility already encoded. The initial default facility is LOG\_USER.
- 7136 The *openlog()* and *syslog()* functions may allocate a file descriptor. It is not necessary to call
- 7137 *openlog()* prior to calling *syslog()*.
- 7138 The *closelog()* function shall close any open file descriptors allocated by previous calls to
- 7139 *openlog()* or *syslog()*.
- 7140 The *setlogmask()* function shall set the log priority mask for the current process to *maskpri* and
- 7141 return the previous mask. If the *maskpri* argument is 0, the current log mask is not modified.
- 7142 Calls by the current process to *syslog()* with a priority not set in *maskpri* shall be rejected. The
- 7143 default log mask allows all priorities to be logged. A call to *openlog()* is not required prior to
- 7144 calling *setlogmask()*.
- 7145 Symbolic constants for use as values of the *logopt*, *facility*, *priority*, and *maskpri* arguments are
- 7146 defined in the `<syslog.h>` header.
- 7147 **RETURN VALUE**
- 7148 The *setlogmask()* function shall return the previous log priority mask. The *closelog()*, *openlog()*,
- 7149 and *syslog()* functions shall not return a value.

7150 **ERRORS**

7151       None.

7152 **EXAMPLES**7153       **Using openlog()**

7154       The following example causes subsequent calls to *syslog()* to log the process ID with each message, and to write messages to the system console if they cannot be sent to the logging facility.

```
7157       #include <syslog.h>
7158       char *ident = "Process demo";
7159       int logopt = LOG_PID | LOG_CONS;
7160       int facility = LOG_USER;
7161       ...
7162       openlog(ident, logopt, facility);
```

7163       **Using setlogmask()**

7164       The following example causes subsequent calls to *syslog()* to accept error messages or messages generated by arbitrary processes, and to reject all other messages.

```
7166       #include <syslog.h>
7167       int result;
7168       int mask = LOG_MASK (LOG_ERR | LOG_USER);
7169       ...
7170       result = setlogmask(mask);
```

7171       **Using syslog**

7172       The following example sends the message "This is a message" to the default logging facility, marking the message as an error message generated by random processes.

```
7174       #include <syslog.h>
7175       char *message = "This is a message";
7176       int priority = LOG_ERR | LOG_USER;
7177       ...
7178       syslog(priority, message);
```

7179 **APPLICATION USAGE**

7180       None.

7181 **RATIONALE**

7182       None.

7183 **FUTURE DIRECTIONS**

7184       None.

7185 **SEE ALSO**7186       *printf()*, the Base Definitions volume of IEEE Std 1003.1-200x, <syslog.h>

7187 **CHANGE HISTORY**

7188       First released in Issue 4, Version 2.

7189 **Issue 5**

7190       Moved from X/OPEN UNIX extension to BASE.

7191 **NAME**

7192 confstr — get configurable variables

7193 **SYNOPSIS**

7194 #include &lt;unistd.h&gt;

7195 size\_t confstr(int name, char \*buf, size\_t len);

7196 **DESCRIPTION**7197 The *confstr()* function shall return configuration-defined string values. Its use and purpose are |  
7198 similar to *sysconf()*, but it is used where string values rather than numeric values are returned. |7199 The *name* argument represents the system variable to be queried. The implementation shall  
7200 support the following name values, defined in <unistd.h>. It may support others:

7201 \_CS\_PATH  
 7202 \_CS\_POSIX\_V6\_ILP32\_OFF32\_CFLAGS  
 7203 \_CS\_POSIX\_V6\_ILP32\_OFF32\_LDFLAGS  
 7204 \_CS\_POSIX\_V6\_ILP32\_OFF32\_LIBS  
 7205 \_CS\_POSIX\_V6\_ILP32\_OFF32\_LINTFLAGS  
 7206 \_CS\_POSIX\_V6\_ILP32\_OFFBIG\_CFLAGS  
 7207 \_CS\_POSIX\_V6\_ILP32\_OFFBIG\_LDFLAGS  
 7208 \_CS\_POSIX\_V6\_ILP32\_OFFBIG\_LIBS  
 7209 \_CS\_POSIX\_V6\_ILP32\_OFFBIG\_LINTFLAGS  
 7210 \_CS\_POSIX\_V6\_LP64\_OFF64\_CFLAGS  
 7211 \_CS\_POSIX\_V6\_LP64\_OFF64\_LDFLAGS  
 7212 \_CS\_POSIX\_V6\_LP64\_OFF64\_LIBS  
 7213 \_CS\_POSIX\_V6\_LP64\_OFF64\_LINTFLAGS  
 7214 \_CS\_POSIX\_V6\_LPBIG\_OFFBIG\_CFLAGS  
 7215 \_CS\_POSIX\_V6\_LPBIG\_OFFBIG\_LDFLAGS  
 7216 \_CS\_POSIX\_V6\_LPBIG\_OFFBIG\_LIBS  
 7217 \_CS\_POSIX\_V6\_LPBIG\_OFFBIG\_LINTFLAGS  
 7218 XSI CS\_XBS5\_ILP32\_OFF32\_CFLAGS (LEGACY)  
 7219 CS\_XBS5\_ILP32\_OFF32\_LDFLAGS (LEGACY)  
 7220 CS\_XBS5\_ILP32\_OFF32\_LIBS (LEGACY)  
 7221 CS\_XBS5\_ILP32\_OFF32\_LINTFLAGS (LEGACY)  
 7222 CS\_XBS5\_ILP32\_OFFBIG\_CFLAGS (LEGACY)  
 7223 CS\_XBS5\_ILP32\_OFFBIG\_LDFLAGS (LEGACY)  
 7224 CS\_XBS5\_ILP32\_OFFBIG\_LIBS (LEGACY)  
 7225 CS\_XBS5\_ILP32\_OFFBIG\_LINTFLAGS (LEGACY)  
 7226 CS\_XBS5\_LP64\_OFF64\_CFLAGS (LEGACY)  
 7227 CS\_XBS5\_LP64\_OFF64\_LDFLAGS (LEGACY)  
 7228 CS\_XBS5\_LP64\_OFF64\_LIBS (LEGACY)  
 7229 CS\_XBS5\_LP64\_OFF64\_LINTFLAGS (LEGACY)  
 7230 CS\_XBS5\_LPBIG\_OFFBIG\_CFLAGS (LEGACY)  
 7231 CS\_XBS5\_LPBIG\_OFFBIG\_LDFLAGS (LEGACY)  
 7232 CS\_XBS5\_LPBIG\_OFFBIG\_LIBS (LEGACY)  
 7233 CS\_XBS5\_LPBIG\_OFFBIG\_LINTFLAGS (LEGACY)

7234

7235 If *len* is not 0, and if *name* has a configuration-defined value, *confstr()* shall copy that value into  
7236 the *len*-byte buffer pointed to by *buf*. If the string to be returned is longer than *len* bytes,  
7237 including the terminating null, then *confstr()* shall truncate the string to *len*−1 bytes and null-  
7238 terminate the result. The application can detect that the string was truncated by comparing the  
7239 value returned by *confstr()* with *len*.

7240 If *len* is 0 and *buf* is a null pointer, then *confstr()* shall still return the integer value as defined  
 7241 below, but shall not return a string. If *len* is 0 but *buf* is not a null pointer, the result is  
 7242 unspecified.

7243 If the implementation supports the Shell option, the string stored in *buf* after a call to:

```
7244 confstr(_CS_PATH, buf, sizeof(buf))
```

7245 can be used as a value of the *PATH* environment variable that accesses all of the standard  
 7246 utilities of IEEE Std 1003.1-200x, if the return value is less than or equal to *sizeof(buf)*.

#### 7247 RETURN VALUE

7248 If *name* has a configuration-defined value, *confstr()* shall return the size of buffer that would be  
 7249 needed to hold the entire configuration-defined value including the terminating null. If this  
 7250 return value is greater than *len*, the string returned in *buf* is truncated.

7251 If *name* is invalid, *confstr()* shall return 0 and set *errno* to indicate the error.

7252 If *name* does not have a configuration-defined value, *confstr()* shall return 0 and leave *errno*  
 7253 unchanged.

#### 7254 ERRORS

7255 The *confstr()* function shall fail if:

7256 [EINVAL] The value of the *name* argument is invalid.

#### 7257 EXAMPLES

7258 None.

#### 7259 APPLICATION USAGE

7260 An application can distinguish between an invalid *name* parameter value and one that  
 7261 corresponds to a configurable variable that has no configuration-defined value by checking if  
 7262 *errno* is modified. This mirrors the behavior of *sysconf()*.

7263 The original need for this function was to provide a way of finding the configuration-defined  
 7264 default value for the environment variable *PATH*. Since *PATH* can be modified by the user to  
 7265 include directories that could contain utilities replacing the standard utilities in the Shell and  
 7266 Utilities volume of IEEE Std 1003.1-200x, applications need a way to determine the system-  
 7267 supplied *PATH* environment variable value that contains the correct search path for the standard  
 7268 utilities.

7269 An application could use:

```
7270 confstr(name, (char *)NULL, (size_t)0)
```

7271 to find out how big a buffer is needed for the string value; use *malloc()* to allocate a buffer to  
 7272 hold the string; and call *confstr()* again to get the string. Alternately, it could allocate a fixed,  
 7273 static buffer that is big enough to hold most answers (perhaps 512 or 1 024 bytes), but then use  
 7274 *malloc()* to allocate a larger buffer if it finds that this is too small.

#### 7275 RATIONALE

7276 Application developers can normally determine any configuration variable by means of reading  
 7277 from the stream opened by a call to:

```
7278 popen("command -p getconf variable", "r");
```

7279 The *confstr()* function with a *name* argument of *\_CS\_PATH* returns a string that can be used as a  
 7280 *PATH* environment variable setting that will reference the standard shell and utilities as  
 7281 described in the Shell and Utilities volume of IEEE Std 1003.1-200x.

7282 The *confstr()* function copies the returned string into a buffer supplied by the application instead  
7283 of returning a pointer to a string. This allows a cleaner function in some implementations (such  
7284 as those with lightweight threads) and resolves questions about when the application must copy  
7285 the string returned.

7286 **FUTURE DIRECTIONS**

7287 None.

7288 **SEE ALSO**

7289 *pathconf()*, *sysconf()*, the Base Definitions volume of IEEE Std 1003.1-200x, <**unistd.h**>, the Shell  
7290 and Utilities volume of IEEE Std 1003.1-200x, *c99*

7291 **CHANGE HISTORY**

7292 First released in Issue 4. Derived from the ISO POSIX-2 standard.

7293 **Issue 5**

7294 A table indicating the permissible values of *name* are added to the DESCRIPTION. All those  
7295 marked EX are new in this issue.

7296 **Issue 6**

7297 The Open Group Corrigendum U033/7 is applied. The return value for the case returning the  
7298 size of the buffer now explicitly states that this includes the terminating null.

7299 The following new requirements on POSIX implementations derive from alignment with the  
7300 Single UNIX Specification:

- 7301 • The DESCRIPTION is updated with new arguments which can be used to determine  
7302 configuration strings for C compiler flags, linker/loader flags, and libraries for each different  
7303 supported programming environment. This is a change to support data size neutrality.

7304 The following changes were made to align with the IEEE P1003.1a draft standard:

- 7305 • The DESCRIPTION is updated to include text describing how `_CS_PATH` can be used to  
7306 obtain a *PATH* to access the standard utilities.

7307 The macros associated with the *c89* programming models are marked LEGACY and new  
7308 equivalent macros associated with *c99* are introduced.

7309 **NAME**

7310 conj, conjf, conjl — complex conjugate functions

7311 **SYNOPSIS**

7312 #include &lt;complex.h&gt;

7313 double complex conj(double complex z);

7314 float complex conjf(float complex z);

7315 long double complex conjl(long double complex z);

7316 **DESCRIPTION**7317 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
7318 conflict between the requirements described here and the ISO C standard is unintentional. This  
7319 volume of IEEE Std 1003.1-200x defers to the ISO C standard.7320 These functions shall compute the complex conjugate of *z*, by reversing the sign of its imaginary  
7321 part.7322 **RETURN VALUE**

7323 These functions return the complex conjugate value.

7324 **ERRORS**

7325 No errors are defined.

7326 **EXAMPLES**

7327 None.

7328 **APPLICATION USAGE**

7329 None.

7330 **RATIONALE**

7331 None.

7332 **FUTURE DIRECTIONS**

7333 None.

7334 **SEE ALSO**7335 *carg()*, *cimag()*, *cproj()*, *creal()*, the Base Definitions volume of IEEE Std 1003.1-200x,  
7336 <complex.h>7337 **CHANGE HISTORY**

7338 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

7339 **NAME**

7340 connect — connect a socket

7341 **SYNOPSIS**

7342 #include &lt;sys/socket.h&gt;

7343 int connect(int *socket*, const struct sockaddr \**address*,  
7344 socklen\_t *address\_len*);7345 **DESCRIPTION**7346 The *connect()* function shall attempt to make a connection on a socket. The function takes the  
7347 following arguments:

- 7348 *socket* Specifies the file descriptor associated with the socket.
- 7349 *address* Points to a **sockaddr** structure containing the peer address. The length and  
7350 format of the address depend on the address family of the socket.
- 7351 *address\_len* Specifies the length of the **sockaddr** structure pointed to by the *address*  
7352 argument.

7353 If the socket has not already been bound to a local address, *connect()* shall bind it to an address  
7354 which, unless the socket's address family is AF\_UNIX, is an unused local address.

7355 If the initiating socket is not connection-mode, then *connect()* shall set the socket's peer address,  
7356 and no connection is made. For SOCK\_DGRAM sockets, the peer address identifies where all  
7357 datagrams are sent on subsequent *send()* functions, and limits the remote sender for subsequent  
7358 *recv()* functions. If *address* is a null address for the protocol, the socket's peer address shall be  
7359 reset.

7360 If the initiating socket is connection-mode, then *connect()* shall attempt to establish a connection  
7361 to the address specified by the *address* argument. If the connection cannot be established  
7362 immediately and O\_NONBLOCK is not set for the file descriptor for the socket, *connect()* shall  
7363 block for up to an unspecified timeout interval until the connection is established. If the timeout  
7364 interval expires before the connection is established, *connect()* shall fail and the connection  
7365 attempt shall be aborted. If *connect()* is interrupted by a signal that is caught while blocked  
7366 waiting to establish a connection, *connect()* shall fail and set *errno* to [EINTR], but the connection  
7367 request shall not be aborted, and the connection shall be established asynchronously.

7368 If the connection cannot be established immediately and O\_NONBLOCK is set for the file  
7369 descriptor for the socket, *connect()* shall fail and set *errno* to [EINPROGRESS], but the connection  
7370 request shall not be aborted, and the connection shall be established asynchronously.  
7371 Subsequent calls to *connect()* for the same socket, before the connection is established, shall fail  
7372 and set *errno* to [EALREADY].

7373 When the connection has been established asynchronously, *select()* and *poll()* shall indicate that  
7374 the file descriptor for the socket is ready for writing.

7375 The socket in use may require the process to have appropriate privileges to use the *connect()*  
7376 function.

7377 **RETURN VALUE**

7378 Upon successful completion, *connect()* shall return 0; otherwise, -1 shall be returned and *errno*  
7379 set to indicate the error.

7380 **ERRORS**7381 The *connect()* function shall fail if:

7382 [EADDRNOTAVAIL]

7383 The specified address is not available from the local machine.

7384	[EAFNOSUPPORT]	
7385		The specified address is not a valid address for the address family of the
7386		specified socket.
7387	[EALREADY]	A connection request is already in progress for the specified socket.
7388	[EBADF]	The <i>socket</i> argument is not a valid file descriptor.
7389	[ECONNREFUSED]	
7390		The target address was not listening for connections or refused the connection
7391		request.
7392	[EINPROGRESS]	O_NONBLOCK is set for the file descriptor for the socket and the connection
7393		cannot be immediately established; the connection shall be established
7394		asynchronously.
7395	[EINTR]	The attempt to establish a connection was interrupted by delivery of a signal
7396		that was caught; the connection shall be established asynchronously.
7397	[EISCONN]	The specified socket is connection-mode and is already connected.
7398	[ENETUNREACH]	
7399		No route to the network is present.
7400	[ENOTSOCK]	The <i>socket</i> argument does not refer to a socket.
7401	[EPROTOTYPE]	The specified address has a different type than the socket bound to the
7402		specified peer address.
7403	[ETIMEDOUT]	The attempt to connect timed out before a connection was made.
7404		If the address family of the socket is AF_UNIX, then <i>connect()</i> shall fail if:
7405	[EIO]	An I/O error occurred while reading from or writing to the file system.
7406	[ELOOP]	A loop exists in symbolic links encountered during resolution of the pathname
7407		in <i>address</i> .
7408	[ENAMETOOLONG]	
7409		A component of a pathname exceeded {NAME_MAX} characters, or an entire
7410		pathname exceeded {PATH_MAX} characters.
7411	[ENOENT]	A component of the pathname does not name an existing file or the pathname
7412		is an empty string.
7413	[ENOTDIR]	A component of the path prefix of the pathname in <i>address</i> is not a directory.
7414		The <i>connect()</i> function may fail if:
7415	[EACCES]	Search permission is denied for a component of the path prefix; or write
7416		access to the named socket is denied.
7417	[EADDRINUSE]	Attempt to establish a connection that uses addresses that are already in use.
7418	[ECONNRESET]	Remote host reset the connection request.
7419	[EHOSTUNREACH]	
7420		The destination host cannot be reached (probably because the host is down or
7421		a remote router cannot reach it).
7422	[EINVAL]	The <i>address_len</i> argument is not a valid length for the address family; or
7423		invalid address family in the <b>sockaddr</b> structure.

7424 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during |  
7425 resolution of the pathname in *address*. |

7426 [ENAMETOOLONG]  
7427 Pathname resolution of a symbolic link produced an intermediate result |  
7428 whose length exceeds {PATH\_MAX}. |

7429 [ENETDOWN] The local network interface used to reach the destination is down.

7430 [ENOBUFS] No buffer space is available.

7431 [EOPNOTSUPP] The socket is listening and cannot be connected.

7432 **EXAMPLES**  
7433 None.

7434 **APPLICATION USAGE**  
7435 If *connect()* fails, the state of the socket is unspecified. Conforming applications should close the |  
7436 file descriptor and create a new socket before attempting to reconnect. |

7437 **RATIONALE**  
7438 None.

7439 **FUTURE DIRECTIONS**  
7440 None.

7441 **SEE ALSO**  
7442 *accept()*, *bind()*, *close()*, *getsockname()*, *poll()*, *select()*, *send()*, *shutdown()*, *socket()*, the Base  
7443 Definitions volume of IEEE Std 1003.1-200x, <**sys/socket.h**>

7444 **CHANGE HISTORY**  
7445 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.  
7446 The wording of the mandatory [ELOOP] error condition is updated, and a second optional  
7447 [ELOOP] error condition is added.

7448 **NAME**

7449 copysign, copysignf, copysignl — number manipulation function

7450 **SYNOPSIS**

7451 #include &lt;math.h&gt;

7452 double copysign(double x, double y);

7453 float copysignf(float x, float y);

7454 long double copysignl(long double x, long double y);

7455 **DESCRIPTION**

7456 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
7457 conflict between the requirements described here and the ISO C standard is unintentional. This  
7458 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

7459 These functions shall produce a value with the magnitude of *x* and the sign of *y*. On  
7460 implementations that represent a signed zero but do not treat negative zero consistently in  
7461 arithmetic operations, these functions regard the sign of zero as positive.

7462 **RETURN VALUE**

7463 Upon successful completion, these functions shall return a value with the magnitude of *x* and  
7464 the sign of *y*.

7465 **ERRORS**

7466 No errors are defined.

7467 **EXAMPLES**

7468 None.

7469 **APPLICATION USAGE**

7470 None.

7471 **RATIONALE**

7472 None.

7473 **FUTURE DIRECTIONS**

7474 None.

7475 **SEE ALSO**7476 *signbit()*, the Base Definitions volume of IEEE Std 1003.1-200x, <math.h>7477 **CHANGE HISTORY**

7478 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

7479 **NAME**

7480 cos, cosf, cosl — cosine function

7481 **SYNOPSIS**

7482 #include &lt;math.h&gt;

7483 double cos(double x);

7484 float cosf(float x);

7485 long double cosl(long double x);

7486 **DESCRIPTION**

7487 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 7488 conflict between the requirements described here and the ISO C standard is unintentional. This  
 7489 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

7490 These functions shall compute the cosine of their argument *x*, measured in radians.

7491 An application wishing to check for error situations should set *errno* to zero and call  
 7492 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 7493 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 7494 zero, an error has occurred.

7495 **RETURN VALUE**7496 Upon successful completion, these functions shall return the cosine of *x*.7497 **MX** If *x* is NaN, a NaN shall be returned.7498 If *x* is ±0, the value 1.0 shall be returned.

7499 If *x* is ±Inf, a domain error shall occur, and either a NaN (if supported), or an implementation-  
 7500 defined value shall be returned.

7501 **ERRORS**

7502 These functions shall fail if:

7503 **MX** **Domain Error** The *x* argument is ±Inf.

7504 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero, |  
 7505 then *errno* shall be set to [EDOM]. If the integer expression (math\_errhandling |  
 7506 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception |  
 7507 shall be raised. |

7508 **EXAMPLES**7509 **Taking the Cosine of a 45-Degree Angle**

7510 #include &lt;math.h&gt;

7511 ...

7512 double radians = 45 \* M\_PI / 180;

7513 double result;

7514 ...

7515 result = cos(radians);

7516 **APPLICATION USAGE**

7517 These functions may lose accuracy when their argument is near an odd multiple of  $\pi/2$  or is far  
 7518 from 0.

7519 On error, the expressions (math\_errhandling & MATH\_ERRNO) and (math\_errhandling &  
 7520 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

7521 **RATIONALE**

7522 None.

7523 **FUTURE DIRECTIONS**

7524 None.

7525 **SEE ALSO**

7526 *acos()*, *feclearexcept()*, *fetestexcept()*, *isnan()*, *sin()*, *tan()*, the Base Definitions volume of |  
7527 IEEE Std 1003.1-200x, Section 4.18, Treatment of Error Conditions for Mathematical Functions, |  
7528 **<math.h>**

7529 **CHANGE HISTORY**

7530 First released in Issue 1. Derived from Issue 1 of the SVID.

7531 **Issue 5**

7532 The DESCRIPTION is updated to indicate how an application should check for an error. This  
7533 text was previously published in the APPLICATION USAGE section.

7534 **Issue 6**7535 The *cosf()* and *cosl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

7536 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
7537 revised to align with the ISO/IEC 9899:1999 standard.

7538 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
7539 marked.

7540 **NAME**

7541       cosf — cosine function

7542 **SYNOPSIS**

7543       #include &lt;math.h&gt;

7544       float cosf(float x);

7545 **DESCRIPTION**7546       Refer to *cos()*.

7547 **NAME**

7548 cosh, coshf, coshl — hyperbolic cosine functions

7549 **SYNOPSIS**

7550 #include &lt;math.h&gt;

7551 double cosh(double x);

7552 float coshf(float x);

7553 long double coshl(long double x);

7554 **DESCRIPTION**

7555 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 7556 conflict between the requirements described here and the ISO C standard is unintentional. This  
 7557 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

7558 These functions shall compute the hyperbolic cosine of their argument  $x$ .

7559 An application wishing to check for error situations should set *errno* to zero and call  
 7560 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 7561 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 7562 zero, an error has occurred.

7563 **RETURN VALUE**7564 Upon successful completion, these functions shall return the hyperbolic cosine of  $x$ .

7565 If the correct value would cause overflow, a range error shall occur and *cosh*( $x$ ), *coshf*( $x$ ), and  
 7566 *coshl*( $x$ ) shall return the value of the macro HUGE\_VAL, HUGE\_VALF, and HUGE\_VALL,  
 7567 respectively.

7568 **MX** If  $x$  is NaN, a NaN shall be returned.7569 If  $x$  is  $\pm 0$ , the value 1.0 shall be returned.7570 If  $x$  is  $\pm\text{Inf}$ ,  $+\text{Inf}$  shall be returned.7571 **ERRORS**

7572 These functions shall fail if:

7573 **Range Error** The result would cause an overflow.

7574 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero, |  
 7575 then *errno* shall be set to [ERANGE]. If the integer expression |  
 7576 (math\_errhandling & MATH\_ERREXCEPT) is non-zero, then the overflow |  
 7577 floating-point exception shall be raised. |

7578 **EXAMPLES**

7579 None.

7580 **APPLICATION USAGE**

7581 On error, the expressions (math\_errhandling & MATH\_ERRNO) and (math\_errhandling &  
 7582 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

7583 For IEEE Std 754-1985 **double**,  $710.5 < |x|$  implies that *cosh*( $x$ ) has overflowed.7584 **RATIONALE**

7585 None.

7586 **FUTURE DIRECTIONS**

7587 None.

7588 **SEE ALSO**

7589 *acosh()*, *feclearexcept()*, *fetestexcept()*, *isnan()*, *sinh()*, *tanh()*, the Base Definitions volume of |  
7590 IEEE Std 1003.1-200x, Section 4.18, Treatment of Error Conditions for Mathematical Functions, |  
7591 **<math.h>**

7592 **CHANGE HISTORY**

7593 First released in Issue 1. Derived from Issue 1 of the SVID.

7594 **Issue 5**

7595 The DESCRIPTION is updated to indicate how an application should check for an error. This  
7596 text was previously published in the APPLICATION USAGE section.

7597 **Issue 6**

7598 The *coshf()* and *coshl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

7599 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
7600 revised to align with the ISO/IEC 9899:1999 standard.

7601 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
7602 marked.

7603 **NAME**

7604       cosl — cosine function

7605 **SYNOPSIS**

7606       #include <math.h>

7607       long double cosl(long double x);

7608 **DESCRIPTION**

7609       Refer to *cos()*.

7610 **NAME**

7611 cpow, cpowf, cpowl — complex power functions

7612 **SYNOPSIS**

7613 #include &lt;complex.h&gt;

7614 double complex cpow(double complex *x*, double complex *y*);7615 float complex cpowf(float complex *x*, float complex *y*);7616 long double complex cpowl(long double complex *x*,7617 long double complex *y*);7618 **DESCRIPTION**7619 **cx** The functionality described on this reference page is aligned with the ISO C standard. Any  
7620 conflict between the requirements described here and the ISO C standard is unintentional. This  
7621 volume of IEEE Std 1003.1-200x defers to the ISO C standard.7622 These functions shall compute the complex power function  $x^y$ , with a branch cut for the first  
7623 parameter along the negative real axis.7624 **RETURN VALUE**

7625 These functions shall return the complex power function value.

7626 **ERRORS**

7627 No errors are defined.

7628 **EXAMPLES**

7629 None.

7630 **APPLICATION USAGE**

7631 None.

7632 **RATIONALE**

7633 None.

7634 **FUTURE DIRECTIONS**

7635 None.

7636 **SEE ALSO**7637 *cabs()*, *csqrt()*, the Base Definitions volume of IEEE Std 1003.1-200x, <complex.h>7638 **CHANGE HISTORY**

7639 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

7640 **NAME**

7641 cproj, cprojf, cprojl — complex projection functions

7642 **SYNOPSIS**

7643 #include &lt;complex.h&gt;

7644 double complex cproj(double complex z);

7645 float complex cprojf(float complex z);

7646 long double complex cprojl(long double complex z);

7647 **DESCRIPTION**

7648 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 7649 conflict between the requirements described here and the ISO C standard is unintentional. This  
 7650 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

7651 These functions shall compute a projection of  $z$  onto the Riemann sphere:  $z$  projects to  $z$ , except  
 7652 that all complex infinities (even those with one infinite part and one NaN part) project to  
 7653 positive infinity on the real axis. If  $z$  has an infinite part, then  $cproj(z)$  shall be equivalent to:

7654  $INFINITY + I * copysign(0.0, cimag(z))$ 7655 **RETURN VALUE**

7656 These functions shall return the value of the projection onto the Riemann sphere.

7657 **ERRORS**

7658 No errors are defined.

7659 **EXAMPLES**

7660 None.

7661 **APPLICATION USAGE**

7662 None.

7663 **RATIONALE**

7664 Two topologies are commonly used in complex mathematics: the complex plane with its  
 7665 continuum of infinities, and the Riemann sphere with its single infinity. The complex plane is  
 7666 better suited for transcendental functions, the Riemann sphere for algebraic functions. The  
 7667 complex types with their multiplicity of infinities provide a useful (though imperfect) model for  
 7668 the complex plane. The  $cproj()$  function helps model the Riemann sphere by mapping all  
 7669 infinities to one, and should be used just before any operation, especially comparisons, that  
 7670 might give spurious results for any of the other infinities. Note that a complex value with one  
 7671 infinite part and one NaN part is regarded as an infinity, not a NaN, because if one part is  
 7672 infinite, the complex value is infinite independent of the value of the other part. For the same  
 7673 reason,  $cabs()$  returns an infinity if its argument has an infinite part and a NaN part.

7674 **FUTURE DIRECTIONS**

7675 None.

7676 **SEE ALSO**

7677  $carg()$ ,  $cimag()$ ,  $conj()$ ,  $creal()$ , the Base Definitions volume of IEEE Std 1003.1-200x,  
 7678 <complex.h>

7679 **CHANGE HISTORY**

7680 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

7681 **NAME**

7682           creal, crealf, creall — complex real functions

7683 **SYNOPSIS**

7684           #include &lt;complex.h&gt;

7685           double creal(double complex z);

7686           float crealf(float complex z);

7687           long double creall(long double complex z);

7688 **DESCRIPTION**

7689 cx       The functionality described on this reference page is aligned with the ISO C standard. Any  
7690 conflict between the requirements described here and the ISO C standard is unintentional. This  
7691 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

7692           These functions shall compute the real part of *z*.7693 **RETURN VALUE**

7694           These functions shall return the real part value.

7695 **ERRORS**

7696           No errors are defined.

7697 **EXAMPLES**

7698           None.

7699 **APPLICATION USAGE**7700           For a variable *z* of complex type:7701           *z* == creal(*z*) + cimag(*z*)\*I7702 **RATIONALE**

7703           None.

7704 **FUTURE DIRECTIONS**

7705           None.

7706 **SEE ALSO**

7707           *carg()*, *cimag()*, *conj()*, *cproj()*, the Base Definitions volume of IEEE Std 1003.1-200x,  
7708 <complex.h>

7709 **CHANGE HISTORY**

7710           First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

7711 **NAME**

7712         creat — create a new file or rewrite an existing one

7713 **SYNOPSIS**

7714 OH         #include &lt;sys/stat.h&gt;

7715         #include &lt;fcntl.h&gt;

7716         int creat(const char \*path, mode\_t mode);

7717 **DESCRIPTION**

7718         The function call:

7719         creat(path, mode)

7720         shall be equivalent to:

7721         open(path, O\_WRONLY|O\_CREAT|O\_TRUNC, mode)

7722 **RETURN VALUE**7723         Refer to *open()*.7724 **ERRORS**7725         Refer to *open()*.7726 **EXAMPLES**7727         **Creating a File**7728         The following example creates the file **/tmp/file** with read and write permissions for the file  
7729         owner and read permission for group and others. The resulting file descriptor is assigned to the  
7730         *fd* variable.

7731         #include &lt;fcntl.h&gt;

7732         ...

7733         int fd;

7734         mode\_t mode = S\_IRUSR | S\_IWUSR | S\_IRGRP | S\_IROTH;

7735         char \*filename = "/tmp/file";

7736         ...

7737         fd = creat(filename, mode);

7738         ...

7739 **APPLICATION USAGE**

7740         None.

7741 **RATIONALE**7742         The *creat()* function is redundant. Its services are also provided by the *open()* function. It has  
7743         been included primarily for historical purposes since many existing applications depend on it. It  
7744         is best considered a part of the C binding rather than a function that should be provided in other  
7745         languages.7746 **FUTURE DIRECTIONS**

7747         None.

7748 **SEE ALSO**7749         *open()*, the Base Definitions volume of IEEE Std 1003.1-200x, <fcntl.h>, <sys/stat.h>,  
7750         <sys/types.h>

7751 **CHANGE HISTORY**

7752 First released in Issue 1. Derived from Issue 1 of the SVID.

7753 **Issue 6**7754 In the SYNOPSIS, the optional include of the `<sys/types.h>` header is removed.7755 The following new requirements on POSIX implementations derive from alignment with the  
7756 Single UNIX Specification:

- 7757
- The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was  
7758 required for conforming implementations of previous POSIX specifications, it was not  
7759 required for UNIX applications.

7760 **NAME**7761 crypt — string encoding function (**CRYPT**)7762 **SYNOPSIS**7763 xSI `#include <unistd.h>`7764 `char *crypt(const char *key, const char *salt);`

7765

7766 **DESCRIPTION**7767 The *crypt()* function is a string encoding function. The algorithm is implementation-defined.7768 The *key* argument points to a string to be encoded. The *salt* argument is a string chosen from the  
7769 set:

7770 a b c d e f g h i j k l m n o p q r s t u v w x y z

7771 A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

7772 0 1 2 3 4 5 6 7 8 9 . /

7773 The first two characters of this string may be used to perturb the encoding algorithm.

7774 The return value of *crypt()* points to static data that is overwritten by each call.7775 The *crypt()* function need not be reentrant. A function that is not required to be reentrant is not  
7776 required to be thread-safe.7777 **RETURN VALUE**7778 Upon successful completion, *crypt()* shall return a pointer to the encoded string. The first two  
7779 characters of the returned value shall be those of the *salt* argument. Otherwise, it shall return a  
7780 null pointer and set *errno* to indicate the error.7781 **ERRORS**7782 The *crypt()* function shall fail if:

7783 [ENOSYS] The functionality is not supported on this implementation.

7784 **EXAMPLES**7785 **Encoding Passwords**7786 The following example finds a user database entry matching a particular user name and changes  
7787 the current password to a new password. The *crypt()* function generates an encoded version of  
7788 each password. The first call to *crypt()* produces an encoded version of the old password; that  
7789 encoded password is then compared to the password stored in the user database. The second  
7790 call to *crypt()* encodes the new password before it is stored.7791 The *putpwent()* function, used in the following example, is not part of IEEE Std 1003.1-200x.7792 `#include <unistd.h>`7793 `#include <pwd.h>`7794 `#include <string.h>`7795 `#include <stdio.h>`7796 `...`7797 `int valid_change;`7798 `int pfd; /* Integer for file descriptor returned by open(). */`7799 `FILE *fpfd; /* File pointer for use in putpwent(). */`7800 `struct passwd *p;`7801 `char user[100];`7802 `char oldpasswd[100];`7803 `char newpasswd[100];`

```
7804     char savepasswd[100];
7805     ...
7806     valid_change = 0;
7807     while ((p = getpwent()) != NULL) {
7808         /* Change entry if found. */
7809         if (strcmp(p->pw_name, user) == 0) {
7810             if (strcmp(p->pw_passwd, crypt(oldpasswd, p->pw_passwd)) == 0) {
7811                 strcpy(savepasswd, crypt(newpasswd, user));
7812                 p->pw_passwd = savepasswd;
7813                 valid_change = 1;
7814             }
7815             else {
7816                 fprintf(stderr, "Old password is not valid\n");
7817             }
7818         }
7819         /* Put passwd entry into ptmp. */
7820         putpwent(p, fpfd);
7821     }
```

7822 **APPLICATION USAGE**

7823 The values returned by this function need not be portable among XSI-conformant systems.

7824 **RATIONALE**

7825 None.

7826 **FUTURE DIRECTIONS**

7827 None.

7828 **SEE ALSO**

7829 *encrypt()*, *setkey()*, the Base Definitions volume of IEEE Std 1003.1-200x, <**unistd.h**>

7830 **CHANGE HISTORY**

7831 First released in Issue 1. Derived from Issue 1 of the SVID.

7832 **Issue 5**

7833 Normative text previously in the APPLICATION USAGE section is moved to the  
7834 DESCRIPTION.

7835 **NAME**

7836       csin, csinf, csinl — complex sine functions

7837 **SYNOPSIS**

7838       #include &lt;complex.h&gt;

7839       double complex csin(double complex z);

7840       float complex csinf(float complex z);

7841       long double complex csinl(long double complex z);

7842 **DESCRIPTION**7843 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
7844       conflict between the requirements described here and the ISO C standard is unintentional. This  
7845       volume of IEEE Std 1003.1-200x defers to the ISO C standard.7846       These functions shall compute the complex sine of *z*.7847 **RETURN VALUE**

7848       These functions shall return the complex sine value.

7849 **ERRORS**

7850       No errors are defined.

7851 **EXAMPLES**

7852       None.

7853 **APPLICATION USAGE**

7854       None.

7855 **RATIONALE**

7856       None.

7857 **FUTURE DIRECTIONS**

7858       None.

7859 **SEE ALSO**7860       *casin()*, the Base Definitions volume of IEEE Std 1003.1-200x, <complex.h>7861 **CHANGE HISTORY**

7862       First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

7863 **NAME**7864 `csinf` — complex sine functions7865 **SYNOPSIS**7866 `#include <complex.h>`7867 `float complex csinf(float complex z);`7868 **DESCRIPTION**7869 Refer to *csin()*.

7870 **NAME**

7871       csinh, csinhf, csinhl — complex hyperbolic sine functions

7872 **SYNOPSIS**

7873       #include &lt;complex.h&gt;

7874       double complex csinh(double complex z);

7875       float complex csinhf(float complex z);

7876       long double complex csinhl(long double complex z);

7877 **DESCRIPTION**7878 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
7879       conflict between the requirements described here and the ISO C standard is unintentional. This  
7880       volume of IEEE Std 1003.1-200x defers to the ISO C standard.7881       These functions shall compute the complex hyperbolic sine of *z*.7882 **RETURN VALUE**

7883       These functions shall return the complex hyperbolic sine value.

7884 **ERRORS**

7885       No errors are defined.

7886 **EXAMPLES**

7887       None.

7888 **APPLICATION USAGE**

7889       None.

7890 **RATIONALE**

7891       None.

7892 **FUTURE DIRECTIONS**

7893       None.

7894 **SEE ALSO**7895       *casinh()*, the Base Definitions volume of IEEE Std 1003.1-200x, <complex.h>7896 **CHANGE HISTORY**

7897       First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

7898 **NAME**

7899       csinl — complex sine functions

7900 **SYNOPSIS**

7901       #include &lt;complex.h&gt;

7902       long double complex csinl(long double complex z);

7903 **DESCRIPTION**7904       Refer to *csin()*.

7905 **NAME**

7906 csqrt, csqrtf, csqrtl — complex square root functions

7907 **SYNOPSIS**

7908 #include &lt;complex.h&gt;

7909 double complex csqrt(double complex z);

7910 float complex csqrtf(float complex z);

7911 long double complex csqrtl(long double complex z);

7912 **DESCRIPTION**

7913 **cx** The functionality described on this reference page is aligned with the ISO C standard. Any  
7914 conflict between the requirements described here and the ISO C standard is unintentional. This  
7915 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

7916 These functions shall compute the complex square root of *z*, with a branch cut along the  
7917 negative real axis.

7918 **RETURN VALUE**

7919 These functions shall return the complex square root value, in the range of the right half-plane  
7920 (including the imaginary axis).

7921 **ERRORS**

7922 No errors are defined.

7923 **EXAMPLES**

7924 None.

7925 **APPLICATION USAGE**

7926 None.

7927 **RATIONALE**

7928 None.

7929 **FUTURE DIRECTIONS**

7930 None.

7931 **SEE ALSO**7932 *cabs()*, *cpow()*, the Base Definitions volume of IEEE Std 1003.1-200x, <complex.h>7933 **CHANGE HISTORY**

7934 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

7935 **NAME**

7936           ctan, ctanf, ctanl — complex tangent functions

7937 **SYNOPSIS**

7938           #include &lt;complex.h&gt;

7939           double complex ctan(double complex *z*);7940           float complex ctanf(float complex *z*);7941           long double complex ctanl(long double complex *z*);7942 **DESCRIPTION**7943 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
7944 conflict between the requirements described here and the ISO C standard is unintentional. This  
7945 volume of IEEE Std 1003.1-200x defers to the ISO C standard.7946           These functions shall compute the complex tangent of *z*.7947 **RETURN VALUE**

7948           These functions shall return the complex tangent value.

7949 **ERRORS**

7950           No errors are defined.

7951 **EXAMPLES**

7952           None.

7953 **APPLICATION USAGE**

7954           None.

7955 **RATIONALE**

7956           None.

7957 **FUTURE DIRECTIONS**

7958           None.

7959 **SEE ALSO**7960           *catan()*, the Base Definitions volume of IEEE Std 1003.1-200x, <complex.h>7961 **CHANGE HISTORY**

7962           First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

7963 **NAME**

7964       ctanf — complex tangent functions

7965 **SYNOPSIS**

7966       #include &lt;complex.h&gt;

7967       float complex ctanf(float complex z);

7968 **DESCRIPTION**7969       Refer to *ctan()*.

7970 **NAME**

7971 ctanh, ctanhf, ctanhl — complex hyperbolic tangent functions

7972 **SYNOPSIS**

7973 #include &lt;complex.h&gt;

7974 double complex ctanh(double complex z);

7975 float complex ctanhf(float complex z);

7976 long double complex ctanhl(long double complex z);

7977 **DESCRIPTION**

7978 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
7979 conflict between the requirements described here and the ISO C standard is unintentional. This  
7980 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

7981 These functions shall compute the complex hyperbolic tangent of  $z$ .7982 **RETURN VALUE**

7983 These functions shall return the complex hyperbolic tangent value.

7984 **ERRORS**

7985 No errors are defined.

7986 **EXAMPLES**

7987 None.

7988 **APPLICATION USAGE**

7989 None.

7990 **RATIONALE**

7991 None.

7992 **FUTURE DIRECTIONS**

7993 None.

7994 **SEE ALSO**7995 *catanh()*, the Base Definitions volume of IEEE Std 1003.1-200x, <complex.h>7996 **CHANGE HISTORY**

7997 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

7998 **NAME**

7999       ctanl — complex tangent functions

8000 **SYNOPSIS**

8001       #include &lt;complex.h&gt;

8002       long double complex ctanl(long double complex z);

8003 **DESCRIPTION**8004       Refer to *ctan()*.

8005 **NAME**

8006 ctermid — generate a pathname for controlling terminal |

8007 **SYNOPSIS**

8008 cx #include &lt;stdio.h&gt; |

8009 char \*ctermid(char \*s); |

8010 |

8011 **DESCRIPTION**

8012 The *ctermid()* function shall generate a string that, when used as a pathname, refers to the |  
8013 current controlling terminal for the current process. If *ctermid()* returns a pathname, access to the |  
8014 file is not guaranteed. |

8015 If the application uses any of the `_POSIX_THREAD_SAFE_FUNCTIONS` or `_POSIX_THREADS` |  
8016 functions, it shall ensure that the *ctermid()* function is called with a non-NULL parameter. |

8017 **RETURN VALUE**

8018 If *s* is a null pointer, the string shall be generated in an area that may be static (and therefore may |  
8019 be overwritten by each call), the address of which shall be returned. Otherwise, *s* is assumed to |  
8020 point to a character array of at least `L_ctermid` bytes; the string is placed in this array and the |  
8021 value of *s* shall be returned. The symbolic constant `L_ctermid` is defined in `<stdio.h>`, and shall |  
8022 have a value greater than 0. |

8023 The *ctermid()* function shall return an empty string if the pathname that would refer to the |  
8024 controlling terminal cannot be determined, or if the function is unsuccessful. |

8025 **ERRORS**

8026 No errors are defined.

8027 **EXAMPLES**8028 **Determining the Controlling Terminal for the Current Process**

8029 The following example returns a pointer to a string that identifies the controlling terminal for the |  
8030 current process. The pathname for the terminal is stored in the array pointed to by the *ptr* |  
8031 argument, which has a size of `L_ctermid` bytes, as indicated by the *term* argument. |

8032 #include &lt;stdio.h&gt;

8033 ...

8034 char term[L\_ctermid];

8035 char \*ptr;

8036 ptr = ctermid(term);

8037 **APPLICATION USAGE**

8038 The difference between *ctermid()* and *ttyname()* is that *ttyname()* must be handed a file |  
8039 descriptor and return a path of the terminal associated with that file descriptor, while *ctermid()* |  
8040 returns a string (such as `"/dev/tty"`) that refers to the current controlling terminal if used as a |  
8041 pathname. |

8042 **RATIONALE**

8043 `L_ctermid` must be defined appropriately for a given implementation and must be greater than |  
8044 zero so that array declarations using it are accepted by the compiler. The value includes the |  
8045 terminating null byte. |

8046 Conforming applications that use threads cannot call *ctermid()* with NULL as the parameter if |  
8047 either `_POSIX_THREAD_SAFE_FUNCTIONS` or `_POSIX_THREADS` is defined. If *s* is not |  
8048 NULL, the *ctermid()* function generates a string that, when used as a pathname, refers to the |

8049 current controlling terminal for the current process. If *s* is NULL, the return value of *ctermid()* is  
8050 undefined.

8051 There is no additional burden on the programmer—changing to use a hypothetical thread-safe  
8052 version of *ctermid()* along with allocating a buffer is more of a burden than merely allocating a  
8053 buffer. Application code should not assume that the returned string is short, as some  
8054 implementations have more than two pathname components before reaching a logical device  
8055 name. |

8056 **FUTURE DIRECTIONS**

8057 None.

8058 **SEE ALSO**

8059 *ttyname()*, the Base Definitions volume of IEEE Std 1003.1-200x, <**stdio.h**>

8060 **CHANGE HISTORY**

8061 First released in Issue 1. Derived from Issue 1 of the SVID.

8062 **Issue 5**

8063 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

8064 **Issue 6**

8065 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

8066 **NAME**

8067 ctime, ctime\_r — convert a time value to date and time string

8068 **SYNOPSIS**

8069 #include &lt;time.h&gt;

8070 char \*ctime(const time\_t \*clock);

8071 TSF char \*ctime\_r(const time\_t \*clock, char \*buf);

8072

8073 **DESCRIPTION**8074 CX For *ctime()*: The functionality described on this reference page is aligned with the ISO C |  
8075 standard. Any conflict between the requirements described here and the ISO C standard is |  
8076 unintentional. This volume of IEEE Std 1003.1-200x defers to the ISO C standard.8077 The *ctime()* function shall convert the time pointed to by *clock*, representing time in seconds |  
8078 since the Epoch, to local time in the form of a string. It shall be equivalent to: |

8079 asctime(localtime(clock))

8080 CX The *asctime()*, *ctime()*, *gmtime()*, and *localtime()* functions shall return values in one of two static |  
8081 objects: a broken-down time structure and an array of **char**. Execution of any of the functions |  
8082 may overwrite the information returned in either of these objects by any of the other functions.8083 The *ctime()* function need not be reentrant. A function that is not required to be reentrant is not |  
8084 required to be thread-safe.8085 TSF The *ctime\_r()* function shall convert the calendar time pointed to by *clock* to local time in exactly |  
8086 the same form as *ctime()* and puts the string into the array pointed to by *buf* (which shall be at |  
8087 least 26 bytes in size) and return *buf*.8088 Unlike *ctime()*, the thread-safe version *ctime\_r()* is not required to set *tzname*.8089 **RETURN VALUE**8090 The *ctime()* function shall return the pointer returned by *asctime()* with that broken-down time |  
8091 as an argument.8092 TSF Upon successful completion, *ctime\_r()* shall return a pointer to the string pointed to by *buf*. |  
8093 When an error is encountered, a null pointer shall be returned.8094 **ERRORS**

8095 No errors are defined.

8096 **EXAMPLES**

8097 None.

8098 **APPLICATION USAGE**8099 Values for the broken-down time structure can be obtained by calling *gmtime()* or *localtime()*. |  
8100 The *ctime()* function is included for compatibility with older implementations, and does not |  
8101 support localized date and time formats. Applications should use the *strptime()* function to |  
8102 achieve maximum portability.8103 The *ctime\_r()* function is thread-safe and shall return values in a user-supplied buffer instead of |  
8104 possibly using a static data area that may be overwritten by each call.8105 **RATIONALE**

8106 None.

8107 **FUTURE DIRECTIONS**

8108 None.

8109 **SEE ALSO**8110 *asctime()*, *clock()*, *difftime()*, *gmtime()*, *localtime()*, *mktime()*, *strftime()*, *strptime()*, *time()*, *utime()*,  
8111 the Base Definitions volume of IEEE Std 1003.1-200x, <**time.h**>8112 **CHANGE HISTORY**

8113 First released in Issue 1. Derived from Issue 1 of the SVID.

8114 **Issue 5**8115 Normative text previously in the APPLICATION USAGE section is moved to the  
8116 DESCRIPTION.8117 The *ctime\_r()* function is included for alignment with the POSIX Threads Extension.8118 A note indicating that the *ctime()* function need not be reentrant is added to the DESCRIPTION.8119 **Issue 6**

8120 Extensions beyond the ISO C standard are now marked.

8121 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

8122 The APPLICATION USAGE section is updated to include a note on the thread-safe function and  
8123 its avoidance of possibly using a static data area.

8124 **NAME**

8125 daylight — daylight savings time flag

8126 **SYNOPSIS**

8127 xSI #include <time.h>

8128 extern int daylight;

8129

8130 **DESCRIPTION**

8131 Refer to *tzset()*.

## 8132 NAME

8133 dbm\_clearerr, dbm\_close, dbm\_delete, dbm\_error, dbm\_fetch, dbm\_firstkey, dbm\_nextkey,  
8134 dbm\_open, dbm\_store — database functions

## 8135 SYNOPSIS

```
8136 xSI #include <ndbm.h>
8137
8137 int dbm_clearerr(DBM *db);
8138 void dbm_close(DBM *db);
8139 int dbm_delete(DBM *db, datum key);
8140 int dbm_error(DBM *db);
8141 datum dbm_fetch(DBM *db, datum key);
8142 datum dbm_firstkey(DBM *db);
8143 datum dbm_nextkey(DBM *db);
8144 DBM *dbm_open(const char *file, int open_flags, mode_t file_mode);
8145 int dbm_store(DBM *db, datum key, datum content, int store_mode);
8146
```

## 8147 DESCRIPTION

8148 These functions create, access, and modify a database.

8149 A **datum** consists of at least two members, *dptr* and *dsize*. The *dptr* member points to an object  
8150 that is *dsize* bytes in length. Arbitrary binary data, as well as character strings, may be stored in  
8151 the object pointed to by *dptr*.

8152 The database is stored in two files. One file is a directory containing a bit map of keys and has  
8153 **.dir** as its suffix. The second file contains all data and has **.pag** as its suffix.

8154 The *dbm\_open()* function shall open a database. The *file* argument to the function is the  
8155 pathname of the database. The function opens two files named *file.dir* and *file.pag*. The  
8156 *open\_flags* argument has the same meaning as the *flags* argument of *open()* except that a database  
8157 opened for write-only access opens the files for read and write access and the behavior of the  
8158 O\_APPEND flag is unspecified. The *file\_mode* argument has the same meaning as the third  
8159 argument of *open()*.

8160 The *dbm\_close()* function shall close a database. The application shall ensure that argument *db* is  
8161 a pointer to a **dbm** structure that has been returned from a call to *dbm\_open()*.

8162 These database functions shall support an internal block size large enough to support  
8163 key/content pairs of at least 1023 bytes.

8164 The *dbm\_fetch()* function shall read a record from a database. The argument *db* is a pointer to a  
8165 database structure that has been returned from a call to *dbm\_open()*. The argument *key* is a  
8166 **datum** that has been initialized by the application to the value of the key that matches the key of  
8167 the record the program is fetching.

8168 The *dbm\_store()* function shall write a record to a database. The argument *db* is a pointer to a  
8169 database structure that has been returned from a call to *dbm\_open()*. The argument *key* is a  
8170 **datum** that has been initialized by the application to the value of the key that identifies (for  
8171 subsequent reading, writing, or deleting) the record the application is writing. The argument  
8172 *content* is a **datum** that has been initialized by the application to the value of the record the  
8173 program is writing. The argument *store\_mode* controls whether *dbm\_store()* replaces any pre-  
8174 existing record that has the same key that is specified by the *key* argument. The application shall  
8175 set *store\_mode* to either DBM\_INSERT or DBM\_REPLACE. If the database contains a record that  
8176 matches the *key* argument and *store\_mode* is DBM\_REPLACE, the existing record shall be  
8177 replaced with the new record. If the database contains a record that matches the *key* argument  
8178 and *store\_mode* is DBM\_INSERT, the existing record shall be left unchanged and the new record

8179 ignored. If the database does not contain a record that matches the *key* argument and *store\_mode* |  
8180 is either DBM\_INSERT or DBM\_REPLACE, the new record shall be inserted in the database. |

8181 If the sum of a key/content pair exceeds the internal block size, the result is unspecified. |  
8182 Moreover, the application shall ensure that all key/content pairs that hash together fit on a |  
8183 single block. The *dbm\_store()* function shall return an error in the event that a disk block fills |  
8184 with inseparable data.

8185 The *dbm\_delete()* function shall delete a record and its key from the database. The argument *db* is  
8186 a pointer to a database structure that has been returned from a call to *dbm\_open()*. The argument  
8187 *key* is a **datum** that has been initialized by the application to the value of the key that identifies  
8188 the record the program is deleting.

8189 The *dbm\_firstkey()* function shall return the first key in the database. The argument *db* is a  
8190 pointer to a database structure that has been returned from a call to *dbm\_open()*.

8191 The *dbm\_nextkey()* function shall return the next key in the database. The argument *db* is a  
8192 pointer to a database structure that has been returned from a call to *dbm\_open()*. The application  
8193 shall ensure that the *dbm\_firstkey()* function is called before calling *dbm\_nextkey()*. Subsequent  
8194 calls to *dbm\_nextkey()* return the next key until all of the keys in the database have been  
8195 returned.

8196 The *dbm\_error()* function shall return the error condition of the database. The argument *db* is a  
8197 pointer to a database structure that has been returned from a call to *dbm\_open()*.

8198 The *dbm\_clearerr()* function shall clear the error condition of the database. The argument *db* is a  
8199 pointer to a database structure that has been returned from a call to *dbm\_open()*.

8200 The *dptr* pointers returned by these functions may point into static storage that may be changed |  
8201 by subsequent calls.

8202 These functions need not be reentrant. A function that is not required to be reentrant is not  
8203 required to be thread-safe.

8204 **RETURN VALUE**

8205 The *dbm\_store()* and *dbm\_delete()* functions shall return 0 when they succeed and a negative  
8206 value when they fail.

8207 The *dbm\_store()* function shall return 1 if it is called with a *flags* value of DBM\_INSERT and the  
8208 function finds an existing record with the same key.

8209 The *dbm\_error()* function shall return 0 if the error condition is not set and return a non-zero  
8210 value if the error condition is set.

8211 The return value of *dbm\_clearerr()* is unspecified.

8212 The *dbm\_firstkey()* and *dbm\_nextkey()* functions shall return a key **datum**. When the end of the  
8213 database is reached, the *dptr* member of the key is a null pointer. If an error is detected, the *dptr*  
8214 member of the key shall be a null pointer and the error condition of the database shall be set.

8215 The *dbm\_fetch()* function shall return a content **datum**. If no record in the database matches the  
8216 key or if an error condition has been detected in the database, the *dptr* member of the content  
8217 shall be a null pointer.

8218 The *dbm\_open()* function shall return a pointer to a database structure. If an error is detected  
8219 during the operation, *dbm\_open()* shall return a **(DBM \*)0**.

**8220 ERRORS**

8221 No errors are defined.

**8222 EXAMPLES**

8223 None.

**8224 APPLICATION USAGE**

8225 The following code can be used to traverse the database:

```
8226 for(key = dbm_firstkey(db); key.dptr != NULL; key = dbm_nextkey(db))
```

8227 The *dbm\_* functions provided in this library should not be confused in any way with those of a  
8228 general-purpose database management system. These functions do not provide for multiple  
8229 search keys per entry, they do not protect against multi-user access (in other words they do not  
8230 lock records or files), and they do not provide the many other useful database functions that are  
8231 found in more robust database management systems. Creating and updating databases by use of  
8232 these functions is relatively slow because of data copies that occur upon hash collisions. These  
8233 functions are useful for applications requiring fast lookup of relatively static information that is  
8234 to be indexed by a single key.

8235 The *dbm\_delete()* function need not physically reclaim file space, although it does make it  
8236 available for reuse by the database.

8237 After calling *dbm\_store()* or *dbm\_delete()* during a pass through the keys by *dbm\_firstkey()* and  
8238 *dbm\_nextkey()*, the application should reset the database by calling *dbm\_firstkey()* before again  
8239 calling *dbm\_nextkey()*. The contents of these files are unspecified and may not be portable.

**8240 RATIONALE**

8241 None.

**8242 FUTURE DIRECTIONS**

8243 None.

**8244 SEE ALSO**

8245 *open()*, the Base Definitions volume of IEEE Std 1003.1-200x, <**ndbm.h**>

**8246 CHANGE HISTORY**

8247 First released in Issue 4, Version 2.

**8248 Issue 5**

8249 Moved from X/OPEN UNIX extension to BASE.

8250 Normative text previously in the APPLICATION USAGE section is moved to the  
8251 DESCRIPTION.

8252 A note indicating that these functions need not be reentrant is added to the DESCRIPTION.

**8253 Issue 6**

8254 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

8255 **NAME**

8256 difftime — compute the difference between two calendar time values

8257 **SYNOPSIS**

8258 #include <time.h>

8259 double difftime(time\_t *time1*, time\_t *time0*);

8260 **DESCRIPTION**

8261 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
8262 conflict between the requirements described here and the ISO C standard is unintentional. This  
8263 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

8264 The *difftime()* function shall compute the difference between two calendar times (as returned by  
8265 *time()*): *time1*– *time0*.

8266 **RETURN VALUE**

8267 The *difftime()* function shall return the difference expressed in seconds as a type **double**.

8268 **ERRORS**

8269 No errors are defined.

8270 **EXAMPLES**

8271 None.

8272 **APPLICATION USAGE**

8273 None.

8274 **RATIONALE**

8275 None.

8276 **FUTURE DIRECTIONS**

8277 None.

8278 **SEE ALSO**

8279 *asctime()*, *clock()*, *ctime()*, *gmtime()*, *localtime()*, *mktime()*, *strftime()*, *strptime()*, *time()*, *utime()*,  
8280 the Base Definitions volume of IEEE Std 1003.1-200x, <**time.h**>

8281 **CHANGE HISTORY**

8282 First released in Issue 4. Derived from the ISO C standard.

8283 **NAME**8284 `dirname` — report the parent directory name of a file pathname |8285 **SYNOPSIS**8286 `XSI` `#include <libgen.h>`8287 `char *dirname(char *path);`

8288

8289 **DESCRIPTION**8290 The `dirname()` function shall take a pointer to a character string that contains a pathname, and |  
8291 return a pointer to a string that is a pathname of the parent directory of that file. Trailing '/' |  
8292 characters in the path are not counted as part of the path.8293 If `path` does not contain a '/', then `dirname()` shall return a pointer to the string ".". If `path` is a  
8294 null pointer or points to an empty string, `dirname()` shall return a pointer to the string ".".8295 The `dirname()` function need not be reentrant. A function that is not required to be reentrant is  
8296 not required to be thread-safe.8297 **RETURN VALUE**8298 The `dirname()` function shall return a pointer to a string that is the parent directory of `path`. If  
8299 `path` is a null pointer or points to an empty string, a pointer to a string "." is returned.8300 The `dirname()` function may modify the string pointed to by `path`, and may return a pointer to  
8301 static storage that may then be overwritten by subsequent calls to `dirname()`.8302 **ERRORS**

8303 No errors are defined.

8304 **EXAMPLES**8305 The following code fragment reads a pathname, changes the current working directory to the |  
8306 parent directory, and opens the file.8307 `char path[MAXPATHLEN], *pathcopy;`  
8308 `int fd;`  
8309 `fgets(path, MAXPATHLEN, stdin);`  
8310 `pathcopy = strdup(path);`  
8311 `chdir(dirname(pathcopy));`  
8312 `fd = open(basename(path), O_RDONLY);`8313 **Sample Input and Output Strings for `dirname()`**8314 In the following table, the input string is the value pointed to by `path`, and the output string is  
8315 the return value of the `dirname()` function.

Input String	Output String
"/usr/lib"	"/usr"
"/usr/"	"/"
"usr"	."
"/"	"/"
."	."
".."	."

8316

8317

8318

8319

8320

8321

8322

8323 **Changing the Current Directory to the Parent Directory**

8324 The following program fragment reads a pathname, changes the current working directory to |  
8325 the parent directory, and opens the file.

```
8326 #include <unistd.h>
8327 #include <limits.h>
8328 #include <stdio.h>
8329 #include <fcntl.h>
8330 #include <string.h>
8331 #include <libgen.h>
8332 ...
8333 char path[PATH_MAX], *pathcopy;
8334 int fd;
8335 ...
8336 fgets(path, PATH_MAX, stdin);
8337 pathcopy = strdup(path);
8338 chdir(dirname(pathcopy));
8339 fd = open(basename(path), O_RDONLY);
```

8340 **APPLICATION USAGE**

8341 The *dirname()* and *basename()* functions together yield a complete pathname. The expression |  
8342 *dirname(path)* obtains the pathname of the directory where *basename(path)* is found. |

8343 Since the meaning of the leading `"/"` is implementation-defined, *dirname("/foo)* may return  
8344 either `"/"` or `'/'` (but nothing else).

8345 **RATIONALE**

8346 None.

8347 **FUTURE DIRECTIONS**

8348 None.

8349 **SEE ALSO**

8350 *basename()*, the Base Definitions volume of IEEE Std 1003.1-200x, `<libgen.h>`

8351 **CHANGE HISTORY**

8352 First released in Issue 4, Version 2.

8353 **Issue 5**

8354 Moved from X/OPEN UNIX extension to BASE.

8355 Normative text previously in the APPLICATION USAGE section is moved to the  
8356 DESCRIPTION.

8357 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

8358 **NAME**8359 `div` — compute the quotient and remainder of an integer division8360 **SYNOPSIS**8361 `#include <stdlib.h>`8362 `div_t div(int numer, int denom);`8363 **DESCRIPTION**

8364 `CX` The functionality described on this reference page is aligned with the ISO C standard. Any  
8365 conflict between the requirements described here and the ISO C standard is unintentional. This  
8366 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

8367 The `div()` function shall compute the quotient and remainder of the division of the numerator  
8368 `numer` by the denominator `denom`. If the division is inexact, the resulting quotient is the integer  
8369 of lesser magnitude that is the nearest to the algebraic quotient. If the result cannot be  
8370 represented, the behavior is undefined; otherwise, `quot*denom+rem` shall equal `numer`.

8371 **RETURN VALUE**

8372 The `div()` function shall return a structure of type `div_t`, comprising both the quotient and the  
8373 remainder. The structure includes the following members, in any order:

8374 `int quot; /* quotient */`8375 `int rem; /* remainder */`8376 **ERRORS**

8377 No errors are defined.

8378 **EXAMPLES**

8379 None.

8380 **APPLICATION USAGE**

8381 None.

8382 **RATIONALE**

8383 None.

8384 **FUTURE DIRECTIONS**

8385 None.

8386 **SEE ALSO**8387 `ldiv()`, the Base Definitions volume of IEEE Std 1003.1-200x, `<stdlib.h>`8388 **CHANGE HISTORY**

8389 First released in Issue 4. Derived from the ISO C standard.

## 8390 NAME

8391 dldclose — close a *dlopen()* object

## 8392 SYNOPSIS

8393 XSI #include &lt;dldfcn.h&gt;

8394 int dldclose(void \*handle);

8395

## 8396 DESCRIPTION

8397 The *dldclose()* function shall inform the system that the object referenced by a *handle* returned |  
8398 from a previous *dlopen()* invocation is no longer needed by the application.8399 The use of *dldclose()* reflects a statement of intent on the part of the process, but does not create  
8400 any requirement upon the implementation, such as removal of the code or symbols referenced  
8401 by *handle*. Once an object has been closed using *dldclose()* an application should assume that its  
8402 symbols are no longer available to *dlsym()*. All objects loaded automatically as a result of  
8403 invoking *dlopen()* on the referenced object shall also be closed if this is the last reference to it. |8404 Although a *dldclose()* operation is not required to remove structures from an address space,  
8405 neither is an implementation prohibited from doing so. The only restriction on such a removal is  
8406 that no object shall be removed to which references have been relocated, until or unless all such  
8407 references are removed. For instance, an object that had been loaded with a *dlopen()* operation  
8408 specifying the RTLD\_GLOBAL flag might provide a target for dynamic relocations performed in  
8409 the processing of other objects—in such environments, an application may assume that no  
8410 relocation, once made, shall be undone or remade unless the object requiring the relocation has  
8411 itself been removed.

## 8412 RETURN VALUE

8413 If the referenced object was successfully closed, *dldclose()* shall return 0. If the object could not be  
8414 closed, or if *handle* does not refer to an open object, *dldclose()* shall return a non-zero value. More  
8415 detailed diagnostic information shall be available through *dlderror()*.

## 8416 ERRORS

8417 No errors are defined.

## 8418 EXAMPLES

8419 The following example illustrates use of *dlopen()* and *dldclose()*:8420 ...  
8421 /\* Open a dynamic library and then close it ... \*/  
8422 #include <dldfcn.h>  
8423 void \*mylib;  
8424 int eret;  
8425 mylib = dlopen("mylib.so", RTLD\_LOCAL | RTLD\_LAZY); |  
8426 ... |  
8427 eret = dldclose(mylib);  
8428 ...

## 8429 APPLICATION USAGE

8430 A conforming application should employ a *handle* returned from a *dlopen()* invocation only |  
8431 within a given scope bracketed by the *dlopen()* and *dldclose()* operations. Implementations are  
8432 free to use reference counting or other techniques such that multiple calls to *dlopen()* referencing  
8433 the same object may return the same object for *handle*. Implementations are also free to reuse a  
8434 *handle*. For these reasons, the value of a *handle* must be treated as an opaque object by the  
8435 application, used only in calls to *dlsym()* and *dldclose()*.

8436 **RATIONALE**

8437       None.

8438 **FUTURE DIRECTIONS**

8439       None.

8440 **SEE ALSO**8441       *derror()*, *dlopen()*, *dlsym()*, the Base Definitions volume of IEEE Std 1003.1-200x, <dlfcn.h>8442 **CHANGE HISTORY**

8443       First released in Issue 5.

8444 **Issue 6**8445       The DESCRIPTION is updated to say that the referenced object is closed “if this is the last  
8446       reference to it”.

8447 **NAME**8448 `dlderror` — get diagnostic information8449 **SYNOPSIS**8450 XSI `#include <dldfcn.h>`8451 `char *dlderror(void);`

8452

8453 **DESCRIPTION**

8454 The `dlderror()` function shall return a null-terminated character string (with no trailing <newline>) that describes the last error that occurred during dynamic linking processing. If no dynamic linking errors have occurred since the last invocation of `dlderror()`, `dlderror()` shall return NULL. Thus, invoking `dlderror()` a second time, immediately following a prior invocation, shall result in NULL being returned.

8459 The `dlderror()` function need not be reentrant. A function that is not required to be reentrant is not required to be thread-safe.

8461 **RETURN VALUE**

8462 If successful, `dlderror()` shall return a null-terminated character string; otherwise, NULL shall be returned.

8464 **ERRORS**

8465 No errors are defined.

8466 **EXAMPLES**

8467 The following example prints out the last dynamic linking error:

```
8468 ...
8469 #include <dldfcn.h>
8470 char *errstr;
8471 errstr = dlderror();
8472 if (errstr != NULL)
8473     printf ("A dynamic linking error occurred: (%s)\n", errstr);
8474 ...
```

8475 **APPLICATION USAGE**

8476 The messages returned by `dlderror()` may reside in a static buffer that is overwritten on each call to `dlderror()`. Application code should not write to this buffer. Programs wishing to preserve an error message should make their own copies of that message. Depending on the application environment with respect to asynchronous execution events, such as signals or other asynchronous computation sharing the address space, conforming applications should use a critical section to retrieve the error pointer and buffer.

8482 **RATIONALE**

8483 None.

8484 **FUTURE DIRECTIONS**

8485 None.

8486 **SEE ALSO**8487 `dldclose()`, `dldopen()`, `dldsym()`, the Base Definitions volume of IEEE Std 1003.1-200x, <dldfcn.h>

8488 **CHANGE HISTORY**

8489 First released in Issue 5.

8490 **Issue 6**

8491 In the DESCRIPTION the note about reentrancy and thread-safety is added.

## 8492 NAME

8493 dlopen — gain access to an executable object file

## 8494 SYNOPSIS

8495 XSI `#include <dlfcn.h>`8496 `void *dlopen(const char *file, int mode);`

8497

## 8498 DESCRIPTION

8499 The *dlopen()* function shall make an executable object file specified by *file* available to the calling  
 8500 program. The class of files eligible for this operation and the manner of their construction are  
 8501 implementation-defined, though typically such files are executable objects such as shared  
 8502 libraries, relocatable files, or programs. Note that some implementations permit the construction  
 8503 of dependencies between such objects that are embedded within files. In such cases, a *dlopen()*  
 8504 operation shall load such dependencies in addition to the object referenced by *file*.  
 8505 Implementations may also impose specific constraints on the construction of programs that can  
 8506 employ *dlopen()* and its related services.

8507 A successful *dlopen()* shall return a *handle* which the caller may use on subsequent calls to  
 8508 *dlsym()* and *dlclose()*. The value of this *handle* should not be interpreted in any way by the caller.

8509 The *file* argument is used to construct a pathname to the object file. If *file* contains a slash  
 8510 character, the *file* argument is used as the pathname for the file. Otherwise, *file* is used in an  
 8511 implementation-defined manner to yield a pathname.

8512 If the value of *file* is 0, *dlopen()* shall provide a *handle* on a global symbol object. This object shall  
 8513 provide access to the symbols from an ordered set of objects consisting of the original program  
 8514 image file, together with any objects loaded at program start-up as specified by that process  
 8515 image file (for example, shared libraries), and the set of objects loaded using a *dlopen()* operation  
 8516 together with the RTLD\_GLOBAL flag. As the latter set of objects can change during execution,  
 8517 the set identified by *handle* can also change dynamically.

8518 Only a single copy of an object file is brought into the address space, even if *dlopen()* is invoked  
 8519 multiple times in reference to the file, and even if different pathnames are used to reference the  
 8520 file.

8521 The *mode* parameter describes how *dlopen()* shall operate upon *file* with respect to the processing  
 8522 of relocations and the scope of visibility of the symbols provided within *file*. When an object is  
 8523 brought into the address space of a process, it may contain references to symbols whose  
 8524 addresses are not known until the object is loaded. These references shall be relocated before the  
 8525 symbols can be accessed. The *mode* parameter governs when these relocations take place and  
 8526 may have the following values:

8527 RTLD\_LAZY Relocations shall be performed at an implementation-defined time,  
 8528 ranging from the time of the *dlopen()* call until the first reference to a  
 8529 given symbol occurs. Specifying RTLD\_LAZY should improve  
 8530 performance on implementations supporting dynamic symbol binding as  
 8531 a process may not reference all of the functions in any given object. And,  
 8532 for systems supporting dynamic symbol resolution for normal process  
 8533 execution, this behavior mimics the normal handling of process  
 8534 execution.

8535 RTLD\_NOW All necessary relocations shall be performed when the object is first  
 8536 loaded. This may waste some processing if relocations are performed for  
 8537 functions that are never referenced. This behavior may be useful for  
 8538 applications that need to know as soon as an object is loaded that all

8539 symbols referenced during execution are available.

8540 Any object loaded by *dlopen()* that requires relocations against global symbols can reference the  
8541 symbols in the original process image file, any objects loaded at program start-up, from the  
8542 object itself as well as any other object included in the same *dlopen()* invocation, and any objects  
8543 that were loaded in any *dlopen()* invocation and which specified the RTLD\_GLOBAL flag. To  
8544 determine the scope of visibility for the symbols loaded with a *dlopen()* invocation, the *mode*  
8545 parameter should be a bitwise-inclusive OR with one of the following values:

8546 RTLD\_GLOBAL The object's symbols shall be made available for the relocation processing  
8547 of any other object. In addition, symbol lookup using *dlopen(0, mode)* and  
8548 an associated *dlsym()* allows objects loaded with this *mode* to be searched.

8549 RTLD\_LOCAL The object's symbols shall not be made available for the relocation  
8550 processing of any other object.

8551 If neither RTLD\_GLOBAL nor RTLD\_LOCAL are specified, then an implementation-defined  
8552 default behavior shall be applied.

8553 If a *file* is specified in multiple *dlopen()* invocations, *mode* is interpreted at each invocation. Note,  
8554 however, that once RTLD\_NOW has been specified all relocations shall have been completed  
8555 rendering further RTLD\_NOW operations redundant and any further RTLD\_LAZY operations  
8556 irrelevant. Similarly, note that once RTLD\_GLOBAL has been specified the object shall maintain  
8557 the RTLD\_GLOBAL status regardless of any previous or future specification of RTLD\_LOCAL,  
8558 as long as the object remains in the address space (see *dlclose()*).

8559 Symbols introduced into a program through calls to *dlopen()* may be used in relocation  
8560 activities. Symbols so introduced may duplicate symbols already defined by the program or  
8561 previous *dlopen()* operations. To resolve the ambiguities such a situation might present, the  
8562 resolution of a symbol reference to symbol definition is based on a symbol resolution order. Two  
8563 such resolution orders are defined: *load* or *dependency* ordering. Load order establishes an  
8564 ordering among symbol definitions, such that the definition first loaded (including definitions  
8565 from the image file and any dependent objects loaded with it) has priority over objects added  
8566 later (via *dlopen()*). Load ordering is used in relocation processing. Dependency ordering uses a  
8567 breadth-first order starting with a given object, then all of its dependencies, then any dependents  
8568 of those, iterating until all dependencies are satisfied. With the exception of the global symbol  
8569 object obtained via a *dlopen()* operation on a *file* of 0, dependency ordering is used by the  
8570 *dlsym()* function. Load ordering is used in *dlsym()* operations upon the global symbol object.

8571 When an object is first made accessible via *dlopen()* it and its dependent objects are added in  
8572 dependency order. Once all the objects are added, relocations are performed using load order.  
8573 Note that if an object or its dependencies had been previously loaded, the load and dependency  
8574 orders may yield different resolutions.

8575 The symbols introduced by *dlopen()* operations, and available through *dlsym()* are at a  
8576 minimum those which are exported as symbols of global scope by the object. Typically such  
8577 symbols shall be those that were specified in (for example) C source code as having *extern*  
8578 linkage. The precise manner in which an implementation constructs the set of exported symbols  
8579 for a *dlopen()* object is specified by that implementation.

8580 **RETURN VALUE**

8581 If *file* cannot be found, cannot be opened for reading, is not of an appropriate object format for  
8582 processing by *dlopen()*, or if an error occurs during the process of loading *file* or relocating its  
8583 symbolic references, *dlopen()* shall return NULL. More detailed diagnostic information shall be  
8584 available through *dlerror()*.

8585 **ERRORS**

8586       No errors are defined.

8587 **EXAMPLES**

8588       None.

8589 **APPLICATION USAGE**

8590       None.

8591 **RATIONALE**

8592       None.

8593 **FUTURE DIRECTIONS**

8594       None.

8595 **SEE ALSO**

8596       *dlclose()*, *dLError()*, *dlsym()*, the Base Definitions volume of IEEE Std 1003.1-200x, <**dlfcn.h**>

8597 **CHANGE HISTORY**

8598       First released in Issue 5.

8599 **NAME**8600 dlsym — obtain the address of a symbol from a *dlopen()* object8601 **SYNOPSIS**8602 XSI `#include <dlfcn.h>`8603 `void *dlsym(void *restrict handle, const char *restrict name);`

8604

8605 **DESCRIPTION**

8606 The *dlsym()* function shall obtain the address of a symbol defined within an object made |  
 8607 accessible through a *dlopen()* call. The *handle* argument is the value returned from a call to |  
 8608 *dlopen()* (and which has not since been released via a call to *dlclose()*), and *name* is the symbol's |  
 8609 name as a character string.

8610 The *dlsym()* function shall search for the named symbol in all objects loaded automatically as a |  
 8611 result of loading the object referenced by *handle* (see *dlopen()*). Load ordering is used in *dlsym()* |  
 8612 operations upon the global symbol object. The symbol resolution algorithm used shall be |  
 8613 dependency order as described in *dlopen()*.

8614 The RTLD\_NEXT flag is reserved for future use.

8615 **RETURN VALUE**

8616 If *handle* does not refer to a valid object opened by *dlopen()*, or if the named symbol cannot be |  
 8617 found within any of the objects associated with *handle*, *dlsym()* shall return NULL. More |  
 8618 detailed diagnostic information shall be available through *dLError()*.

8619 **ERRORS**

8620 No errors are defined.

8621 **EXAMPLES**

8622 The following example shows how *dlopen()* and *dlsym()* can be used to access either function or |  
 8623 data objects. For simplicity, error checking has been omitted.

```
8624 void *handle;
8625 int *iptr, (*fptr)(int);

8626 /* open the needed object */
8627 handle = dlopen("/usr/home/me/libfoo.so", RTLD_LOCAL | RTLD_LAZY);

8628 /* find the address of function and data objects */
8629 fptr = (int (*)(int))dlsym(handle, "my_function");
8630 iptr = (int *)dlsym(handle, "my_object");

8631 /* invoke function, passing value of integer as a parameter */
8632 (*fptr)(*iptr);
```

8633 **APPLICATION USAGE**

8634 Special purpose values for *handle* are reserved for future use. These values and their meanings |  
 8635 are:

8636 RTLD\_DEFAULT The symbol lookup happens in the normal global scope; that is, a search for a |  
 8637 symbol using this handle would find the same definition as a direct use of this |  
 8638 symbol in the program code.

8639 RTLD\_NEXT Specifies the next object after this one that defines *name*. *This one* refers to the |  
 8640 object containing the invocation of *dlsym()*. The *next* object is the one found |  
 8641 upon the application of a load order symbol resolution algorithm (see |  
 8642 *dlopen()*). The next object is either one of global scope (because it was |  
 8643 introduced as part of the original process image or because it was added with

- 8644 a *dlopen()* operation including the `RTLD_GLOBAL` flag), or is an object that  
8645 was included in the same *dlopen()* operation that loaded this one.
- 8646 The `RTLD_NEXT` flag is useful to navigate an intentionally created hierarchy  
8647 of multiply-defined symbols created through *interposition*. For example, if a  
8648 program wished to create an implementation of *malloc()* that embedded some  
8649 statistics gathering about memory allocations, such an implementation could  
8650 use the real *malloc()* definition to perform the memory allocation—and itself  
8651 only embed the necessary logic to implement the statistics gathering function.
- 8652 **RATIONALE**  
8653 None.
- 8654 **FUTURE DIRECTIONS**  
8655 None.
- 8656 **SEE ALSO**  
8657 *dlclose()*, *dLError()*, *dlopen()*, the Base Definitions volume of IEEE Std 1003.1-200x, <**dlfcn.h**>
- 8658 **CHANGE HISTORY**  
8659 First released in Issue 5.
- 8660 **Issue 6**  
8661 The **restrict** keyword is added to the *dlsym()* prototype for alignment with the  
8662 ISO/IEC 9899:1999 standard.

## 8663 NAME

8664 drand48, erand48, jrand48, lcong48, lrand48, mrand48, nrand48, seed48, srand48 — generate  
8665 uniformly distributed pseudo-random numbers

## 8666 SYNOPSIS

```
8667 xSI #include <stdlib.h>
8668
8668 double drand48(void);
8669 double erand48(unsigned short xsubi[3]);
8670 long jrand48(unsigned short xsubi[3]);
8671 void lcong48(unsigned short param[7]);
8672 long lrand48(void);
8673 long mrand48(void);
8674 long nrand48(unsigned short xsubi[3]);
8675 unsigned short *seed48(unsigned short seed16v[3]);
8676 void srand48(long seedval);
8677
```

## 8678 DESCRIPTION

8679 This family of functions shall generate pseudo-random numbers using a linear congruential  
8680 algorithm and 48-bit integer arithmetic.

8681 The *drand48()* and *erand48()* functions shall return non-negative, double-precision, floating-  
8682 point values, uniformly distributed over the interval [0.0,1.0).

8683 The *lrnd48()* and *nrnd48()* functions shall return non-negative, long integers, uniformly  
8684 distributed over the interval  $[0, 2^{31})$ .

8685 The *mrnd48()* and *jrnd48()* functions shall return signed long integers uniformly distributed  
8686 over the interval  $[-2^{31}, 2^{31})$ .

8687 The *srand48()*, *seed48()*, and *lcong48()* are initialization entry points, one of which should be  
8688 invoked before either *drand48()*, *lrnd48()*, or *mrnd48()* is called. (Although it is not  
8689 recommended practice, constant default initializer values shall be supplied automatically if  
8690 *drand48()*, *lrnd48()*, or *mrnd48()* is called without a prior call to an initialization entry point.)  
8691 The *erand48()*, *nrnd48()*, and *jrnd48()* functions do not require an initialization entry point to  
8692 be called first.

8693 All the routines work by generating a sequence of 48-bit integer values,  $X_i$ , according to the  
8694 linear congruential formula:

$$8695 \quad X_{n+1} = (aX_n + c)_{\text{mod } m} \quad n \geq 0$$

8696 The parameter  $m = 2^{48}$ ; hence 48-bit integer arithmetic is performed. Unless *lcong48()* is invoked,  
8697 the multiplier value  $a$  and the addend value  $c$  are given by:

$$8698 \quad a = 5\text{DEECE66D}_{16} = 273673163155_8$$

$$8699 \quad c = \text{B}_{16} = 13_8$$

8700 The value returned by any of the *drand48()*, *erand48()*, *jrnd48()*, *lrnd48()*, *mrnd48()*, or  
8701 *nrnd48()* functions is computed by first generating the next 48-bit  $X_i$  in the sequence. Then the  
8702 appropriate number of bits, according to the type of data item to be returned, are copied from  
8703 the high-order (leftmost) bits of  $X_i$  and transformed into the returned value.

8704 The *drand48()*, *lrnd48()*, and *mrnd48()* functions store the last 48-bit  $X_i$  generated in an  
8705 internal buffer; that is why the application shall ensure that these are initialized prior to being  
8706 invoked. The *erand48()*, *nrnd48()*, and *jrnd48()* functions require the calling program to  
8707 provide storage for the successive  $X_i$  values in the array specified as an argument when the

8708 functions are invoked. That is why these routines do not have to be initialized; the calling  
8709 program merely has to place the desired initial value of  $X_i$  into the array and pass it as an  
8710 argument. By using different arguments, *erand48()*, *nrand48()*, and *jrand48()* allow separate  
8711 modules of a large program to generate several *independent* streams of pseudo-random numbers;  
8712 that is, the sequence of numbers in each stream shall *not* depend upon how many times the  
8713 routines are called to generate numbers for the other streams.

8714 The initializer function *srand48()* sets the high-order 32 bits of  $X_i$  to the low-order 32 bits  
8715 contained in its argument. The low-order 16 bits of  $X_i$  are set to the arbitrary value  $330E_{16}$ .

8716 The initializer function *seed48()* sets the value of  $X_i$  to the 48-bit value specified in the argument  
8717 array. The low-order 16 bits of  $X_i$  are set to the low-order 16 bits of *seed16v*[0]. The mid-order 16  
8718 bits of  $X_i$  are set to the low-order 16 bits of *seed16v*[1]. The high-order 16 bits of  $X_i$  are set to the  
8719 low-order 16 bits of *seed16v*[2]. In addition, the previous value of  $X_i$  is copied into a 48-bit  
8720 internal buffer, used only by *seed48()*, and a pointer to this buffer is the value returned by  
8721 *seed48()*. This returned pointer, which can just be ignored if not needed, is useful if a program is  
8722 to be restarted from a given point at some future time—use the pointer to get at and store the  
8723 last  $X_i$  value, and then use this value to reinitialize via *seed48()* when the program is restarted.

8724 The initializer function *lcg48()* allows the user to specify the initial  $X_i$ , the multiplier value *a*,  
8725 and the addend value *c*. Argument array elements *param*[0-2] specify  $X_i$ , *param*[3-5] specify the  
8726 multiplier *a*, and *param*[6] specifies the 16-bit addend *c*. After *lcg48()* is called, a subsequent  
8727 call to either *srand48()* or *seed48()* shall restore the standard multiplier and addend values, *a* and  
8728 *c*, specified above.

8729 The *drand48()*, *lrnd48()*, and *mrnd48()* functions need not be reentrant. A function that is not  
8730 required to be reentrant is not required to be thread-safe.

#### 8731 RETURN VALUE

8732 As described in the DESCRIPTION above.

#### 8733 ERRORS

8734 No errors are defined.

#### 8735 EXAMPLES

8736 None.

#### 8737 APPLICATION USAGE

8738 None.

#### 8739 RATIONALE

8740 None.

#### 8741 FUTURE DIRECTIONS

8742 None.

#### 8743 SEE ALSO

8744 *rand()*, the Base Definitions volume of IEEE Std 1003.1-200x, <stdlib.h>

#### 8745 CHANGE HISTORY

8746 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 8747 Issue 5

8748 A note indicating that these functions need not be reentrant is added to the DESCRIPTION.

#### 8749 Issue 6

8750 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

8751 **NAME**

8752 dup, dup2 — duplicate an open file descriptor

8753 **SYNOPSIS**

8754 #include &lt;unistd.h&gt;

8755 int dup(int *fil-des*);8756 int dup2(int *fil-des*, int *fil-des2*);8757 **DESCRIPTION**8758 The *dup()* and *dup2()* functions provide an alternative interface to the service provided by *fcntl()* using the `F_DUPFD` command. The call:8760 *fid* = dup(*fil-des*);

8761 shall be equivalent to:

8762 *fid* = fcntl(*fil-des*, `F_DUPFD`, 0);

8763 The call:

8764 *fid* = dup2(*fil-des*, *fil-des2*);

8765 shall be equivalent to:

8766 close(*fil-des2*);8767 *fid* = fcntl(*fil-des*, `F_DUPFD`, *fil-des2*);

8768 except for the following:

- 8769 • If *fil-des2* is less than 0 or greater than or equal to `{OPEN_MAX}`, *dup2()* shall return `-1` with *errno* set to `[EBADF]`.
- 8770
- 8771 • If *fil-des* is a valid file descriptor and is equal to *fil-des2*, *dup2()* shall return *fil-des2* without closing it.
- 8772
- 8773 • If *fil-des* is not a valid file descriptor, *dup2()* shall return `-1` and shall not close *fil-des2*.
- 8774 • The value returned shall be equal to the value of *fil-des2* upon successful completion, or `-1` upon failure.
- 8775

8776 **RETURN VALUE**8777 Upon successful completion a non-negative integer, namely the file descriptor, shall be returned; otherwise, `-1` shall be returned and *errno* set to indicate the error.8779 **ERRORS**8780 The *dup()* function shall fail if:8781 `[EBADF]` The *fil-des* argument is not a valid open file descriptor.8782 `[EMFILE]` The number of file descriptors in use by this process would exceed `{OPEN_MAX}`.8784 The *dup2()* function shall fail if:8785 `[EBADF]` The *fil-des* argument is not a valid open file descriptor or the argument *fil-des2* is negative or greater than or equal to `{OPEN_MAX}`.8787 `[EINTR]` The *dup2()* function was interrupted by a signal.

8788 **EXAMPLES**8789 **Redirecting Standard Output to a File**

8790 The following example closes standard output for the current processes, re-assigns standard  
8791 output to go to the file referenced by *pfid*, and closes the original file descriptor to clean up.

```
8792 #include <unistd.h>  
8793 ...  
8794 int pfd;  
8795 ...  
8796 close(1);  
8797 dup(pfd);  
8798 close(pfd);  
8799 ...
```

8800 **Redirecting Error Messages**

8801 The following example redirects messages from *stderr* to *stdout*.

```
8802 #include <unistd.h>  
8803 ...  
8804 dup2(1, 2);  
8805 ...
```

8806 **APPLICATION USAGE**

8807 None.

8808 **RATIONALE**

8809 The *dup()* and *dup2()* functions are redundant. Their services are also provided by the *fcntl()*  
8810 function. They have been included in this volume of IEEE Std 1003.1-200x primarily for historical  
8811 reasons, since many existing applications use them.

8812 While the brief code segment shown is very similar in behavior to *dup2()*, a conforming  
8813 implementation based on other functions defined in this volume of IEEE Std 1003.1-200x  
8814 is significantly more complex. Least obvious is the possible effect of a signal-catching function that  
8815 could be invoked between steps and allocate or deallocate file descriptors. This could be avoided  
8816 by blocking signals.

8817 The *dup2()* function is not marked obsolescent because it presents a type-safe version of  
8818 functionality provided in a type-unsafe version by *fcntl()*. It is used in the POSIX Ada binding.

8819 The *dup2()* function is not intended for use in critical regions as a synchronization mechanism.

8820 In the description of [EBADF], the case of *fildes* being out of range is covered by the given case of  
8821 *fildes* not being valid. The descriptions for *fildes* and *fildes2* are different because the only kind of  
8822 invalidity that is relevant for *fildes2* is whether it is out of range; that is, it does not matter  
8823 whether *fildes2* refers to an open file when the *dup2()* call is made.

8824 **FUTURE DIRECTIONS**

8825 None.

8826 **SEE ALSO**

8827 *close()*, *fcntl()*, *open()*, the Base Definitions volume of IEEE Std 1003.1-200x, <**unistd.h**>

8828 **CHANGE HISTORY**

8829 First released in Issue 1. Derived from Issue 1 of the SVID.

8830 **Issue 6**

8831 The RATIONALE section is added.

## 8832 NAME

8833 ecvt, fcvt, gcvt — convert a floating-point number to a string (**LEGACY**)

## 8834 SYNOPSIS

8835 xSI #include &lt;stdlib.h&gt;

8836 char \*ecvt(double value, int ndigit, int \*restrict decpt,  
8837 int \*restrict sign);8838 char \*fcvt(double value, int ndigit, int \*restrict decpt,  
8839 int \*restrict sign);

8840 char \*gcvt(double value, int ndigit, char \*buf);

8841

## 8842 DESCRIPTION

8843 The *ecvt()*, *fcvt()*, and *gcvt()* functions shall convert floating-point numbers to null-terminated  
8844 strings.

8845 The *ecvt()* function shall convert *value* to a null-terminated string of *ndigit* digits (where *ndigit* is  
8846 reduced to an unspecified limit determined by the precision of a **double**) and return a pointer to  
8847 the string. The high-order digit shall be non-zero, unless the value is 0. The low-order digit shall  
8848 be rounded in an implementation-defined manner. The position of the radix character relative to  
8849 the beginning of the string shall be stored in the integer pointed to by *decpt* (negative means to  
8850 the left of the returned digits). If *value* is zero, it is unspecified whether the integer pointed to by  
8851 *decpt* would be 0 or 1. The radix character shall not be included in the returned string. If the sign  
8852 of the result is negative, the integer pointed to by *sign* shall be non-zero; otherwise, it shall be 0.

8853 If the converted value is out of range or is not representable, the contents of the returned string  
8854 are unspecified.

8855 The *fcvt()* function shall be equivalent to *ecvt()*, except that *ndigit* specifies the number of digits  
8856 desired after the radix character. The total number of digits in the result string is restricted to an  
8857 unspecified limit as determined by the precision of a **double**.

8858 The *gcvt()* function shall convert *value* to a null-terminated string (similar to that of the %g  
8859 conversion specification format of *printf()*) in the array pointed to by *buf* and shall return *buf*. It  
8860 shall produce *ndigit* significant digits (limited to an unspecified value determined by the  
8861 precision of a **double**) in the %E conversion specification format of *printf()* if possible, or the %e  
8862 conversion specification format of *printf()* (scientific notation) otherwise. A minus sign shall be  
8863 included in the returned string if *value* is less than 0. A radix character shall be included in the  
8864 returned string if *value* is not a whole number. Trailing zeros shall be suppressed where *value* is  
8865 not a whole number. The radix character is determined by the current locale. If *setlocale()* has not  
8866 been called successfully, the default locale, POSIX, is used. The default locale specifies a period  
8867 ('.') as the radix character. The *LC\_NUMERIC* category determines the value of the radix  
8868 character within the current locale.

8869 These functions need not be reentrant. A function that is not required to be reentrant is not  
8870 required to be thread-safe.

## 8871 RETURN VALUE

8872 The *ecvt()* and *fcvt()* functions shall return a pointer to a null-terminated string of digits.8873 The *gcvt()* function shall return *buf*.

8874 The return values from *ecvt()* and *fcvt()* may point to static data which may be overwritten by  
8875 subsequent calls to these functions.

8876 **ERRORS**

8877 No errors are defined.

8878 **EXAMPLES**

8879 None.

8880 **APPLICATION USAGE**8881 *sprintf()* is preferred over this function.8882 **RATIONALE**

8883 None.

8884 **FUTURE DIRECTIONS**

8885 These functions may be withdrawn in a future version.

8886 **SEE ALSO**8887 *printf()*, *setlocale()*, the Base Definitions volume of IEEE Std 1003.1-200x, <stdlib.h>8888 **CHANGE HISTORY**

8889 First released in Issue 4, Version 2.

8890 **Issue 5**

8891 Moved from X/OPEN UNIX extension to BASE.

8892 Normative text previously in the APPLICATION USAGE section is moved to the  
8893 DESCRIPTION.

8894 A note indicating that these functions need not be reentrant is added to the DESCRIPTION.

8895 **Issue 6**

8896 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

8897 This function is marked LEGACY.

8898 The **restrict** keyword is added to the *ecvt()* and *fcvt()* prototypes for alignment with the  
8899 ISO/IEC 9899:1999 standard.8900 The DESCRIPTION is updated to explicitly use “conversion specification” to describe %g, %f,  
8901 and %e.

8902 **NAME**8903 encrypt — encoding function (**CRYPT**)8904 **SYNOPSIS**8905 XSI `#include <unistd.h>`8906 `void encrypt(char block[64], int edflag);`

8907

8908 **DESCRIPTION**8909 The *encrypt()* function shall provide access to an implementation-defined encoding algorithm. |8910 The key generated by *setkey()* is used to encrypt the string *block* with *encrypt()*. |8911 The *block* argument to *encrypt()* shall be an array of length 64 bytes containing only the bytes |

8912 with values of 0 and 1. The array is modified in place to a similar array using the key set by |

8913 *setkey()*. If *edflag* is 0, the argument is encoded. If *edflag* is 1, the argument may be decoded (see |8914 the APPLICATION USAGE section); if the argument is not decoded, *errno* shall be set to |

8915 [ENOSYS].

8916 The *encrypt()* function shall not change the setting of *errno* if successful. An application wishing |8917 to check for error situations should set *errno* to 0 before calling *encrypt()*. If *errno* is non-zero on |

8918 return, an error has occurred.

8919 The *encrypt()* function need not be reentrant. A function that is not required to be reentrant is |

8920 not required to be thread-safe.

8921 **RETURN VALUE**8922 The *encrypt()* function shall not return a value.8923 **ERRORS**8924 The *encrypt()* function shall fail if:

8925 [ENOSYS] The functionality is not supported on this implementation.

8926 **EXAMPLES**

8927 None.

8928 **APPLICATION USAGE**8929 Historical implementations of the *encrypt()* function used a rather primitive encoding algorithm. |

8930 In some environments, decoding might not be implemented. This is related to some Government |

8931 restrictions on encryption and decryption routines. Historical practice has been to ship a |

8932 different version of the encryption library without the decryption feature in the routines |

8933 supplied. Thus the exported version of *encrypt()* does encoding but not decoding.8934 **RATIONALE**

8935 None.

8936 **FUTURE DIRECTIONS**

8937 None.

8938 **SEE ALSO**8939 *crypt()*, *setkey()*, the Base Definitions volume of IEEE Std 1003.1-200x, <**unistd.h**>8940 **CHANGE HISTORY**

8941 First released in Issue 1. Derived from Issue 1 of the SVID.

8942 **Issue 5**

8943 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

8944 **Issue 6**

8945 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

8946 **NAME**

8947 endgrent, getgrent, setgrent — group database entry functions

8948 **SYNOPSIS**

```
8949 xSI #include <grp.h>
8950
8950 void endgrent(void);
8951 struct group *getgrent(void);
8952 void setgrent(void);
8953
```

8954 **DESCRIPTION**

8955 The *getgrent()* function shall return a pointer to a structure containing the broken-out fields of an  
 8956 entry in the group database. When first called, *getgrent()* shall return a pointer to a **group**  
 8957 structure containing the first entry in the group database. Thereafter, it shall return a pointer to a  
 8958 **group** structure containing the next group structure in the group database, so successive calls  
 8959 may be used to search the entire database.

8960 An implementation that provides extended security controls may impose further  
 8961 implementation-defined restrictions on accessing the group database. In particular, the system  
 8962 may deny the existence of some or all of the group database entries associated with groups other  
 8963 than those groups associated with the caller and may omit users other than the caller from the  
 8964 list of members of groups in database entries that are returned.

8965 The *setgrent()* function shall rewind the group database to allow repeated searches.

8966 The *endgrent()* function may be called to close the group database when processing is complete.

8967 These functions need not be reentrant. A function that is not required to be reentrant is not  
 8968 required to be thread-safe.

8969 **RETURN VALUE**

8970 When first called, *getgrent()* shall return a pointer to the first group structure in the group  
 8971 database. Upon subsequent calls it shall return the next group structure in the group database.  
 8972 The *getgrent()* function shall return a null pointer on end-of-file or an error and *errno* may be set  
 8973 to indicate the error.

8974 The return value may point to a static area which is overwritten by a subsequent call to  
 8975 *getgrgid()*, *getgrnam()*, or *getgrent()*.

8976 **ERRORS**

8977 The *getgrent()* function may fail if:

- |      |          |  |
|------|----------|--|
| 8978 | [EINTR]  | A signal was caught during the operation.                              |
| 8979 | [EIO]    | An I/O error has occurred.   |
| 8980 | [EMFILE] | {OPEN_MAX} file descriptors are currently open in the calling process. |
| 8981 | [ENFILE] | The maximum allowable number of files is currently open in the system. |

8982 **EXAMPLES**

8983 None.

8984 **APPLICATION USAGE**

8985 These functions are provided due to their historical usage. Applications should avoid  
8986 dependencies on fields in the group database, whether the database is a single file, or where in  
8987 the file system name space the database resides. Applications should use *getgrnam()* and  
8988 *getgrgid()* whenever possible because it avoids these dependencies.

8989 **RATIONALE**

8990 None.

8991 **FUTURE DIRECTIONS**

8992 None.

8993 **SEE ALSO**

8994 *getgrgid()*, *getgrnam()*, *getlogin()*, *getpwent()*, the Base Definitions volume of  
8995 IEEE Std 1003.1-200x, <grp.h>

8996 **CHANGE HISTORY**

8997 First released in Issue 4, Version 2.

8998 **Issue 5**

8999 Moved from X/OPEN UNIX extension to BASE.

9000 Normative text previously in the APPLICATION USAGE section is moved to the RETURN  
9001 VALUE section.

9002 A note indicating that these functions need not be reentrant is added to the DESCRIPTION.

9003 **Issue 6**

9004 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

9005 **NAME**

9006 endhostent, gethostent, sethostent — network host database functions

9007 **SYNOPSIS**

9008 #include &lt;netdb.h&gt;

9009 void endhostent(void);

9010 struct hostent \*gethostent(void);

9011 void sethostent(int *stayopen*);9012 **DESCRIPTION**

9013 These functions shall retrieve information about hosts. This information is considered to be |  
9014 stored in a database that can be accessed sequentially or randomly. The implementation of this |  
9015 database is unspecified. |

9016 **Note:** In many cases it is implemented by the Domain Name System, as documented in RFC 1034,  
9017 RFC 1035, and RFC 1886.

9018 The *sethostent()* function shall open a connection to the database and set the next entry for  
9019 retrieval to the first entry in the database. If the *stayopen* argument is non-zero, the connection  
9020 shall not be closed by a call to *gethostent()*, *gethostbyname()*, or *gethostbyaddr()*, and the  
9021 implementation may maintain an open file descriptor.

9022 The *gethostent()* function shall read the next entry in the database, opening and closing a  
9023 connection to the database as necessary.

9024 Entries shall be returned in **hostent** structures. Refer to *gethostbyaddr()* for a definition of the |  
9025 **hostent** structure.

9026 The *endhostent()* function shall close the connection to the database, releasing any open file  
9027 descriptor.

9028 These functions need not be reentrant. A function that is not required to be reentrant is not  
9029 required to be thread-safe.

9030 **RETURN VALUE**

9031 Upon successful completion, the *gethostent()* function shall return a pointer to a **hostent**  
9032 structure if the requested entry was found, and a null pointer if the end of the database was  
9033 reached or the requested entry was not found.

9034 **ERRORS**9035 No errors are defined for *endhostent()*, *gethostent()*, and *sethostent()*.9036 **EXAMPLES**

9037 None.

9038 **APPLICATION USAGE**

9039 The *gethostent()* function may return pointers to static data, which may be overwritten by  
9040 subsequent calls to any of these functions.

9041 **RATIONALE**

9042 None.

9043 **FUTURE DIRECTIONS**

9044 None.

9045 **SEE ALSO**

9046 *endservent()*, *gethostbyaddr()*, *gethostbyname()*, the Base Definitions volume of  
9047 IEEE Std 1003.1-200x, <netdb.h>

9048 **CHANGE HISTORY**

9049 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

9050 **NAME**

9051 endnetent, getnetbyaddr, getnetbyname, getnetent, setnetent — network database functions

9052 **SYNOPSIS**

9053 #include &lt;netdb.h&gt;

9054 void endnetent(void);

9055 struct netent \*getnetbyaddr(uint32\_t net, int type);

9056 struct netent \*getnetbyname(const char \*name);

9057 struct netent \*getnetent(void);

9058 void setnetent(int stayopen);

9059 **DESCRIPTION**

9060 These functions shall retrieve information about networks. This information is considered to be |  
9061 stored in a database that can be accessed sequentially or randomly. The implementation of this |  
9062 database is unspecified. |

9063 The *setnetent()* function shall open and rewind the database. If the *stayopen* argument is non-  
9064 zero, the connection to the *net* database shall not be closed after each call to *getnetent()* (either  
9065 directly, or indirectly through one of the other *getnet\**(*)* functions), and the implementation may  
9066 maintain an open file descriptor to the database.

9067 The *getnetent()* function shall read the next entry of the database, opening and closing a  
9068 connection to the database as necessary.

9069 The *getnetbyaddr()* function shall search the database from the beginning, and find the first entry  
9070 for which the address family specified by *type* matches the *n\_addrtype* member and the network  
9071 number *net* matches the *n\_net* member, opening and closing a connection to the database as  
9072 necessary. The *net* argument shall be the network number in host byte order.

9073 The *getnetbyname()* function shall search the database from the beginning and find the first entry  
9074 for which the network name specified by *name* matches the *n\_name* member, opening and  
9075 closing a connection to the database as necessary.

9076 The *getnetbyaddr()*, *getnetbyname()*, and *getnetent()*, functions shall each return a pointer to a  
9077 **netent** structure, the members of which shall contain the fields of an entry in the network  
9078 database.

9079 The *endnetent()* function shall close the database, releasing any open file descriptor.

9080 These functions need not be reentrant. A function that is not required to be reentrant is not  
9081 required to be thread-safe.

9082 **RETURN VALUE**

9083 Upon successful completion, *getnetbyaddr()*, *getnetbyname()*, and *getnetent()*, shall return a  
9084 pointer to a **netent** structure if the requested entry was found, and a null pointer if the end of the  
9085 database was reached or the requested entry was not found. Otherwise, a null pointer shall be  
9086 returned.

9087 **ERRORS**

9088 No errors are defined.

9089 **EXAMPLES**

9090       None.

9091 **APPLICATION USAGE**9092       The *getnetbyaddr()*, *getnetbyname()*, and *getnetent()*, functions may return pointers to static data,  
9093       which may be overwritten by subsequent calls to any of these functions.9094 **RATIONALE**

9095       None.

9096 **FUTURE DIRECTIONS**

9097       None.

9098 **SEE ALSO**9099       The Base Definitions volume of IEEE Std 1003.1-200x, <**netdb.h**>9100 **CHANGE HISTORY**

9101       First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

9102 **NAME**

9103 endprotoent, getprotobyname, getprotobynumber, getprotoent, setprotoent — network protocol  
9104 database functions

9105 **SYNOPSIS**

```
9106     #include <netdb.h>

9107     void endprotoent(void);
9108     struct protoent *getprotobyname(const char *name);
9109     struct protoent *getprotobynumber(int proto);
9110     struct protoent *getprotoent(void);
9111     void setprotoent(int stayopen);
```

9112 **DESCRIPTION**

9113 These functions shall retrieve information about protocols. This information is considered to be |  
9114 stored in a database that can be accessed sequentially or randomly. The implementation of this |  
9115 database is unspecified. |

9116 The *setprotoent()* function shall open a connection to the database, and set the next entry to the  
9117 first entry. If the *stayopen* argument is non-zero, the connection to the network protocol database  
9118 shall not be closed after each call to *getprotoent()* (either directly, or indirectly through one of the  
9119 other *getproto\**() functions), and the implementation may maintain an open file descriptor for  
9120 the database.

9121 The *getprotobyname()* function shall search the database from the beginning and find the first  
9122 entry for which the protocol name specified by *name* matches the *p\_name* member, opening and  
9123 closing a connection to the database as necessary.

9124 The *getprotobynumber()* function shall search the database from the beginning and find the first  
9125 entry for which the protocol number specified by *proto* matches the *p\_proto* member, opening  
9126 and closing a connection to the database as necessary.

9127 The *getprotoent()* function shall read the next entry of the database, opening and closing a  
9128 connection to the database as necessary.

9129 The *getprotobyname()*, *getprotobynumber()*, and *getprotoent()*, functions shall each return a pointer  
9130 to a **protoent** structure, the members of which shall contain the fields of an entry in the network  
9131 protocol database.

9132 The *endprotoent()* function shall close the connection to the database, releasing any open file  
9133 descriptor.

9134 These functions need not be reentrant. A function that is not required to be reentrant is not  
9135 required to be thread-safe.

9136 **RETURN VALUE**

9137 Upon successful completion, *getprotobyname()*, *getprotobynumber()*, and *getprotoent()* return a  
9138 pointer to a **protoent** structure if the requested entry was found, and a null pointer if the end of  
9139 the database was reached or the requested entry was not found. Otherwise, a null pointer is  
9140 returned.

9141 **ERRORS**

9142 No errors are defined.

9143 **EXAMPLES**

9144       None.

9145 **APPLICATION USAGE**9146       The *getprotobyname()*, *getprotobynumber()*, and *getprotoent()* functions may return pointers to  
9147       static data, which may be overwritten by subsequent calls to any of these functions.9148 **RATIONALE**

9149       None.

9150 **FUTURE DIRECTIONS**

9151       None.

9152 **SEE ALSO**9153       The Base Definitions volume of IEEE Std 1003.1-200x, <**netdb.h**>9154 **CHANGE HISTORY**

9155       First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

9156 **NAME**  
 9157 endpwent, getpwent, setpwent — user database functions

9158 **SYNOPSIS**

```
9159 xSI #include <pwd.h>
9160 void endpwent(void);
9161 struct passwd *getpwent(void);
9162 void setpwent(void);
9163
```

9164 **DESCRIPTION**

9165 These functions shall retrieve information about users. |

9166 The *getpwent()* function shall return a pointer to a structure containing the broken-out fields of |  
 9167 an entry in the user database. Each entry in the user database contains a **passwd** structure. When |  
 9168 first called, *getpwent()* shall return a pointer to a **passwd** structure containing the first entry in |  
 9169 the user database. Thereafter, it shall return a pointer to a **passwd** structure containing the next |  
 9170 entry in the user database. Successive calls can be used to search the entire user database.

9171 If an end-of-file or an error is encountered on reading, *getpwent()* shall return a null pointer.

9172 An implementation that provides extended security controls may impose further  
 9173 implementation-defined restrictions on accessing the user database. In particular, the system  
 9174 may deny the existence of some or all of the user database entries associated with users other  
 9175 than the caller.

9176 The *setpwent()* function effectively rewinds the user database to allow repeated searches.

9177 The *endpwent()* function may be called to close the user database when processing is complete.

9178 These functions need not be reentrant. A function that is not required to be reentrant is not  
 9179 required to be thread-safe.

9180 **RETURN VALUE**

9181 The *getpwent()* function shall return a null pointer on end-of-file or error.

9182 **ERRORS**

9183 The *getpwent()*, *setpwent()*, and *endpwent()* functions may fail if:

9184 [EIO] An I/O error has occurred.

9185 In addition, *getpwent()* and *setpwent()* may fail if:

9186 [EMFILE] {OPEN\_MAX} file descriptors are currently open in the calling process.

9187 [ENFILE] The maximum allowable number of files is currently open in the system.

9188 The return value may point to a static area which is overwritten by a subsequent call to  
 9189 *getpwuid()*, *getpwnam()*, or *getpwent()*. |

9190 **EXAMPLES**9191 **Searching the User Database**

9192 The following example uses the *getpwent()* function to get successive entries in the user  
 9193 database, returning a pointer to a **passwd** structure that contains information about each user.  
 9194 The call to *endpwent()* closes the user database and cleans up.

```
9195 #include <pwd.h>
9196 ...
9197 struct passwd *p;
9198 ...
9199 while ((p = getpwent ()) != NULL) {
9200     ...
9201 }
9202 endpwent();
9203 ...
```

9204 **APPLICATION USAGE**

9205 These functions are provided due to their historical usage. Applications should avoid  
 9206 dependencies on fields in the password database, whether the database is a single file, or where  
 9207 in the file system name space the database resides. Applications should use *getpwuid()*  
 9208 whenever possible because it avoids these dependencies.

9209 **RATIONALE**

9210 None.

9211 **FUTURE DIRECTIONS**

9212 None.

9213 **SEE ALSO**

9214 *endgrent()*, *getlogin()*, *getpwnam()*, *getpwuid()*, the Base Definitions volume of  
 9215 IEEE Std 1003.1-200x, **<pwd.h>**

9216 **CHANGE HISTORY**

9217 First released in Issue 4, Version 2.

9218 **Issue 5**

9219 Moved from X/OPEN UNIX extension to BASE.

9220 Normative text previously in the APPLICATION USAGE section is moved to the RETURN  
 9221 VALUE section.

9222 A note indicating that these functions need not be reentrant is added to the DESCRIPTION.

9223 **Issue 6**

9224 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

9225 **NAME**

9226 endservent, getservbyname, getservbyport, getservent, setservent — network services database  
9227 functions

9228 **SYNOPSIS**

```
9229 #include <netdb.h>

9230 void endservent(void);
9231 struct servent *getservbyname(const char *name, const char *proto);
9232 struct servent *getservbyport(int port, const char *proto);
9233 struct servent *getservent(void);
9234 void setservent(int stayopen);
```

9235 **DESCRIPTION**

9236 These functions shall retrieve information about network services. This information is |  
9237 considered to be stored in a database that can be accessed sequentially or randomly. The |  
9238 implementation of this database is unspecified. |

9239 The *setservent()* function shall open a connection to the database, and set the next entry to the  
9240 first entry. If the *stayopen* argument is non-zero, the *net* database shall not be closed after each  
9241 call to the *getservent()* function (either directly, or indirectly through one of the other *getserv\*()*  
9242 functions), and the implementation may maintain an open file descriptor for the database.

9243 The *getservent()* function shall read the next entry of the database, opening and closing a  
9244 connection to the database as necessary.

9245 The *getservbyname()* function shall search the database from the beginning and find the first  
9246 entry for which the service name specified by *name* matches the *s\_name* member and the protocol  
9247 name specified by *proto* matches the *s\_proto* member, opening and closing a connection to the  
9248 database as necessary. If *proto* is a null pointer, any value of the *s\_proto* member shall be  
9249 matched.

9250 The *getservbyport()* function shall search the database from the beginning and find the first entry  
9251 for which the port specified by *port* matches the *s\_port* member and the protocol name specified  
9252 by *proto* matches the *s\_proto* member, opening and closing a connection to the database as  
9253 necessary. If *proto* is a null pointer, any value of the *s\_proto* member shall be matched. The *port*  
9254 argument shall be in network byte order.

9255 The *getservbyname()*, *getservbyport()*, and *getservent()* functions shall each return a pointer to a  
9256 **servent** structure, the members of which shall contain the fields of an entry in the network  
9257 services database.

9258 The *endservent()* function shall close the database, releasing any open file descriptor.

9259 These functions need not be reentrant. A function that is not required to be reentrant is not  
9260 required to be thread-safe.

9261 **RETURN VALUE**

9262 Upon successful completion, *getservbyname()*, *getservbyport()*, and *getservent()* return a pointer to  
9263 a **servent** structure if the requested entry was found, and a null pointer if the end of the database  
9264 was reached or the requested entry was not found. Otherwise, a null pointer is returned.

9265 **ERRORS**

9266 No errors are defined.

9267 **EXAMPLES**

9268 None.

9269 **APPLICATION USAGE**9270 The *port* argument of *getservbyport()* need not be compatible with the port values of all address  
9271 families.9272 The *getservbyname()*, *getservbyport()*, and *getservent()* functions may return pointers to static  
9273 data, which may be overwritten by subsequent calls to any of these functions.9274 **RATIONALE**

9275 None.

9276 **FUTURE DIRECTIONS**

9277 None.

9278 **SEE ALSO**9279 *endhostent()*, *endprotoent()*, *htonl()*, *inet\_addr()*, the Base Definitions volume of  
9280 IEEE Std 1003.1-200x, <**netdb.h**>9281 **CHANGE HISTORY**

9282 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

## 9283 NAME

9284 endutxent, getutxent, getutxid, getutxline, pututxline, setutxent — user accounting database  
 9285 functions

## 9286 SYNOPSIS

```
9287 xSI #include <utmpx.h>
9288
9288 void endutxent(void);
9289 struct utmpx *getutxent(void);
9290 struct utmpx *getutxid(const struct utmpx *id);
9291 struct utmpx *getutxline(const struct utmpx *line);
9292 struct utmpx *pututxline(const struct utmpx *utmpx);
9293 void setutxent(void);
9294
```

## 9295 DESCRIPTION

9296 These functions shall provide access to the user accounting database. |

9297 The *getutxent()* function shall read the next entry from the user accounting database. If the |  
 9298 database is not already open, it shall open it. If it reaches the end of the database, it shall fail. |

9299 The *getutxid()* function shall search forward from the current point in the database. If the |  
 9300 *ut\_type* value of the **utmpx** structure pointed to by *id* is *BOOT\_TIME*, *OLD\_TIME*, or |  
 9301 *NEW\_TIME*, then it shall stop when it finds an entry with a matching *ut\_type* value. If the |  
 9302 *ut\_type* value is *INIT\_PROCESS*, *LOGIN\_PROCESS*, *USER\_PROCESS*, or *DEAD\_PROCESS*, |  
 9303 then it shall stop when it finds an entry whose type is one of these four and whose *ut\_id* member |  
 9304 matches the *ut\_id* member of the **utmpx** structure pointed to by *id*. If the end of the database is |  
 9305 reached without a match, *getutxid()* shall fail. |

9306 The *getutxline()* function shall search forward from the current point in the database until it |  
 9307 finds an entry of the type *LOGIN\_PROCESS* or *USER\_PROCESS* which also has a *ut\_line* value |  
 9308 matching that in the **utmpx** structure pointed to by *line*. If the end of the database is reached |  
 9309 without a match, *getutxline()* shall fail. |

9310 The *getutxid()* or *getutxline()* function may cache data. For this reason, to use *getutxline()* to |  
 9311 search for multiple occurrences, the application shall zero out the static data after each success, |  
 9312 or *getutxline()* may return a pointer to the same **utmpx** structure. |

9313 There is one exception to the rule about clearing the structure before further reads are done. The |  
 9314 implicit read done by *pututxline()* (if it finds that it is not already at the correct place in the user |  
 9315 accounting database) shall not modify the static structure returned by *getutxent()*, *getutxid()*, or |  
 9316 *getutxline()*, if the application has modified this structure and passed the pointer back to |  
 9317 *pututxline()*.

9318 For all entries that match a request, the *ut\_type* member indicates the type of the entry. Other |  
 9319 members of the entry shall contain meaningful data based on the value of the *ut\_type* member as |  
 9320 follows:

9321  
9322  
9323  
9324  
9325  
9326  
9327  
9328  
9329  
9330  
9331

<b>ut_type Member</b>	<b>Other Members with Meaningful Data</b>
EMPTY	No others
BOOT_TIME	<i>ut_tv</i>
OLD_TIME	<i>ut_tv</i>
NEW_TIME	<i>ut_tv</i>
USER_PROCESS	<i>ut_id</i> , <i>ut_user</i> (login name of the user), <i>ut_line</i> , <i>ut_pid</i> , <i>ut_tv</i>
INIT_PROCESS	<i>ut_id</i> , <i>ut_pid</i> , <i>ut_tv</i>
LOGIN_PROCESS	<i>ut_id</i> , <i>ut_user</i> (implementation-defined name of the login process), <i>ut_pid</i> , <i>ut_tv</i>
DEAD_PROCESS	<i>ut_id</i> , <i>ut_pid</i> , <i>ut_tv</i>

9332 An implementation that provides extended security controls may impose implementation-  
9333 defined restrictions on accessing the user accounting database. In particular, the system may  
9334 deny the existence of some or all of the user accounting database entries associated with users  
9335 other than the caller.

9336 If the process has appropriate privileges, the *pututxline()* function shall write out the structure  
9337 into the user accounting database. It shall use *getutxid()* to search for a record that satisfies the  
9338 request. If this search succeeds, then the entry shall be replaced. Otherwise, a new entry shall be  
9339 made at the end of the user accounting database.

9340 The *endutxent()* function shall close the user accounting database.

9341 The *setutxent()* function shall reset the input to the beginning of the database. This should be  
9342 done before each search for a new entry if it is desired that the entire database be examined.

9343 These functions need not be reentrant. A function that is not required to be reentrant is not  
9344 required to be thread-safe.

#### 9345 RETURN VALUE

9346 Upon successful completion, *getutxent()*, *getutxid()*, and *getutxline()* shall return a pointer to a  
9347 **utmpx** structure containing a copy of the requested entry in the user accounting database.  
9348 Otherwise, a null pointer shall be returned.

9349 The return value may point to a static area which is overwritten by a subsequent call to  
9350 *getutxid()* or *getutxline()*.

9351 Upon successful completion, *pututxline()* shall return a pointer to a **utmpx** structure containing a  
9352 copy of the entry added to the user accounting database. Otherwise, a null pointer shall be  
9353 returned.

9354 The *endutxent()* and *setutxent()* functions shall not return a value.

#### 9355 ERRORS

9356 No errors are defined for the *endutxent()*, *getutxent()*, *getutxid()*, *getutxline()*, and *setutxent()*  
9357 functions.

9358 The *pututxline()* function may fail if:

9359 [EPERM] The process does not have appropriate privileges.

9360 **EXAMPLES**

9361 None.

9362 **APPLICATION USAGE**9363 The sizes of the arrays in the structure can be found using the *sizeof* operator.9364 **RATIONALE**

9365 None.

9366 **FUTURE DIRECTIONS**

9367 None.

9368 **SEE ALSO**

9369 The Base Definitions volume of IEEE Std 1003.1-200x, &lt;utmpx.h&gt;

9370 **CHANGE HISTORY**

9371 First released in Issue 4, Version 2.

9372 **Issue 5**

9373 Moved from X/OPEN UNIX extension to BASE.

9374 Normative text previously in the APPLICATION USAGE section is moved to the  
9375 DESCRIPTION.

9376 A note indicating that these functions need not be reentrant is added to the DESCRIPTION.

9377 **Issue 6**

9378 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

9379 **NAME**

9380        **environ** — array of character pointers to the environment strings

9381 **SYNOPSIS**

9382        extern char \*\*environ;

9383 **DESCRIPTION**

9384        Refer to the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 8, Environment Variables  
9385        and *exec*.

9386 **NAME**9387        **erand48** — generate uniformly distributed pseudo-random numbers9388 **SYNOPSIS**9389 **XSI**        #include <stdlib.h>9390        double erand48(unsigned short *xsubi*[3]);

9391

9392 **DESCRIPTION**9393        Refer to *drand48()*.

9394 **NAME**

9395 erf, erff, erfl — error functions

9396 **SYNOPSIS**

9397 #include &lt;math.h&gt;

9398 double erf(double x);

9399 float erff(float x);

9400 long double erfl(long double x);

9401 **DESCRIPTION**

9402 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 9403 conflict between the requirements described here and the ISO C standard is unintentional. This  
 9404 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

9405 These functions shall compute the error function of their argument *x*, defined as:

$$9406 \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$$

9407 An application wishing to check for error situations should set *errno* to zero and call  
 9408 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 9409 *fetetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 9410 zero, an error has occurred.

9411 **RETURN VALUE**

9412 Upon successful completion, these functions shall return the value of the error function.

9413 **MX** If *x* is NaN, a NaN shall be returned.9414 If *x* is ±0, ±0 shall be returned.9415 If *x* is ±Inf, ±1 shall be returned.9416 If *x* is subnormal, a range error may occur, and  $2 * x / \text{sqrt}(\pi)$  should be returned.9417 **ERRORS**

9418 These functions may fail if:

9419 **MX** **Range Error** The result underflows.

9420 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero,  
 9421 then *errno* shall be set to [ERANGE]. If the integer expression  
 9422 (math\_errhandling & MATH\_ERREXCEPT) is non-zero, then the underflow  
 9423 floating-point exception shall be raised.

9424 **EXAMPLES**

9425 None.

9426 **APPLICATION USAGE**9427 Underflow occurs when  $|x| < \text{DBL\_MIN} * (\text{sqrt}(\pi)/2)$ .

9428 On error, the expressions (math\_errhandling & MATH\_ERRNO) and (math\_errhandling &  
 9429 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

9430 **RATIONALE**

9431 None.

9432 **FUTURE DIRECTIONS**

9433 None.

9434 **SEE ALSO**9435 *erfc()*, *feclearexcept()*, *fetestexcept()*, *isnan()*, the Base Definitions volume of IEEE Std 1003.1-200x, |  
9436 Section 4.18, Treatment of Error Conditions for Mathematical Functions, <math.h> |9437 **CHANGE HISTORY**

9438 First released in Issue 1. Derived from Issue 1 of the SVID.

9439 **Issue 5**9440 The DESCRIPTION is updated to indicate how an application should check for an error. This  
9441 text was previously published in the APPLICATION USAGE section.9442 **Issue 6**9443 The *erf()* function is no longer marked as an extension.9444 The *erfc()* function is now split out onto its own reference page.9445 The *erff()* and *erfl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.9446 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
9447 revised to align with the ISO/IEC 9899:1999 standard.9448 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
9449 marked.

9450 **NAME**

9451           erfc, erfcf, erfcl — complementary error functions

9452 **SYNOPSIS**

9453           #include &lt;math.h&gt;

9454           double erfc(double x);

9455           float erfcf(float x);

9456           long double erfcl(long double x);

9457 **DESCRIPTION**

9458 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
 9459 conflict between the requirements described here and the ISO C standard is unintentional. This  
 9460 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

9461           These functions shall compute the complementary error function  $1.0 - \operatorname{erf}(x)$ .

9462           An application wishing to check for error situations should set *errno* to zero and call  
 9463 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 9464 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 9465 zero, an error has occurred.

9466 **RETURN VALUE**9467           Upon successful completion, these functions shall return the value of the complementary error  
9468 function.9469           If the correct value would cause underflow and is not representable, a range error may occur  
9470 **MX**       and either 0.0 (if representable), or an implementation-defined value shall be returned.9471 **MX**       If *x* is NaN, a NaN shall be returned.9472           If *x* is  $\pm 0$ , +1 shall be returned.9473           If *x* is  $-\operatorname{Inf}$ , +2 shall be returned.9474           If *x* is  $+\operatorname{Inf}$ , +0 shall be returned.9475           If the correct value would cause underflow and is representable, a range error may occur and the  
9476 correct value shall be returned.9477 **ERRORS**

9478           These functions may fail if:

9479           Range Error       The result underflows.

9480           If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero, |  
 9481 then *errno* shall be set to [ERANGE]. If the integer expression |  
 9482 (math\_errhandling & MATH\_ERREXCEPT) is non-zero, then the underflow |  
 9483 floating-point exception shall be raised. |

9484 **EXAMPLES**

9485           None.

9486 **APPLICATION USAGE**9487           The *erfc()* function is provided because of the extreme loss of relative accuracy if *erf(x)* is called  
9488 for large *x* and the result subtracted from 1.0.9489           Note for IEEE Std 754-1985 **double**,  $26.55 < x$  implies *erfc(x)* has underflowed.9490           On error, the expressions (math\_errhandling & MATH\_ERRNO) and (math\_errhandling &  
9491 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

9492 **RATIONALE**

9493       None.

9494 **FUTURE DIRECTIONS**

9495       None.

9496 **SEE ALSO**9497       *erf()*, *feclearexcept()*, *fetetestexcept()*, *isnan()*, the Base Definitions volume of IEEE Std 1003.1-200x, |  
9498       Section 4.18, Treatment of Error Conditions for Mathematical Functions, <math.h> |9499 **CHANGE HISTORY**

9500       First released in Issue 1. Derived from Issue 1 of the SVID.

9501 **Issue 5**9502       The DESCRIPTION is updated to indicate how an application should check for an error. This  
9503       text was previously published in the APPLICATION USAGE section.9504 **Issue 6**9505       The *erfc()* function is no longer marked as an extension.9506       These functions are split out from the *erf()* reference page.9507       The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
9508       revised to align with the ISO/IEC 9899:1999 standard.9509       IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
9510       marked.

9511 **NAME**

9512       erff, erfl — error functions

9513 **SYNOPSIS**

9514       #include &lt;math.h&gt;

9515       float erff(float x);

9516       long double erfl(long double x);

9517 **DESCRIPTION**9518       Refer to *erf()*.

9519 **NAME**

9520            errno — error return value

9521 **SYNOPSIS**

9522            #include &lt;errno.h&gt;

9523 **DESCRIPTION**9524            The lvalue *errno* is used by many functions to return error values. |

9525            Many functions provide an error number in *errno*. It has type **int** and is defined in <**errno.h**>. |  
9526            The value of *errno* shall be defined only after a call to a function for which it is explicitly stated to |  
9527            be set and until it is changed by the next function call or if the application assigns it a value. The |  
9528            value of *errno* should only be examined when it is indicated to be valid by a function's return |  
9529            value. Applications shall obtain the definition of *errno* by the inclusion of <**errno.h**>. No |  
9530            function in this volume of IEEE Std 1003.1-200x shall set *errno* to 0.

9531            It is unspecified whether *errno* is a macro or an identifier declared with external linkage. If a |  
9532            macro definition is suppressed in order to access an actual object, or a program defines an |  
9533            identifier with the name *errno*, the behavior is undefined.

9534            The symbolic values stored in *errno* are documented in the ERRORS sections on all relevant |  
9535            pages.

9536 **RETURN VALUE**

9537            None.

9538 **ERRORS**

9539            None.

9540 **EXAMPLES**

9541            None.

9542 **APPLICATION USAGE**

9543            Previously both POSIX and X/Open documents were more restrictive than the ISO C standard |  
9544            in that they required *errno* to be defined as an external variable, whereas the ISO C standard |  
9545            required only that *errno* be defined as a modifiable lvalue with type **int**. |

9546            A program that uses *errno* for error checking should set it to 0 before a function call, then inspect |  
9547            it before a subsequent function call.

9548 **RATIONALE**

9549            None.

9550 **FUTURE DIRECTIONS**

9551            None.

9552 **SEE ALSO**9553            Section 2.3, the Base Definitions volume of IEEE Std 1003.1-200x, <**errno.h**>9554 **CHANGE HISTORY**

9555            First released in Issue 1. Derived from Issue 1 of the SVID.

9556 **Issue 5**

9557            The following sentence is deleted from the DESCRIPTION: “The value of *errno* is 0 at program |  
9558            start-up, but is never set to 0 by any XSI function”. The DESCRIPTION also no longer states that |  
9559            conforming implementations may support the declaration:

9560            extern int errno;

9561 **Issue 6**

9562           Obsolescent text regarding defining *errno* as:

9563           extern int errno

9564           is removed.

9565           Text regarding no function setting *errno* to zero to indicate an error is changed to no function  
9566           shall set *errno* to zero. This is for alignment with the ISO/IEC 9899:1999 standard.

9567 **NAME**

9568 environ, execl, execv, execl, execve, execlp, execvp — execute a file

9569 **SYNOPSIS**

```

9570 #include <unistd.h>

9571 extern char **environ;
9572 int execl(const char *path, const char *arg0, ... /*, (char *)0 */);
9573 int execv(const char *path, char *const argv[]);
9574 int execl(const char *path, const char *arg0, ... /*,
9575           (char *)0, char *const envp[] */);
9576 int execve(const char *path, char *const argv[], char *const envp[]);
9577 int execlp(const char *file, const char *arg0, ... /*, (char *)0 */);
9578 int execvp(const char *file, char *const argv[]);

```

9579 **DESCRIPTION**

9580 The *exec* family of functions shall replace the current process image with a new process image. |  
 9581 The new image shall be constructed from a regular, executable file called the *new process image* |  
 9582 *file*. There shall be no return from a successful *exec*, because the calling process image is overlaid |  
 9583 by the new process image.

9584 When a C-language program is executed as a result of this call, it shall be entered as a C-  
 9585 language function call as follows:

```
9586 int main (int argc, char *argv[]);
```

9587 where *argc* is the argument count and *argv* is an array of character pointers to the arguments  
 9588 themselves. In addition, the following variable:

```
9589 extern char **environ;
```

9590 is initialized as a pointer to an array of character pointers to the environment strings. The *argv*  
 9591 and *environ* arrays are each terminated by a null pointer. The null pointer terminating the *argv*  
 9592 array is not counted in *argc*.

9593 **THR** Conforming multi-threaded applications shall not use the *environ* variable to access or modify  
 9594 any environment variable while any other thread is concurrently modifying any environment  
 9595 variable. A call to any function dependent on any environment variable shall be considered a use  
 9596 of the *environ* variable to access that environment variable.

9597 The arguments specified by a program with one of the *exec* functions shall be passed on to the  
 9598 new process image in the corresponding *main()* arguments.

9599 The argument *path* points to a pathname that identifies the new process image file. |

9600 The argument *file* is used to construct a pathname that identifies the new process image file. If |  
 9601 the *file* argument contains a slash character, the *file* argument shall be used as the pathname for |  
 9602 this file. Otherwise, the path prefix for this file is obtained by a search of the directories passed |  
 9603 as the environment variable *PATH* (see the Base Definitions volume of IEEE Std 1003.1-200x,  
 9604 Chapter 8, Environment Variables). If this environment variable is not present, the results of the  
 9605 search are implementation-defined.

9606 If the process image file is not a valid executable object, and the system does not recognize it as  
 9607 something that cannot be executed (and thus returns [EINVAL]), *execlp()* and *execvp()* shall use  
 9608 the contents of that file as standard input to a command interpreter conforming to *system()*. In  
 9609 this case, the command interpreter becomes the new process image.

9610 The arguments represented by *arg0*,... are pointers to null-terminated character strings. These  
 9611 strings shall constitute the argument list available to the new process image. The list is |

9612 terminated by a null pointer. The argument *arg0* should point to a filename that is associated |  
 9613 with the process being started by one of the *exec* functions.

9614 The argument *argv* is an array of character pointers to null-terminated strings. The application |  
 9615 shall ensure that the last member of this array is a null pointer. These strings shall constitute the |  
 9616 argument list available to the new process image. The value in *argv*[0] should point to a filename |  
 9617 that is associated with the process being started by one of the *exec* functions.

9618 The argument *envp* is an array of character pointers to null-terminated strings. These strings |  
 9619 shall constitute the environment for the new process image. The *envp* array is terminated by a |  
 9620 null pointer.

9621 For those forms not containing an *envp* pointer (*execl*(), *execv*(), *execlp*(), and *execvp*()), the |  
 9622 environment for the new process image shall be taken from the external variable *environ* in the |  
 9623 calling process.

9624 The number of bytes available for the new process' combined argument and environment lists is |  
 9625 {ARG\_MAX}. It is implementation-defined whether null terminators, pointers, and/or any |  
 9626 alignment bytes are included in this total.

9627 File descriptors open in the calling process image shall remain open in the new process image, |  
 9628 except for those whose close-on-exec flag FD\_CLOEXEC is set. For those file descriptors that |  
 9629 remain open, all attributes of the open file description remain unchanged. For any file descriptor |  
 9630 that is closed for this reason, file locks are removed as a result of the close as described in *close*(). |  
 9631 Locks that are not removed by closing of file descriptors remain unchanged.

9632 Directory streams open in the calling process image shall be closed in the new process image.

9633 The state of the floating-point environment in the new process image shall be set to the default. |

9634 XSI The state of conversion descriptors and message catalog descriptors in the new process image is |  
 9635 undefined. For the new process image, the equivalent of:

```
9636 setlocale(LC_ALL, "C")
```

9637 shall be executed at start-up. |

9638 Signals set to the default action (SIG\_DFL) in the calling process image shall be set to the default |  
 9639 action in the new process image. Except for SIGCHLD, signals set to be ignored (SIG\_IGN) by |  
 9640 the calling process image shall be set to be ignored by the new process image. Signals set to be |  
 9641 caught by the calling process image shall be set to the default action in the new process image |  
 9642 (see <signal.h>). If the SIGCHLD signal is set to be ignored by the calling process image, it is |  
 9643 unspecified whether the SIGCHLD signal is set to be ignored or to the default action in the new |  
 9644 XSI process image. After a successful call to any of the *exec* functions, alternate signal stacks are not |  
 9645 preserved and the SA\_ONSTACK flag shall be cleared for all signals.

9646 After a successful call to any of the *exec* functions, any functions previously registered by *atexit*() |  
 9647 are no longer registered.

9648 XSI If the ST\_NOSUID bit is set for the file system containing the new process image file, then the |  
 9649 effective user ID, effective group ID, saved set-user-ID, and saved set-group-ID are unchanged |  
 9650 in the new process image. Otherwise, if the set-user-ID mode bit of the new process image file is |  
 9651 set, the effective user ID of the new process image shall be set to the user ID of the new process |  
 9652 image file. Similarly, if the set-group-ID mode bit of the new process image file is set, the |  
 9653 effective group ID of the new process image shall be set to the group ID of the new process |  
 9654 image file. The real user ID, real group ID, and supplementary group IDs of the new process |  
 9655 image shall remain the same as those of the calling process image. The effective user ID and |  
 9656 effective group ID of the new process image shall be saved (as the saved set-user-ID and the |  
 9657 saved set-group-ID) for use by *setuid*().

9658	XSI	Any shared memory segments attached to the calling process image shall not be attached to the new process image.
9659		
9660	SEM	Any named semaphores open in the calling process shall be closed as if by appropriate calls to <i>sem_close()</i> .
9661		
9662	TYM	Any blocks of typed memory that were mapped in the calling process are unmapped, as if <i>munmap()</i> was implicitly called to unmap them.
9663		
9664	ML	Memory locks established by the calling process via calls to <i>mlockall()</i> or <i>mlock()</i> shall be removed. If locked pages in the address space of the calling process are also mapped into the address spaces of other processes and are locked by those processes, the locks established by the other processes shall be unaffected by the call by this process to the <i>exec</i> function. If the <i>exec</i> function fails, the effect on memory locks is unspecified.
9665		
9666		
9667		
9668		
9669	MF SHM	Memory mappings created in the process are unmapped before the address space is rebuilt for the new process image.
9670		
9671	PS	For the SCHED_FIFO and SCHED_RR scheduling policies, the policy and priority settings shall not be changed by a call to an <i>exec</i> function. For other scheduling policies, the policy and priority settings on <i>exec</i> are implementation-defined.
9672		
9673		
9674	TMR	Per-process timers created by the calling process shall be deleted before replacing the current process image with the new process image.
9675		
9676	MSG	All open message queue descriptors in the calling process shall be closed, as described in <i>mq_close()</i> .
9677		
9678	AIO	Any outstanding asynchronous I/O operations may be canceled. Those asynchronous I/O operations that are not canceled shall complete as if the <i>exec</i> function had not yet occurred, but any associated signal notifications shall be suppressed. It is unspecified whether the <i>exec</i> function itself blocks awaiting such I/O completion. In no event, however, shall the new process image created by the <i>exec</i> function be affected by the presence of outstanding asynchronous I/O operations at the time the <i>exec</i> function is called. Whether any I/O is canceled, and which I/O may be canceled upon <i>exec</i> , is implementation-defined.
9679		
9680		
9681		
9682		
9683		
9684		
9685	CPT	The new process image shall inherit the CPU-time clock of the calling process image. This inheritance means that the process CPU-time clock of the process being <i>execed</i> shall not be reinitialized or altered as a result of the <i>exec</i> function other than to reflect the time spent by the process executing the <i>exec</i> function itself.
9686		
9687		
9688		
9689	TCT	The initial value of the CPU-time clock of the initial thread of the new process image shall be set to zero.
9690		
9691	TRC	If the calling process is being traced, the new process image shall continue to be traced into the same trace stream as the original process image, but the new process image shall not inherit the mapping of trace event names to trace event type identifiers that was defined by calls to the <i>posix_trace_eventid_open()</i> or the <i>posix_trace_trid_eventid_open()</i> functions in the calling process image.
9692		
9693		
9694		
9695		
9696		If the calling process is a trace controller process, any trace streams that were created by the calling process shall be shut down as described in the <i>posix_trace_shutdown()</i> function.
9697		
9698		The new process shall inherit at least the following attributes from the calling process image:
9699	XSI	• Nice value (see <i>nice()</i> )
9700	XSI	• <i>semadj</i> values (see <i>semop()</i> )

- 9701           • Process ID
  - 9702           • Parent process ID
  - 9703           • Process group ID
  - 9704           • Session membership
  - 9705           • Real user ID
  - 9706           • Real group ID
  - 9707           • Supplementary group IDs
  - 9708           • Time left until an alarm clock signal (see *alarm()*)
  - 9709           • Current working directory
  - 9710           • Root directory
  - 9711           • File mode creation mask (see *umask()*)
  - 9712 XSI       • File size limit (see *ulimit()*)
  - 9713           • Process signal mask (see *sigprocmask()*)
  - 9714           • Pending signal (see *sigpending()*)
  - 9715           • *tms\_utime*, *tms\_stime*, *tms\_cutime*, and *tms\_cstime* (see *times()*)
  - 9716 XSI       • Resource limits
  - 9717 XSI       • Controlling terminal
  - 9718 XSI       • Interval timers
- 9719           All other process attributes defined in this volume of IEEE Std 1003.1-200x shall be the same in  
 9720           the new and old process images. The inheritance of process attributes not defined by this  
 9721           volume of IEEE Std 1003.1-200x is implementation-defined.
- 9722           A call to any *exec* function from a process with more than one thread shall result in all threads  
 9723           being terminated and the new executable image being loaded and executed. No destructor  
 9724           functions shall be called.
- 9725           Upon successful completion, the *exec* functions shall mark for update the *st\_atime* field of the file.  
 9726           If an *exec* function failed but was able to locate the *process image file*, whether the *st\_atime* field is  
 9727           marked for update is unspecified. Should the *exec* function succeed, the process image file shall  
 9728           be considered to have been opened with *open()*. The corresponding *close()* shall be considered  
 9729           to occur at a time after this open, but before process termination or successful completion of a  
 9730           subsequent call to one of the *exec* functions, *posix\_spawn()*, or *posix\_spawnp()*. The *argv[]* and  
 9731           *envp[]* arrays of pointers and the strings to which those arrays point shall not be modified by a  
 9732           call to one of the *exec* functions, except as a consequence of replacing the process image.
- 9733 XSI       The saved resource limits in the new process image are set to be a copy of the process'  
 9734           corresponding hard and soft limits.
- 9735   **RETURN VALUE**
- 9736           If one of the *exec* functions returns to the calling process image, an error has occurred; the return  
 9737           value shall be  $-1$ , and *errno* shall be set to indicate the error.

9738 **ERRORS**

- 9739 The *exec* functions shall fail if:
- 9740 [E2BIG] The number of bytes used by the new process image's argument list and  
9741 environment list is greater than the system-imposed limit of {ARG\_MAX}  
9742 bytes.
- 9743 [EACCES] Search permission is denied for a directory listed in the new process image  
9744 file's path prefix, or the new process image file denies execution permission,  
9745 or the new process image file is not a regular file and the implementation does  
9746 not support execution of files of its type.
- 9747 [EINVAL] The new process image file has the appropriate permission and has a  
9748 recognized executable binary format, but the system does not support  
9749 execution of a file with this format.
- 9750 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path* or *file*  
9751 argument.
- 9752 [ENAMETOOLONG]  
9753 The length of the *path* or *file* arguments exceeds {PATH\_MAX} or a pathname |  
9754 component is longer than {NAME\_MAX}. |
- 9755 [ENOENT] A component of *path* or *file* does not name an existing file or *path* or *file* is an  
9756 empty string.
- 9757 [ENOTDIR] A component of the new process image file's path prefix is not a directory.
- 9758 The *exec* functions, except for *execlp()* and *execvp()*, shall fail if:
- 9759 [ENOEXEC] The new process image file has the appropriate access permission but has an  
9760 unrecognized format.
- 9761 The *exec* functions may fail if:
- 9762 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
9763 resolution of the *path* or *file* argument.
- 9764 [ENAMETOOLONG]  
9765 As a result of encountering a symbolic link in resolution of the *path* argument, |  
9766 the length of the substituted pathname string exceeded {PATH\_MAX}. |
- 9767 [ENOMEM] The new process image requires more memory than is allowed by the  
9768 hardware or system-imposed memory management constraints.
- 9769 [ETXTBSY] The new process image file is a pure procedure (shared text) file that is  
9770 currently open for writing by some process.

9771 **EXAMPLES**9772 **Using *execl()***

9773 The following example executes the *ls* command, specifying the pathname of the executable |  
9774 (*/bin/ls*) and using arguments supplied directly to the command to produce single-column  
9775 output.

```
9776 #include <unistd.h>
9777 int ret;
9778 ...
9779 ret = execl ("/bin/ls", "ls", "-l", (char *)0);
```

**9780 Using execl()**

9781 The following example is similar to **Using execl()** (on page 758). In addition, it specifies the  
9782 environment for the new process image using the *env* argument.

```
9783 #include <unistd.h>
9784 int ret;
9785 char *env[] = { "HOME=/usr/home", "LOGNAME=home", (char *)0 };
9786 ...
9787 ret = execl ("/bin/ls", "ls", "-l", (char *)0, env);
```

**9788 Using execlp()**

9789 The following example searches for the location of the *ls* command among the directories  
9790 specified by the *PATH* environment variable.

```
9791 #include <unistd.h>
9792 int ret;
9793 ...
9794 ret = execlp ("ls", "ls", "-l", (char *)0);
```

**9795 Using execv()**

9796 The following example passes arguments to the *ls* command in the *cmd* array.

```
9797 #include <unistd.h>
9798 int ret;
9799 char *cmd[] = { "ls", "-l", (char *)0 };
9800 ...
9801 ret = execv ("/bin/ls", cmd);
```

**9802 Using execve()**

9803 The following example passes arguments to the *ls* command in the *cmd* array, and specifies the  
9804 environment for the new process image using the *env* argument.

```
9805 #include <unistd.h>
9806 int ret;
9807 char *cmd[] = { "ls", "-l", (char *)0 };
9808 char *env[] = { "HOME=/usr/home", "LOGNAME=home", (char *)0 };
9809 ...
9810 ret = execve ("/bin/ls", cmd, env);
```

**9811 Using execvp()**

9812 The following example searches for the location of the *ls* command among the directories  
9813 specified by the *PATH* environment variable, and passes arguments to the *ls* command in the  
9814 *cmd* array.

```
9815 #include <unistd.h>
9816 int ret;
9817 char *cmd[] = { "ls", "-l", (char *)0 };
9818 ...
```

9819           ret = execvp ("ls", cmd);

#### 9820 APPLICATION USAGE

9821           As the state of conversion descriptors and message catalog descriptors in the new process image  
9822           is undefined, conforming applications should not rely on their use and should close them prior  
9823           to calling one of the *exec* functions.

9824           Applications that require other than the default POSIX locale should call *setlocale()* with the  
9825           appropriate parameters to establish the locale of the new process.

9826           The *environ* array should not be accessed directly by the application.

#### 9827 RATIONALE

9828           Early proposals required that the value of *argc* passed to *main()* be “one or greater”. This was  
9829           driven by the same requirement in drafts of the ISO C standard. In fact, historical  
9830           implementations have passed a value of zero when no arguments are supplied to the caller of  
9831           the *exec* functions. This requirement was removed from the ISO C standard and subsequently  
9832           removed from this volume of IEEE Std 1003.1-200x as well. The wording, in particular the use of  
9833           the word *should*, requires a Strictly Conforming POSIX Application to pass at least one argument  
9834           to the *exec* function, thus guaranteeing that *argc* be one or greater when invoked by such an  
9835           application. In fact, this is good practice, since many existing applications reference *argv[0]*  
9836           without first checking the value of *argc*.

9837           The requirement on a Strictly Conforming POSIX Application also states that the value passed  
9838           as the first argument be a filename associated with the process being started. Although some  
9839           existing applications pass a pathname rather than a filename in some circumstances, a filename  
9840           is more generally useful, since the common usage of *argv[0]* is in printing diagnostics. In some  
9841           cases the filename passed is not the actual filename of the file; for example, many  
9842           implementations of the *login* utility use a convention of prefixing a hyphen (‘-’) to the actual  
9843           filename, which indicates to the command interpreter being invoked that it is a “login shell”.

9844           Some implementations can *exec* shell scripts.

9845           One common historical implementation is that the *execl()*, *execv()*, *execle()*, and *execve()*  
9846           functions return an [ENOEXEC] error for any file not recognizable as executable, including a  
9847           shell script. When the *execlp()* and *execvp()* functions encounter such a file, they assume the file  
9848           to be a shell script and invoke a known command interpreter to interpret such files. These  
9849           implementations of *execvp()* and *execlp()* only give the [ENOEXEC] error in the rare case of a  
9850           problem with the command interpreter’s executable file. Because of these implementations, the  
9851           [ENOEXEC] error is not mentioned for *execlp()* or *execvp()*, although implementations can still  
9852           give it.

9853           Another way that some historical implementations handle shell scripts is by recognizing the first  
9854           two bytes of the file as the character string “#!” and using the remainder of the first line of the  
9855           file as the name of the command interpreter to execute.

9856           Some implementations provide a third argument to *main()* called *envp*. This is defined as a  
9857           pointer to the environment. The ISO C standard specifies invoking *main()* with two arguments,  
9858           so implementations must support applications written this way. Since this volume of  
9859           IEEE Std 1003.1-200x defines the global variable *environ*, which is also provided by historical  
9860           implementations and can be used anywhere that *envp* could be used, there is no functional need  
9861           for the *envp* argument. Applications should use the *getenv()* function rather than accessing the  
9862           environment directly via either *envp* or *environ*. Implementations are required to support the  
9863           two-argument calling sequence, but this does not prohibit an implementation from supporting  
9864           *envp* as an optional third argument.

9865 This volume of IEEE Std 1003.1-200x specifies that signals set to SIG\_IGN remain set to  
 9866 SIG\_IGN, and that the process signal mask be unchanged across an *exec*. This is consistent with  
 9867 historical implementations, and it permits some useful functionality, such as the *nohup*  
 9868 command. However, it should be noted that many existing applications wrongly assume that  
 9869 they start with certain signals set to the default action and/or unblocked. In particular,  
 9870 applications written with a simpler signal model that does not include blocking of signals, such  
 9871 as the one in the ISO C standard, may not behave properly if invoked with some signals blocked.  
 9872 Therefore, it is best not to block or ignore signals across *execs* without explicit reason to do so,  
 9873 and especially not to block signals across *execs* of arbitrary (not closely co-operating) programs.

9874 The *exec* functions always save the value of the effective user ID and effective group ID of the  
 9875 process at the completion of the *exec*, whether or not the set-user-ID or the set-group-ID bit of  
 9876 the process image file is set.

9877 The statement about *argv[]* and *envp[]* being constants is included to make explicit to future  
 9878 writers of language bindings that these objects are completely constant. Due to a limitation of  
 9879 the ISO C standard, it is not possible to state that idea in standard C. Specifying two levels of  
 9880 *const-qualification* for the *argv[]* and *envp[]* parameters for the *exec* functions may seem to be the  
 9881 natural choice, given that these functions do not modify either the array of pointers or the  
 9882 characters to which the function points, but this would disallow existing correct code. Instead,  
 9883 only the array of pointers is noted as constant. The table of assignment compatibility for *dst=src*,  
 9884 derived from the ISO C standard summarizes the compatibility:

9885	<i>dst:</i>	char *[]	const char *[]	char *const[]	const char *const[]
9886	<i>src:</i>				
9887	char *[]	VALID	—	VALID	—
9888	const char *[]	—	VALID	—	VALID
9889	char * const []	—	—	VALID	—
9890	const char *const[]	—	—	—	VALID

9891 Since all existing code has a source type matching the first row, the column that gives the most  
 9892 valid combinations is the third column. The only other possibility is the fourth column, but  
 9893 using it would require a cast on the *argv* or *envp* arguments. It is unfortunate that the fourth  
 9894 column cannot be used, because the declaration a non-expert would naturally use would be that  
 9895 in the second row.

9896 The ISO C standard and this volume of IEEE Std 1003.1-200x do not conflict on the use of  
 9897 *environ*, but some historical implementations of *environ* may cause a conflict. As long as *environ*  
 9898 is treated in the same way as an entry point (for example, *fork()*), it conforms to both standards.  
 9899 A library can contain *fork()*, but if there is a user-provided *fork()*, that *fork()* is given precedence  
 9900 and no problem ensues. The situation is similar for *environ*: the definition in this volume of  
 9901 IEEE Std 1003.1-200x is to be used if there is no user-provided *environ* to take precedence. At  
 9902 least three implementations are known to exist that solve this problem.

9903 [E2BIG] The limit {ARG\_MAX} applies not just to the size of the argument list, but to  
 9904 the sum of that and the size of the environment list.

9905 [EFAULT] Some historical systems return [EFAULT] rather than [ENOEXEC] when the  
 9906 new process image file is corrupted. They are non-conforming.

9907 [EINVAL] This error condition was added to IEEE Std 1003.1-200x to allow an  
 9908 implementation to detect executable files generated for different architectures,  
 9909 and indicate this situation to the application. Historical implementations of  
 9910 shells, *execvp()*, and *execlp()* that encounter an [ENOEXEC] error will execute  
 9911 a shell on the assumption that the file is a shell script. This will not produce  
 9912 the desired effect when the file is a valid executable for a different

9913 architecture. An implementation may now choose to avoid this problem by  
 9914 returning [EINVAL] when a valid executable for a different architecture is  
 9915 encountered. Some historical implementations return [EINVAL] to indicate  
 9916 that the *path* argument contains a character with the high order bit set. The  
 9917 standard developers chose to deviate from historical practice for the following  
 9918 reasons:

- 9919 1. The new utilization of [EINVAL] will provide some measure of utility to  
 9920 the user community.
- 9921 2. Historical use of [EINVAL] is not acceptable in an internationalized  
 9922 operating environment.

9923 [ENAMETOOLONG]

9924 Since the file pathname may be constructed by taking elements in the *PATH*  
 9925 variable and putting them together with the filename, the  
 9926 [ENAMETOOLONG] error condition could also be reached this way.

9927 [ETXTBSY]

9928 System V returns this error when the executable file is currently open for  
 9929 writing by some process. This volume of IEEE Std 1003.1-200x neither requires  
 nor prohibits this behavior.

9930 Other systems (such as System V) may return [EINTR] from *exec*. This is not addressed by this  
 9931 volume of IEEE Std 1003.1-200x, but implementations may have a window between the call to  
 9932 *exec* and the time that a signal could cause one of the *exec* calls to return with [EINTR].

9933 An explicit statement regarding the floating-point environment (as defined in the `<fenv.h>`  
 9934 header) was added to make it clear that the floating-point environment is set to its default when  
 9935 a call to one of the *exec* functions succeeds. The requirements for inheritance or setting to the  
 9936 default for other process and thread start-up functions is covered by more generic statements in  
 9937 their descriptions and can be summarized as follows:

9938 <i>posix_spawn()</i>	Set to default.
9939 <i>fork()</i>	Inherit.
9940 <i>pthread_create()</i>	Inherit.

#### 9941 FUTURE DIRECTIONS

9942 None.

#### 9943 SEE ALSO

9944 *alarm()*, *atexit()*, *chmod()*, *close()*, *exit()*, *fcntl()*, *fork()*, *fstatvfs()*, *getenv()*, *getitimer()*, *getrlimit()*,  
 9945 *mmap()*, *nice()*, *posix\_spawn()*, *posix\_trace\_eventid\_open()*, *posix\_trace\_shutdown()*,  
 9946 *posix\_trace\_trid\_eventid\_open()*, *putenv()*, *semop()*, *setlocale()*, *shmat()*, *sigaction()*, *sigaltstack()*,  
 9947 *sigpending()*, *sigprocmask()*, *system()*, *times()*, *ulimit()*, *umask()*, the Base Definitions volume of  
 9948 IEEE Std 1003.1-200x, `<unistd.h>`, the Base Definitions volume of IEEE Std 1003.1-200x, Chapter  
 9949 11, General Terminal Interface

#### 9950 CHANGE HISTORY

9951 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 9952 Issue 5

9953 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX  
 9954 Threads Extension.

9955 Large File Summit extensions are added.

9956 **Issue 6**

9957 The following new requirements on POSIX implementations derive from alignment with the |  
9958 Single UNIX Specification:

9959 • In the DESCRIPTION, behavior is defined for when the process image file is not a valid |  
9960 executable.

9961 • In this issue, `_POSIX_SAVED_IDS` is mandated, thus the effective user ID and effective group |  
9962 ID of the new process image shall be saved (as the saved set-user-ID and the saved set- |  
9963 group-ID) for use by the `setuid()` function.

9964 • The [ELOOP] mandatory error condition is added.

9965 • A second [ENAMETOOLONG] is added as an optional error condition.

9966 • The [ETXTBSY] optional error condition is added.

9967 The following changes were made to align with the IEEE P1003.1a draft standard:

9968 • The [EINVAL] mandatory error condition is added.

9969 • The [ELOOP] optional error condition is added.

9970 The description of CPU-time clock semantics is added for alignment with IEEE Std 1003.1d-1999.

9971 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by adding semantics for |  
9972 typed memory.

9973 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

9974 The description of tracing semantics is added for alignment with IEEE Std 1003.1q-2000. |

9975 IEEE PASC Interpretation 1003.1 #132 is applied. |

9976 The DESCRIPTION is updated to make it explicit that the floating-point environment in the new |  
9977 process image is set to the default. |

## 9978 NAME

9979 exit, \_Exit, \_exit — terminate a process

## 9980 SYNOPSIS

9981 #include &lt;stdlib.h&gt;

9982 void exit(int status);

9983 void \_Exit(int status);

9984 #include &lt;unistd.h&gt;

9985 void \_exit(int status);

## 9986 DESCRIPTION

9987 CX The functionality described on this reference page for the *exit()* and *\_Exit()* functions is aligned  
 9988 with the ISO C standard. Any conflict between the requirements described here and the ISO C  
 9989 standard are unintentional. This volume of IEEE Std 1003.1-200x defers to the ISO C standard.

9990 CX The value of *status* may be 0, EXIT\_SUCCESS, EXIT\_FAILURE, or any other value, though only  
 9991 the least significant 8 bits (that is, *status* & 0377) shall be available to a waiting parent process.

9992 The *exit()* function shall first call all functions registered by *atexit()*, in the reverse order of their  
 9993 registration, except that a function is called after any previously registered functions that had  
 9994 already been called at the time it was registered. Each function is called as many times as it was  
 9995 registered. If, during the call to any such function, a call to the *longjmp()* function is made that  
 9996 would terminate the call to the registered function, the behavior is undefined.

9997 If a function registered by a call to *atexit()* fails to return, the remaining registered functions shall  
 9998 not be called and the rest of the *exit()* processing shall not be completed. If *exit()* is called more  
 9999 than once, the behavior is undefined.

10000 The *exit()* function shall then flush all open streams with unwritten buffered data, close all open  
 10001 streams, and remove all files created by *tmpfile()*. Finally, control shall be terminated with the  
 10002 consequences described below.

10003 CX The *\_Exit()* and *\_exit()* functions shall be functionally equivalent.

10004 CX The *\_Exit()* and *\_exit()* functions shall not call functions registered with *atexit()* nor any  
 10005 registered signal handlers. Whether open streams are flushed or closed, or temporary files are  
 10006 removed is implementation-defined. Finally, the calling process is terminated with the  
 10007 consequences described below.

10008 CX These functions shall terminate the calling process with the following consequences:

10009 **Note:** These consequences are all extensions to the ISO C standard and are not further CX shaded.  
 10010 However, XSI extensions are shaded.

10011 XSI • All of the file descriptors, directory streams, conversion descriptors, and message catalog  
 10012 descriptors open in the calling process shall be closed.

10013 XSI • If the parent process of the calling process is executing a *wait()* or *waitpid()*, and has neither  
 10014 set its SA\_NOCLDWAIT flag nor set SIGCHLD to SIG\_IGN, it shall be notified of the calling  
 10015 process' termination and the low-order eight bits (that is, bits 0377) of *status* are made  
 10016 available to it. If the parent is not waiting, the child's status shall be made available to it  
 10017 when the parent subsequently executes *wait()* or *waitpid()*.

10018 XSI The semantics of the *waitid()* function shall be equivalent to *wait()*.

10019 XSI • If the parent process of the calling process is not executing a *wait()* or *waitpid()*, and has  
 10020 neither set its SA\_NOCLDWAIT flag nor set SIGCHLD to SIG\_IGN, the calling process shall  
 10021 be transformed into a *zombie process*. A *zombie process* is an inactive process and it shall be

10022		deleted at some later time when its parent process executes <i>wait()</i> or <i>waitpid()</i> .	
10023	XSI	The semantics of the <i>waitid()</i> function shall be equivalent to <i>wait()</i> .	
10024		• Termination of a process does not directly terminate its children. The sending of a SIGHUP signal as described below indirectly terminates children in some circumstances.	
10025			
10026		• Either:	
10027		If the implementation supports the SIGCHLD signal, a SIGCHLD shall be sent to the parent process.	
10028			
10029		Or:	
10030	XSI	If the parent process has set its SA_NOCLDWAIT flag, or set SIGCHLD to SIG_IGN, the status shall be discarded, and the lifetime of the calling process shall end immediately. If SA_NOCLDWAIT is set, it is implementation-defined whether a SIGCHLD signal is sent to the parent process.	
10031			
10032			
10033			
10034		• The parent process ID of all of the calling process' existing child processes and zombie processes shall be set to the process ID of an implementation-defined system process. That is, these processes shall be inherited by a special system process.	
10035			
10036			
10037	XSI	• Each attached shared-memory segment is detached and the value of <i>shm_nattch</i> (see <i>shmget()</i> ) in the data structure associated with its shared memory ID shall be decremented by 1.	
10038			
10039			
10040	XSI	• For each semaphore for which the calling process has set a <i>semadj</i> value (see <i>semop()</i> ), that value shall be added to the <i>semval</i> of the specified semaphore.	
10041			
10042		• If the process is a controlling process, the SIGHUP signal shall be sent to each process in the foreground process group of the controlling terminal belonging to the calling process.	
10043			
10044		• If the process is a controlling process, the controlling terminal associated with the session shall be disassociated from the session, allowing it to be acquired by a new controlling process.	
10045			
10046			
10047		• If the exit of the process causes a process group to become orphaned, and if any member of the newly-orphaned process group is stopped, then a SIGHUP signal followed by a SIGCONT signal shall be sent to each process in the newly-orphaned process group.	
10048			
10049			
10050	SEM	• All open named semaphores in the calling process shall be closed as if by appropriate calls to <i>sem_close()</i> .	
10051			
10052	ML	• Any memory locks established by the process via calls to <i>mlockall()</i> or <i>mlock()</i> shall be removed. If locked pages in the address space of the calling process are also mapped into the address spaces of other processes and are locked by those processes, the locks established by the other processes shall be unaffected by the call by this process to <i>_Exit()</i> or <i>_exit()</i> .	
10053			
10054			
10055			
10056	MF SHM	• Memory mappings created in the process shall be unmapped before the process is destroyed.	
10057			
10058	TYM	• Any blocks of typed memory that were mapped in the calling process shall be unmapped, as if <i>munmap()</i> was implicitly called to unmap them.	
10059			
10060	MSG	• All open message queue descriptors in the calling process shall be closed as if by appropriate calls to <i>mq_close()</i> .	
10061			
10062	AIO	• Any outstanding cancelable asynchronous I/O operations may be canceled. Those asynchronous I/O operations that are not canceled shall complete as if the <i>_Exit()</i> or <i>_exit()</i> operation had not yet occurred, but any associated signal notifications shall be suppressed.	
10063			
10064			

10065 The `_Exit()` or `_exit()` operation may block awaiting such I/O completion. Whether any I/O  
 10066 is canceled, and which I/O may be canceled upon `_Exit()` or `_exit()`, is implementation-  
 10067 defined.

10068 • Threads terminated by a call to `_Exit()` or `_exit()` shall not invoke their cancelation cleanup  
 10069 handlers or per-thread data destructors.

10070 TRC • If the calling process is a trace controller process, any trace streams that were created by the  
 10071 calling process shall be shut down as described by the `posix_trace_shutdown()` function, and  
 10072 any process' mapping of trace event names to trace event type identifiers built for these trace  
 10073 streams may be deallocated.

#### 10074 RETURN VALUE

10075 These functions do not return.

#### 10076 ERRORS

10077 No errors are defined.

#### 10078 EXAMPLES

10079 None.

#### 10080 APPLICATION USAGE

10081 Normally applications should use `exit()` rather than `_Exit()` or `_exit()`.

#### 10082 RATIONALE

##### 10083 Process Termination

10084 Early proposals drew a distinction between normal and abnormal process termination.  
 10085 Abnormal termination was caused only by certain signals and resulted in implementation-  
 10086 defined “actions”, as discussed below. Subsequent proposals distinguished three types of  
 10087 termination: *normal termination* (as in the current specification), *simple abnormal termination*, and  
 10088 *abnormal termination with actions*. Again the distinction between the two types of abnormal  
 10089 termination was that they were caused by different signals and that implementation-defined  
 10090 actions would result in the latter case. Given that these actions were completely  
 10091 implementation-defined, the early proposals were only saying when the actions could occur and  
 10092 how their occurrence could be detected, but not what they were. This was of little or no use to  
 10093 conforming applications, and thus the distinction is not made in this volume of |  
 10094 IEEE Std 1003.1-200x.

10095 The implementation-defined actions usually include, in most historical implementations, the  
 10096 creation of a file named **core** in the current working directory of the process. This file contains an  
 10097 image of the memory of the process, together with descriptive information about the process,  
 10098 perhaps sufficient to reconstruct the state of the process at the receipt of the signal.

10099 There is a potential security problem in creating a **core** file if the process was set-user-ID and the  
 10100 current user is not the owner of the program, if the process was set-group-ID and none of the  
 10101 user's groups match the group of the program, or if the user does not have permission to write in  
 10102 the current directory. In this situation, an implementation either should not create a **core** file or  
 10103 should make it unreadable by the user.

10104 Despite the silence of this volume of IEEE Std 1003.1-200x on this feature, applications are  
 10105 advised not to create files named **core** because of potential conflicts in many implementations.  
 10106 Some historical implementations use a different name than **core** for the file, such as by  
 10107 appending the process ID to the filename.

10108 **Terminating a Process**

10109 It is important that the consequences of process termination as described occur regardless of  
10110 whether the process called `_exit()` (perhaps indirectly through `exit()`) or instead was terminated  
10111 due to a signal or for some other reason. Note that in the specific case of `exit()` this means that  
10112 the *status* argument to `exit()` is treated in the same way as the *status* argument to `_exit()`.

10113 A language other than C may have other termination primitives than the C-language `exit()`  
10114 function, and programs written in such a language should use its native termination primitives,  
10115 but those should have as part of their function the behavior of `_exit()` as described.  
10116 Implementations in languages other than C are outside the scope of the present version of this  
10117 volume of IEEE Std 1003.1-200x, however.

10118 As required by the ISO C standard, using **return** from `main()` has the same behavior (other than  
10119 with respect to language scope issues) as calling `exit()` with the returned value. Reaching the end  
10120 of the `main()` function has the same behavior as calling `exit(0)`.

10121 A value of zero (or `EXIT_SUCCESS`, which is required to be zero) for the argument *status*  
10122 conventionally indicates successful termination. This corresponds to the specification for `exit()`  
10123 in the ISO C standard. The convention is followed by utilities such as *make* and various shells,  
10124 which interpret a zero status from a child process as success. For this reason, applications should  
10125 not call `exit(0)` or `_exit(0)` when they terminate unsuccessfully; for example, in signal-catching  
10126 functions.

10127 Historically, the implementation-defined process that inherits children whose parents have  
10128 terminated without waiting on them is called *init* and has a process ID of 1.

10129 The sending of a `SIGHUP` to the foreground process group when a controlling process  
10130 terminates corresponds to somewhat different historical implementations. In System V, the  
10131 kernel sends a `SIGHUP` on termination of (essentially) a controlling process. In 4.2 BSD, the  
10132 kernel does not send `SIGHUP` in a case like this, but the termination of a controlling process is  
10133 usually noticed by a system daemon, which arranges to send a `SIGHUP` to the foreground  
10134 process group with the `vhangup()` function. However, in 4.2 BSD, due to the behavior of the  
10135 shells that support job control, the controlling process is usually a shell with no other processes  
10136 in its process group. Thus, a change to make `_exit()` behave this way in such systems should not  
10137 cause problems with existing applications.

10138 The termination of a process may cause a process group to become orphaned in either of two  
10139 ways. The connection of a process group to its parent(s) outside of the group depends on both  
10140 the parents and their children. Thus, a process group may be orphaned by the termination of the  
10141 last connecting parent process outside of the group or by the termination of the last direct  
10142 descendant of the parent process(es). In either case, if the termination of a process causes a  
10143 process group to become orphaned, processes within the group are disconnected from their job  
10144 control shell, which no longer has any information on the existence of the process group.  
10145 Stopped processes within the group would languish forever. In order to avoid this problem,  
10146 newly orphaned process groups that contain stopped processes are sent a `SIGHUP` signal and a  
10147 `SIGCONT` signal to indicate that they have been disconnected from their session. The `SIGHUP`  
10148 signal causes the process group members to terminate unless they are catching or ignoring  
10149 `SIGHUP`. Under most circumstances, all of the members of the process group are stopped if any  
10150 of them are stopped.

10151 The action of sending a `SIGHUP` and a `SIGCONT` signal to members of a newly orphaned  
10152 process group is similar to the action of 4.2 BSD, which sends `SIGHUP` and `SIGCONT` to each  
10153 stopped child of an exiting process. If such children exit in response to the `SIGHUP`, any  
10154 additional descendants receive similar treatment at that time. In this volume of  
10155 IEEE Std 1003.1-200x, the signals are sent to the entire process group at the same time. Also, in

10156 this volume of IEEE Std 1003.1-200x, but not in 4.2 BSD, stopped processes may be orphaned,  
10157 but may be members of a process group that is not orphaned; therefore, the action taken at  
10158 `_exit()` must consider processes other than child processes.

10159 It is possible for a process group to be orphaned by a call to `setpgid()` or `setsid()`, as well as by  
10160 process termination. This volume of IEEE Std 1003.1-200x does not require sending `SIGHUP` and  
10161 `SIGCONT` in those cases, because, unlike process termination, those cases are not caused  
10162 accidentally by applications that are unaware of job control. An implementation can choose to  
10163 send `SIGHUP` and `SIGCONT` in those cases as an extension; such an extension must be  
10164 documented as required in `<signal.h>`.

10165 The ISO/IEC 9899:1999 standard adds the `_Exit()` function that results in immediate program  
10166 termination without triggering signals or `atexit()`-registered functions. In IEEE Std 1003.1-200x,  
10167 this is equivalent to the `_exit()` function.

#### 10168 FUTURE DIRECTIONS

10169 None.

#### 10170 SEE ALSO

10171 `atexit()`, `close()`, `fclose()`, `longjmp()`, `posix_trace_shutdown()`, `posix_trace_trid_eventid_open()`,  
10172 `semop()`, `shmget()`, `sigaction()`, `wait()`, `waitid()`, `waitpid()`, the Base Definitions volume of  
10173 IEEE Std 1003.1-200x, `<stdlib.h>`, `<unistd.h>`

#### 10174 CHANGE HISTORY

10175 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 10176 Issue 5

10177 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX  
10178 Threads Extension.

10179 Interactions with the `SA_NOCLDWAIT` flag and `SIGCHLD` signal are further clarified.

10180 The values of `status` from `exit()` are better described.

#### 10181 Issue 6

10182 Extensions beyond the ISO C standard are now marked.

10183 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by adding semantics for  
10184 typed memory.

10185 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 10186 • The `_Exit()` function is included.
- 10187 • The DESCRIPTION is updated.

10188 The description of tracing semantics is added for alignment with IEEE Std 1003.1q-2000.

10189 References to the `wait3()` function are removed.

10190 **NAME**

10191 exp, expf, expl — exponential function

10192 **SYNOPSIS**

10193 #include &lt;math.h&gt;

10194 double exp(double x);

10195 float expf(float x);

10196 long double expl(long double x);

10197 **DESCRIPTION**

10198 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 10199 conflict between the requirements described here and the ISO C standard is unintentional. This  
 10200 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

10201 These functions shall compute the base-*e* exponential of *x*.

10202 An application wishing to check for error situations should set *errno* to zero and call  
 10203 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 10204 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 10205 zero, an error has occurred.

10206 **RETURN VALUE**10207 Upon successful completion, these functions shall return the exponential value of *x*.

10208 If the correct value would cause overflow, a range error shall occur and *exp()*, *expf()*, and *expl()*  
 10209 shall return the value of the macro HUGE\_VAL, HUGE\_VALF, and HUGE\_VALL, respectively.

10210 If the correct value would cause underflow, and is not representable, a range error may occur,  
 10211 **MX** and either 0.0 (if supported), or an implementation-defined value shall be returned.

10212 **MX** If *x* is NaN, a NaN shall be returned.10213 If *x* is ±0, 1 shall be returned.10214 If *x* is -Inf, +0 shall be returned.10215 If *x* is +Inf, *x* shall be returned.

10216 If the correct value would cause underflow, and is representable, a range error may occur and  
 10217 the correct value shall be returned.

10218 **ERRORS**

10219 These functions shall fail if:

10220 Range Error The result overflows.

10221 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero, |  
 10222 then *errno* shall be set to [ERANGE]. If the integer expression |  
 10223 (math\_errhandling & MATH\_ERREXCEPT) is non-zero, then the overflow |  
 10224 floating-point exception shall be raised. |

10225 These functions may fail if:

10226 Range Error The result underflows.

10227 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero, |  
 10228 then *errno* shall be set to [ERANGE]. If the integer expression |  
 10229 (math\_errhandling & MATH\_ERREXCEPT) is non-zero, then the underflow |  
 10230 floating-point exception shall be raised. |

10231 **EXAMPLES**

10232 None.

10233 **APPLICATION USAGE**

10234 Note that for IEEE Std 754-1985 **double**,  $709.8 < x$  implies  $\exp(x)$  has overflowed. The value  $x <$   
10235  $-708.4$  implies  $\exp(x)$  has underflowed.

10236 On error, the expressions (math\_errhandling & MATH\_ERRNO) and (math\_errhandling &  
10237 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

10238 **RATIONALE**

10239 None.

10240 **FUTURE DIRECTIONS**

10241 None.

10242 **SEE ALSO**

10243 *feclearexcept()*, *fetetestexcept()*, *isnan()*, *log()*, the Base Definitions volume of IEEE Std 1003.1-200x, |  
10244 Section 4.18, Treatment of Error Conditions for Mathematical Functions, <math.h> |

10245 **CHANGE HISTORY**

10246 First released in Issue 1. Derived from Issue 1 of the SVID.

10247 **Issue 5**

10248 The DESCRIPTION is updated to indicate how an application should check for an error. This  
10249 text was previously published in the APPLICATION USAGE section.

10250 **Issue 6**10251 The *expf()* and *expl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

10252 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
10253 revised to align with the ISO/IEC 9899:1999 standard.

10254 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
10255 marked.

10256 **NAME**

10257 exp2, exp2f, exp2l — exponential base 2 functions

10258 **SYNOPSIS**

10259 #include &lt;math.h&gt;

10260 double exp2(double x);

10261 float exp2f(float x);

10262 long double exp2l(long double x);

10263 **DESCRIPTION**

10264 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 10265 conflict between the requirements described here and the ISO C standard is unintentional. This  
 10266 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

10267 These functions shall compute the base-2 exponential of  $x$ .

10268 An application wishing to check for error situations should set *errno* to zero and call  
 10269 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 10270 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 10271 zero, an error has occurred.

10272 **RETURN VALUE**10273 Upon successful completion, these functions shall return  $2^x$ .

10274 If the correct value would cause overflow, a range error shall occur and *exp2()*, *exp2f()*, and  
 10275 *exp2l()* shall return the value of the macro HUGE\_VAL, HUGE\_VALF, and HUGE\_VALL,  
 10276 respectively.

10277 If the correct value would cause underflow, and is not representable, a range error may occur,  
 10278 **MX** and either 0.0 (if supported), or an implementation-defined value shall be returned.

10279 **MX** If  $x$  is NaN, a NaN shall be returned.10280 If  $x$  is  $\pm 0$ , 1 shall be returned.10281 If  $x$  is  $-\text{Inf}$ , +0 shall be returned.10282 If  $x$  is  $+\text{Inf}$ ,  $x$  shall be returned.

10283 If the correct value would cause underflow, and is representable, a range error may occur and  
 10284 the correct value shall be returned.

10285 **ERRORS**

10286 These functions shall fail if:

10287 Range Error The result overflows.

10288 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero, |  
 10289 then *errno* shall be set to [ERANGE]. If the integer expression |  
 10290 (math\_errhandling & MATH\_ERREXCEPT) is non-zero, then the overflow |  
 10291 floating-point exception shall be raised. |

10292 These functions may fail if:

10293 Range Error The result underflows.

10294 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero, |  
 10295 then *errno* shall be set to [ERANGE]. If the integer expression |  
 10296 (math\_errhandling & MATH\_ERREXCEPT) is non-zero, then the underflow |  
 10297 floating-point exception shall be raised. |

10298 **EXAMPLES**

10299 None.

10300 **APPLICATION USAGE**

10301 For IEEE Std 754-1985 **double**,  $1024 \leq x$  implies  $\text{exp2}(x)$  has overflowed. The value  $x < -1022$   
10302 implies  $\text{exp}(x)$  has underflowed.

10303 On error, the expressions `(math_errhandling & MATH_ERRNO)` and `(math_errhandling &`  
10304 `MATH_ERREXCEPT)` are independent of each other, but at least one of them must be non-zero.

10305 **RATIONALE**

10306 None.

10307 **FUTURE DIRECTIONS**

10308 None.

10309 **SEE ALSO**

10310 *exp()*, *feclearexcept()*, *fetestexcept()*, *isnan()*, *log()*, the Base Definitions volume of |  
10311 IEEE Std 1003.1-200x, Section 4.18, Treatment of Error Conditions for Mathematical Functions, |  
10312 `<math.h>`

10313 **CHANGE HISTORY**

10314 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

10315 **NAME**

10316 expm1, expm1f, expm1l — compute exponential functions

10317 **SYNOPSIS**

10318 #include &lt;math.h&gt;

10319 double expm1(double x);

10320 float expm1f(float x);

10321 long double expm1l(long double x);

10322 **DESCRIPTION**

10323 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 10324 conflict between the requirements described here and the ISO C standard is unintentional. This  
 10325 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

10326 These functions shall compute  $e^x-1.0$ .

10327 An application wishing to check for error situations should set *errno* to zero and call  
 10328 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 10329 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 10330 zero, an error has occurred.

10331 **RETURN VALUE**10332 Upon successful completion, these functions return  $e^x-1.0$ .

10333 If the correct value would cause overflow, a range error shall occur and *expm1()*, *expm1f()*, and  
 10334 *expm1l()* shall return the value of the macro HUGE\_VAL, HUGE\_VALF, and HUGE\_VALL,  
 10335 respectively.

10336 **MX** If *x* is NaN, a NaN shall be returned.10337 If *x* is  $\pm 0$ ,  $\pm 0$  shall be returned.10338 If *x* is  $-\text{Inf}$ ,  $-1$  shall be returned.10339 If *x* is  $+\text{Inf}$ , *x* shall be returned.10340 If *x* is subnormal, a range error may occur and *x* should be returned.10341 **ERRORS**

10342 These functions shall fail if:

10343 **Range Error** The result overflows.

10344 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero, |  
 10345 then *errno* shall be set to [ERANGE]. If the integer expression |  
 10346 (math\_errhandling & MATH\_ERREXCEPT) is non-zero, then the overflow |  
 10347 floating-point exception shall be raised. |

10348 These functions may fail if:

10349 **MX** **Range Error** The value of *x* is subnormal.

10350 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero, |  
 10351 then *errno* shall be set to [ERANGE]. If the integer expression |  
 10352 (math\_errhandling & MATH\_ERREXCEPT) is non-zero, then the underflow |  
 10353 floating-point exception shall be raised. |

10354 **EXAMPLES**

10355 None.

10356 **APPLICATION USAGE**10357 The value of *expm1(x)* may be more accurate than *exp(x)*-1.0 for small values of *x*.10358 The *expm1()* and *log1p()* functions are useful for financial calculations of  $((1+x)^n-1)/x$ , namely:10359  $\text{expm1}(n * \text{log1p}(x))/x$ 10360 when *x* is very small (for example, when calculating small daily interest rates). These functions  
10361 also simplify writing accurate inverse hyperbolic functions.10362 For IEEE Std 754-1985 **double**,  $709.8 < x$  implies *expm1(x)* has overflowed.10363 On error, the expressions (math\_errhandling & MATH\_ERRNO) and (math\_errhandling &  
10364 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.10365 **RATIONALE**

10366 None.

10367 **FUTURE DIRECTIONS**

10368 None.

10369 **SEE ALSO**10370 *exp()*, *feclearexcept()*, *fetestexcept()*, *ilogb()*, *log1p()*, the Base Definitions volume of |  
10371 IEEE Std 1003.1-200x, Section 4.18, Treatment of Error Conditions for Mathematical Functions, |  
10372 <math.h>10373 **CHANGE HISTORY**

10374 First released in Issue 4, Version 2.

10375 **Issue 5**

10376 Moved from X/OPEN UNIX extension to BASE.

10377 **Issue 6**10378 The *expm1f()* and *expm1l()* functions are added for alignment with the ISO/IEC 9899:1999  
10379 standard.10380 The *expm1()* function is no longer marked as an extension. |10381 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are |  
10382 revised to align with the ISO/IEC 9899:1999 standard.10383 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
10384 marked.

10385 **NAME**

10386 fabs, fabsf, fabsl — absolute value function

10387 **SYNOPSIS**

10388 #include &lt;math.h&gt;

10389 double fabs(double x);

10390 float fabsf(float x);

10391 long double fabsl(long double x);

10392 **DESCRIPTION**

10393 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
10394 conflict between the requirements described here and the ISO C standard is unintentional. This  
10395 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

10396 These functions shall compute the absolute value of their argument  $x$ ,  $|x|$ .10397 **RETURN VALUE**10398 Upon successful completion, these functions shall return the absolute value of  $x$ .10399 **MX** If  $x$  is NaN, a NaN shall be returned.10400 If  $x$  is  $\pm 0$ ,  $+0$  shall be returned.10401 If  $x$  is  $\pm\text{Inf}$ ,  $+\text{Inf}$  shall be returned.10402 **ERRORS**

10403 No errors are defined.

10404 **EXAMPLES**

10405 None.

10406 **APPLICATION USAGE**

10407 None.

10408 **RATIONALE**

10409 None.

10410 **FUTURE DIRECTIONS**

10411 None.

10412 **SEE ALSO**10413 *isnan()*, the Base Definitions volume of IEEE Std 1003.1-200x, <math.h>10414 **CHANGE HISTORY**

10415 First released in Issue 1. Derived from Issue 1 of the SVID.

10416 **Issue 5**10417 The DESCRIPTION is updated to indicate how an application should check for an error. This  
10418 text was previously published in the APPLICATION USAGE section.10419 **Issue 6**10420 The *fabsf()* and *fabsl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.10421 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
10422 revised to align with the ISO/IEC 9899:1999 standard.10423 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
10424 marked.

10425 **NAME**

10426 fattach — attach a STREAMS-based file descriptor to a file in the file system name space  
 10427 (STREAMS)

10428 **SYNOPSIS**

```
10429 XSR #include <stropts.h>
```

```
10430 int fattach(int fildev, const char *path);
```

10431

10432 **DESCRIPTION**

10433 The *fattach()* function shall attach a STREAMS-based file descriptor to a file, effectively |  
 10434 associating a pathname with *fildev*. The application shall ensure that the *fildev* argument is a |  
 10435 valid open file descriptor associated with a STREAMS file. The *path* argument points to a |  
 10436 pathname of an existing file. The application shall have the appropriate privileges, or is the |  
 10437 owner of the file named by *path* and has write permission. A successful call to *fattach()* shall |  
 10438 cause all pathnames that name the file named by *path* to name the STREAMS file associated with |  
 10439 *fildev*, until the STREAMS file is detached from the file. A STREAMS file can be attached to more |  
 10440 than one file and can have several pathnames associated with it. |

10441 The attributes of the named STREAMS file shall be initialized as follows: the permissions, user |  
 10442 ID, group ID, and times are set to those of the file named by *path*, the number of links is set to 1, |  
 10443 and the size and device identifier are set to those of the STREAMS file associated with *fildev*. If |  
 10444 any attributes of the named STREAMS file are subsequently changed (for example, by *chmod()*), |  
 10445 neither the attributes of the underlying file nor the attributes of the STREAMS file to which *fildev* |  
 10446 refers shall be affected.

10447 File descriptors referring to the underlying file, opened prior to an *fattach()* call, shall continue to |  
 10448 refer to the underlying file.

10449 **RETURN VALUE**

10450 Upon successful completion, *fattach()* shall return 0. Otherwise, -1 shall be returned and *errno* |  
 10451 set to indicate the error.

10452 **ERRORS**

10453 The *fattach()* function shall fail if:

10454 [EACCES] Search permission is denied for a component of the path prefix, or the process |  
 10455 is the owner of *path* but does not have write permissions on the file named by |  
 10456 *path*.

10457 [EBADF] The *fildev* argument is not a valid open file descriptor.

10458 [EBUSY] The file named by *path* is currently a mount point or has a STREAMS file |  
 10459 attached to it.

10460 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path* |  
 10461 argument.

10462 [ENAMETOOLONG]

10463 The size of *path* exceeds {PATH\_MAX} or a component of *path* is longer than |  
 10464 {NAME\_MAX}.

10465 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

10466 [ENOTDIR] A component of the path prefix is not a directory.

10467 [EPERM] The effective user ID of the process is not the owner of the file named by *path* |  
 10468 and the process does not have appropriate privilege.

- 10469 The *fattach()* function may fail if:
- 10470 [EINVAL] The *fdes* argument does not refer to a STREAMS file.
  - 10471 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
10472 resolution of the *path* argument.
  - 10473 [ENAMETOOLONG]  
10474 Pathname resolution of a symbolic link produced an intermediate result |  
10475 whose length exceeds {PATH\_MAX}.
  - 10476 [EXDEV] A link to a file on another file system was attempted.

10477 **EXAMPLES**10478 **Attaching a File Descriptor to a File**

10479 In the following example, *fd* refers to an open STREAMS file. The call to *fattach()* associates this  
10480 STREAM with the file */tmp/named-STREAM*, such that any future calls to open */tmp/named-*  
10481 *STREAM*, prior to breaking the attachment via a call to *fdetach()*, will instead create a new file  
10482 handle referring to the STREAMS file associated with *fd*.

```
10483 #include <stropts.h>
10484 ...
10485     int fd;
10486     char *filename = "/tmp/named-STREAM";
10487     int ret;
10488
10489     ret = fattach(fd, filename);
```

10489 **APPLICATION USAGE**

10490 The *fattach()* function behaves similarly to the traditional *mount()* function in the way a file is  
10491 temporarily replaced by the root directory of the mounted file system. In the case of *fattach()*, the  
10492 replaced file need not be a directory and the replacing file is a STREAMS file.

10493 **RATIONALE**

10494 The file attributes of a file which has been the subject of an *fattach()* call are specifically set |  
10495 because of an artefact of the original implementation. The internal mechanism was the same as |  
10496 for the *mount()* function. Since *mount()* is typically only applied to directories, the effects when |  
10497 applied to a regular file are a little surprising, especially as regards the link count which rigidly |  
10498 remains one, even if there were several links originally and despite the fact that all original links |  
10499 refer to the STREAM as long as the *fattach()* remains in effect. |

10500 **FUTURE DIRECTIONS**

10501 None.

10502 **SEE ALSO**

10503 *fdetach()*, *isastream()*, the Base Definitions volume of IEEE Std 1003.1-200x, <stropts.h>

10504 **CHANGE HISTORY**

10505 First released in Issue 4, Version 2.

10506 **Issue 5**

10507 Moved from X/OPEN UNIX extension to BASE.

10508 The [EXDEV] error is added to the list of optional errors in the ERRORS section.

10509 **Issue 6**

- 10510 This function is marked as part of the XSI STREAMS Option Group.
- 10511 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.
- 10512 The wording of the mandatory [ELOOP] error condition is updated, and a second optional
- 10513 [ELOOP] error condition is added.

10514 **NAME**

10515 fchdir — change working directory

10516 **SYNOPSIS**

10517 xSI #include &lt;unistd.h&gt;

10518 int fchdir(int *fildev*);

10519

10520 **DESCRIPTION**10521 The *fchdir()* function shall be equivalent to *chdir()* except that the directory that is to be the new |  
10522 current working directory is specified by the file descriptor *fildev*.10523 A conforming application can obtain a file descriptor for a file of type directory using *open()*,  
10524 provided that the file status flags and access modes do not contain O\_WRONLY or O\_RDWR.10525 **RETURN VALUE**10526 Upon successful completion, *fchdir()* shall return 0. Otherwise, it shall return -1 and set *errno* to  
10527 indicate the error. On failure the current working directory shall remain unchanged.10528 **ERRORS**10529 The *fchdir()* function shall fail if:10530 [EACCES] Search permission is denied for the directory referenced by *fildev*.10531 [EBADF] The *fildev* argument is not an open file descriptor.10532 [ENOTDIR] The open file descriptor *fildev* does not refer to a directory.10533 The *fchdir()* may fail if:10534 [EINTR] A signal was caught during the execution of *fchdir()*.

10535 [EIO] An I/O error occurred while reading from or writing to the file system.

10536 **EXAMPLES**

10537 None.

10538 **APPLICATION USAGE**

10539 None.

10540 **RATIONALE**

10541 None.

10542 **FUTURE DIRECTIONS**

10543 None.

10544 **SEE ALSO**10545 *chdir()*, the Base Definitions volume of IEEE Std 1003.1-200x, <unistd.h>10546 **CHANGE HISTORY**

10547 First released in Issue 4, Version 2.

10548 **Issue 5**

10549 Moved from X/OPEN UNIX extension to BASE.

10550 **NAME**

10551 fchmod — change mode of a file

10552 **SYNOPSIS**

10553 #include &lt;sys/stat.h&gt;

10554 int fchmod(int *fildev*, mode\_t *mode*);10555 **DESCRIPTION**10556 The *fchmod()* function shall be equivalent to *chmod()* except that the file whose permissions are  
10557 changed is specified by the file descriptor *fildev*.10558 SHM If *fildev* references a shared memory object, the *fchmod()* function need only affect the S\_IRUSR,  
10559 S\_IWUSR, S\_IRGRP, S\_IWGRP, S\_IROTH, and S\_IWOTH file permission bits.10560 TYM If *fildev* references a typed memory object, the behavior of *fchmod()* is unspecified. |10561 If *fildev* refers to a socket, the behavior of *fchmod()* is unspecified. |10562 XSR If *fildev* refers to a STREAM (which is *fattach()*-ed into the file system name space) the call  
10563 returns successfully, doing nothing. |10564 **RETURN VALUE**10565 Upon successful completion, *fchmod()* shall return 0. Otherwise, it shall return -1 and set *errno* to  
10566 indicate the error.10567 **ERRORS**10568 The *fchmod()* function shall fail if:10569 [EBADF] The *fildev* argument is not an open file descriptor.10570 [EPERM] The effective user ID does not match the owner of the file and the process  
10571 does not have appropriate privilege.10572 [EROFS] The file referred to by *fildev* resides on a read-only file system.10573 The *fchmod()* function may fail if:10574 XSI [EINTR] The *fchmod()* function was interrupted by a signal.10575 XSI [EINVAL] The value of the *mode* argument is invalid.10576 [EINVAL] The *fildev* argument refers to a pipe and the implementation disallows  
10577 execution of *fchmod()* on a pipe.10578 **EXAMPLES**10579 **Changing the Current Permissions for a File**10580 The following example shows how to change the permissions for a file named */home/cnd/mod1*  
10581 so that the owner and group have read/write/execute permissions, but the world only has  
10582 read/write permissions.

10583 #include &lt;sys/stat.h&gt;

10584 #include &lt;fcntl.h&gt;

10585 mode\_t mode;

10586 int fildev;

10587 ...

10588 fildev = open("/home/cnd/mod1", O\_RDWR);

10589 fchmod(fildev, S\_IRWXU | S\_IRWXG | S\_IROTH | S\_IWOTH);

**10590 APPLICATION USAGE**

10591 None.

**10592 RATIONALE**

10593 None.

**10594 FUTURE DIRECTIONS**

10595 None.

**10596 SEE ALSO**

10597 *chmod()*, *chown()*, *creat()*, *fcntl()*, *fstatvfs()*, *mknod()*, *open()*, *read()*, *stat()*, *write()*, the Base  
10598 Definitions volume of IEEE Std 1003.1-200x, <sys/stat.h>

**10599 CHANGE HISTORY**

10600 First released in Issue 4, Version 2.

**10601 Issue 5**

10602 Moved from X/OPEN UNIX extension to BASE and aligned with *fchmod()* in the POSIX  
10603 Realtime Extension. Specifically, the second paragraph of the DESCRIPTION is added and a  
10604 second instance of [EINVAL] is defined in the list of optional errors.

**10605 Issue 6**

10606 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by stating that *fchmod()*  
10607 behavior is unspecified for typed memory objects.

## 10608 NAME

10609 fchown — change owner and group of a file

## 10610 SYNOPSIS

10611 #include &lt;unistd.h&gt;

10612 int fchown(int *fildev*, uid\_t *owner*, gid\_t *group*);

## 10613 DESCRIPTION

10614 The *fchown()* function shall be equivalent to *chown()* except that the file whose owner and group  
 10615 are changed is specified by the file descriptor *fildev*.

## 10616 RETURN VALUE

10617 Upon successful completion, *fchown()* shall return 0. Otherwise, it shall return -1 and set *errno* to  
 10618 indicate the error.

## 10619 ERRORS

10620 The *fchown()* function shall fail if:10621 [EBADF] The *fildev* argument is not an open file descriptor.

10622 [EPERM] The effective user ID does not match the owner of the file or the process does  
 10623 not have appropriate privilege and `_POSIX_CHOWN_RESTRICTED` indicates  
 10624 that such privilege is required.

10625 [EROFS] The file referred to by *fildev* resides on a read-only file system.10626 The *fchown()* function may fail if:

10627 [EINVAL] The owner or group ID is not a value supported by the implementation. The  
 10628 XSR *fildev* argument refers to a pipe or socket or an *fattach()*-ed STREAM and the  
 10629 implementation disallows execution of *fchown()* on a pipe.

10630 [EIO] A physical I/O error has occurred.

10631 [EINTR] The *fchown()* function was interrupted by a signal which was caught.

## 10632 EXAMPLES

## 10633 Changing the Current Owner of a File

10634 The following example shows how to change the owner of a file named `/home/cnd/mod1` to  
 10635 “jones” and the group to “cnd”.

10636 The numeric value for the user ID is obtained by extracting the user ID from the user database  
 10637 entry associated with “jones”. Similarly, the numeric value for the group ID is obtained by  
 10638 extracting the group ID from the group database entry associated with “cnd”. This example  
 10639 assumes the calling program has appropriate privileges.

10640 #include &lt;sys/types.h&gt;

10641 #include &lt;unistd.h&gt;

10642 #include &lt;fcntl.h&gt;

10643 #include &lt;pwd.h&gt;

10644 #include &lt;grp.h&gt;

10645 struct passwd \*pwd;

10646 struct group \*grp;

10647 int fildev;

10648 ...

10649 fildev = open("/home/cnd/mod1", O\_RDWR);

10650 pwd = getpwnam("jones");

```
10651     grp = getgrnam("cnd");
10652     fchown(fildes, pwd->pw_uid, grp->gr_gid);
```

**10653 APPLICATION USAGE**

10654 None.

**10655 RATIONALE**

10656 None.

**10657 FUTURE DIRECTIONS**

10658 None.

**10659 SEE ALSO**

10660 *chown()*, the Base Definitions volume of IEEE Std 1003.1-200x, <unistd.h>

**10661 CHANGE HISTORY**

10662 First released in Issue 4, Version 2.

**10663 Issue 5**

10664 Moved from X/OPEN UNIX extension to BASE. |

**10665 Issue 6**

10666 The following changes were made to align with the IEEE P1003.1a draft standard:

10667 • Clarification is added that a call to *fchown()* may not be allowed on a pipe. |

10668 The *fchown()* function is now defined as mandatory. |

10669 **NAME**

10670 `fclose` — close a stream

10671 **SYNOPSIS**

10672 `#include <stdio.h>`

10673 `int fclose(FILE *stream);`

10674 **DESCRIPTION**

10675 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 10676 conflict between the requirements described here and the ISO C standard is unintentional. This  
 10677 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

10678 The `fclose()` function shall cause the stream pointed to by `stream` to be flushed and the associated  
 10679 file to be closed. Any unwritten buffered data for the stream shall be written to the file; any  
 10680 unread buffered data shall be discarded. Whether or not the call succeeds, the stream shall be  
 10681 disassociated from the file and any buffer set by the `setbuf()` or `setvbuf()` function shall be  
 10682 disassociated from the stream. If the associated buffer was automatically allocated, it shall be  
 10683 deallocated.

10684 CX The `fclose()` function shall mark for update the `st_ctime` and `st_mtime` fields of the underlying file,  
 10685 if the stream was writable, and if buffered data remains that has not yet been written to the file.  
 10686 The `fclose()` function shall perform the equivalent of a `close()` on the file descriptor that is  
 10687 associated with the stream pointed to by `stream`.

10688 After the call to `fclose()`, any use of `stream` results in undefined behavior.

10689 **RETURN VALUE**

10690 CX Upon successful completion, `fclose()` shall return 0; otherwise, it shall return EOF and set `errno` to  
 10691 indicate the error.

10692 **ERRORS**

10693 The `fclose()` function shall fail if:

10694 CX [EAGAIN] The O\_NONBLOCK flag is set for the file descriptor underlying `stream` and the  
 10695 process would be delayed in the write operation.

10696 CX [EBADF] The file descriptor underlying stream is not valid.

10697 CX [EFBIG] An attempt was made to write a file that exceeds the maximum file size.

10698 XSI [EFBIG] An attempt was made to write a file that exceeds the process' file size limit.

10699 CX [EFBIG] The file is a regular file and an attempt was made to write at or beyond the  
 10700 offset maximum associated with the corresponding stream.

10701 CX [EINTR] The `fclose()` function was interrupted by a signal.

10702 CX [EIO] The process is a member of a background process group attempting to write  
 10703 to its controlling terminal, TOSTOP is set, the process is neither ignoring nor  
 10704 blocking SIGTTOU, and the process group of the process is orphaned. This  
 10705 error may also be returned under implementation-defined conditions.

10706 CX [ENOSPC] There was no free space remaining on the device containing the file.

10707 CX [EPIPE] An attempt is made to write to a pipe or FIFO that is not open for reading by  
 10708 any process. A SIGPIPE signal shall also be sent to the thread.

10709 The `fclose()` function may fail if:

10710 CX [ENXIO] A request was made of a nonexistent device, or the request was outside the  
 10711 capabilities of the device.

10712 **EXAMPLES**

10713 None.

10714 **APPLICATION USAGE**

10715 None.

10716 **RATIONALE**

10717 None.

10718 **FUTURE DIRECTIONS**

10719 None.

10720 **SEE ALSO**

10721 *close()*, *fopen()*, *getrlimit()*, *ulimit()*, the Base Definitions volume of IEEE Std 1003.1-200x,  
10722 `<stdio.h>`

10723 **CHANGE HISTORY**

10724 First released in Issue 1. Derived from Issue 1 of the SVID.

10725 **Issue 5**

10726 Large File Summit extensions are added.

10727 **Issue 6**

10728 Extensions beyond the ISO C standard are now marked.

10729 The following new requirements on POSIX implementations derive from alignment with the  
10730 Single UNIX Specification:

- 10731 • The [EFBIG] error is added as part of the large file support extensions.
- 10732 • The [ENXIO] optional error condition is added.

10733 The DESCRIPTION is updated to note that the stream and any buffer are disassociated whether  
10734 or not the call succeeds. This is for alignment with the ISO/IEC 9899:1999 standard.

## 10735 NAME

10736 fcntl — file control

## 10737 SYNOPSIS

10738 OH #include &lt;unistd.h&gt;

10739 #include &lt;fcntl.h&gt;

10740 int fcntl(int *fildes*, int *cmd*, ...);

## 10741 DESCRIPTION

10742 The *fcntl()* function shall perform the operations described below on open files. The *fildes* |  
 10743 argument is a file descriptor.

10744 The available values for *cmd* are defined in <fcntl.h> and are as follows: |

10745 **F\_DUPFD** Return a new file descriptor which shall be the lowest numbered available |  
 10746 (that is, not already open) file descriptor greater than or equal to the third |  
 10747 argument, *arg*, taken as an integer of type **int**. The new file descriptor shall |  
 10748 refer to the same open file description as the original file descriptor, and shall |  
 10749 share any locks. The FD\_CLOEXEC flag associated with the new file |  
 10750 descriptor shall be cleared to keep the file open across calls to one of the *exec* |  
 10751 functions.

10752 **F\_GETFD** Get the file descriptor flags defined in <fcntl.h> that are associated with the |  
 10753 file descriptor *fildes*. File descriptor flags are associated with a single file |  
 10754 descriptor and do not affect other file descriptors that refer to the same file.

10755 **F\_SETFD** Set the file descriptor flags defined in <fcntl.h>, that are associated with *fildes*, |  
 10756 to the third argument, *arg*, taken as type **int**. If the FD\_CLOEXEC flag in the |  
 10757 third argument is 0, the file shall remain open across the *exec* functions; |  
 10758 otherwise, the file shall be closed upon successful execution of one of the *exec* |  
 10759 functions.

10760 **F\_GETFL** Get the file status flags and file access modes, defined in <fcntl.h>, for the file |  
 10761 description associated with *fildes*. The file access modes can be extracted from |  
 10762 the return value using the mask O\_ACCMODE, which is defined in <fcntl.h>. |  
 10763 File status flags and file access modes are associated with the file description |  
 10764 and do not affect other file descriptors that refer to the same file with different |  
 10765 open file descriptions.

10766 **F\_SETFL** Set the file status flags, defined in <fcntl.h>, for the file description associated |  
 10767 with *fildes* from the corresponding bits in the third argument, *arg*, taken as |  
 10768 type **int**. Bits corresponding to the file access mode and the file creation flags, |  
 10769 as defined in <fcntl.h>, that are set in *arg* shall be ignored. If any bits in *arg* |  
 10770 other than those mentioned here are changed by the application, the result is |  
 10771 unspecified.

10772 **F\_GETOWN** If *fildes* refers to a socket, get the process or process group ID specified to |  
 10773 receive SIGURG signals when out-of-band data is available. Positive values |  
 10774 indicate a process ID; negative values, other than -1, indicate a process group |  
 10775 ID. If *fildes* does not refer to a socket, the results are unspecified.

10776 **F\_SETOWN** If *fildes* refers to a socket, set the process or process group ID specified to |  
 10777 receive SIGURG signals when out-of-band data is available, using the value of |  
 10778 the third argument, *arg*, taken as type **int**. Positive values indicate a process |  
 10779 ID; negative values, other than -1, indicate a process group ID. If *fildes* does |  
 10780 not refer to a socket, the results are unspecified.

10781 The following values for *cmd* are available for advisory record locking. Record locking shall be |  
 10782 supported for regular files, and may be supported for other files. |

10783 **F\_GETLK** Get the first lock which blocks the lock description pointed to by the third |  
 10784 argument, *arg*, taken as a pointer to type **struct flock**, defined in `<fcntl.h>`. |  
 10785 The information retrieved shall overwrite the information passed to *fcntl()* in |  
 10786 the structure **flock**. If no lock is found that would prevent this lock from |  
 10787 being created, then the structure shall be left unchanged except for the lock |  
 10788 type which shall be set to **F\_UNLCK**.

10789 **F\_SETLK** Set or clear a file segment lock according to the lock description pointed to by |  
 10790 the third argument, *arg*, taken as a pointer to type **struct flock**, defined in |  
 10791 `<fcntl.h>`. **F\_SETLK** can establish shared (or read) locks (**F\_RDLCK**) or |  
 10792 exclusive (or write) locks (**F\_WRLCK**), as well as to remove either type of lock |  
 10793 (**F\_UNLCK**). **F\_RDLCK**, **F\_WRLCK**, and **F\_UNLCK** are defined in `<fcntl.h>`. |  
 10794 If a shared or exclusive lock cannot be set, *fcntl()* shall return immediately |  
 10795 with a return value of `-1`.

10796 **F\_SETLKW** This command shall be equivalent to **F\_SETLK** except that if a shared or |  
 10797 exclusive lock is blocked by other locks, the thread shall wait until the request |  
 10798 can be satisfied. If a signal that is to be caught is received while *fcntl()* is |  
 10799 waiting for a region, *fcntl()* shall be interrupted. Upon return from the signal |  
 10800 handler, *fcntl()* shall return `-1` with *errno* set to `[EINTR]`, and the lock |  
 10801 operation shall not be done.

10802 Additional implementation-defined values for *cmd* may be defined in `<fcntl.h>`. Their names |  
 10803 shall start with **F\_**.

10804 When a shared lock is set on a segment of a file, other processes shall be able to set shared locks |  
 10805 on that segment or a portion of it. A shared lock prevents any other process from setting an |  
 10806 exclusive lock on any portion of the protected area. A request for a shared lock shall fail if the |  
 10807 file descriptor was not opened with read access.

10808 An exclusive lock shall prevent any other process from setting a shared lock or an exclusive lock |  
 10809 on any portion of the protected area. A request for an exclusive lock shall fail if the file |  
 10810 descriptor was not opened with write access.

10811 The structure **flock** describes the type (*l\_type*), starting offset (*l\_whence*), relative offset (*l\_start*), |  
 10812 size (*l\_len*), and process ID (*l\_pid*) of the segment of the file to be affected.

10813 The value of *l\_whence* is **SEEK\_SET**, **SEEK\_CUR**, or **SEEK\_END**, to indicate that the relative |  
 10814 offset *l\_start* bytes shall be measured from the start of the file, current position, or end of the file, |  
 10815 respectively. The value of *l\_len* is the number of consecutive bytes to be locked. The value of |  
 10816 *l\_len* may be negative (where the definition of **off\_t** permits negative values of *l\_len*). The *l\_pid* |  
 10817 field is only used with **F\_GETLK** to return the process ID of the process holding a blocking lock. |  
 10818 After a successful **F\_GETLK** request, when a blocking lock is found, the values returned in the |  
 10819 **flock** structure shall be as follows:

10820 *l\_type* Type of blocking lock found.

10821 *l\_whence* **SEEK\_SET**.

10822 *l\_start* Start of the blocking lock.

10823 *l\_len* Length of the blocking lock.

10824 *l\_pid* Process ID of the process that holds the blocking lock.

10825 If the command is F\_SETLKW and the process must wait for another process to release a lock,  
 10826 then the range of bytes to be locked shall be determined before the *fcntl()* function blocks. If the  
 10827 file size or file descriptor seek offset change while *fcntl()* is blocked, this shall not affect the  
 10828 range of bytes locked.

10829 If *l\_len* is positive, the area affected shall start at *l\_start* and end at *l\_start+l\_len-1*. If *l\_len* is  
 10830 negative, the area affected shall start at *l\_start+l\_len* and end at *l\_start-1*. Locks may start and  
 10831 extend beyond the current end of a file, but shall not extend before the beginning of the file. A  
 10832 lock shall be set to extend to the largest possible value of the file offset for that file by setting  
 10833 *l\_len* to 0. If such a lock also has *l\_start* set to 0 and *l\_whence* is set to SEEK\_SET, the whole file  
 10834 shall be locked.

10835 There shall be at most one type of lock set for each byte in the file. Before a successful return  
 10836 from an F\_SETLK or an F\_SETLKW request when the calling process has previously existing  
 10837 locks on bytes in the region specified by the request, the previous lock type for each byte in the  
 10838 specified region shall be replaced by the new lock type. As specified above under the  
 10839 descriptions of shared locks and exclusive locks, an F\_SETLK or an F\_SETLKW request  
 10840 (respectively) shall fail or block when another process has existing locks on bytes in the specified  
 10841 region and the type of any of those locks conflicts with the type specified in the request.

10842 All locks associated with a file for a given process shall be removed when a file descriptor for  
 10843 that file is closed by that process or the process holding that file descriptor terminates. Locks are  
 10844 not inherited by a child process.

10845 A potential for deadlock occurs if a process controlling a locked region is put to sleep by  
 10846 attempting to lock another process' locked region. If the system detects that sleeping until a  
 10847 locked region is unlocked would cause a deadlock, *fcntl()* shall fail with an [EDEADLK] error.

10848 An unlock (F\_UNLCK) request in which *l\_len* is non-zero and the offset of the last byte of the  
 10849 requested segment is the maximum value for an object of type **off\_t**, when the process has an  
 10850 existing lock in which *l\_len* is 0 and which includes the last byte of the requested segment, shall  
 10851 be treated as a request to unlock from the start of the requested segment with an *l\_len* equal to 0.  
 10852 Otherwise, an unlock (F\_UNLCK) request shall attempt to unlock only the requested segment.

10853 SHM When the file descriptor *fdes* refers to a shared memory object, the behavior of *fcntl()* shall be  
 10854 the same as for a regular file except the effect of the following values for the argument *cmd* shall  
 10855 be unspecified: F\_SETFL, F\_GETLK, F\_SETLK, and F\_SETLKW.

10856 TYM If *fdes* refers to a typed memory object, the result of the *fcntl()* function is unspecified.

#### 10857 RETURN VALUE

10858 Upon successful completion, the value returned shall depend on *cmd* as follows:

10859	F_DUPFD	A new file descriptor.
10860	F_GETFD	Value of flags defined in <fcntl.h>. The return value shall not be negative.
10861	F_SETFD	Value other than -1.
10862	F_GETFL	Value of file status flags and access modes. The return value is not negative.
10863	F_SETFL	Value other than -1.
10864	F_GETLK	Value other than -1.
10865	F_SETLK	Value other than -1.
10866	F_SETLKW	Value other than -1.
10867	F_GETOWN	Value of the socket owner process or process group; this will not be -1.

- 10868           F\_SETOWN        Value other than `-1`.
- 10869           Otherwise, `-1` shall be returned and *errno* set to indicate the error.
- 10870 **ERRORS**
- 10871           The *fcntl()* function shall fail if:
- 10872           [EACCES] or [EAGAIN]
- 10873                        The *cmd* argument is `F_SETLK`; the type of lock (*l\_type*) is a shared (`F_RDLCK`) or exclusive (`F_WRLCK`) lock and the segment of a file to be locked is already exclusive-locked by another process, or the type is an exclusive lock and some portion of the segment of a file to be locked is already shared-locked or exclusive-locked by another process.
- 10874
- 10875
- 10876
- 10877
- 10878           [EBADF]        The *fildev* argument is not a valid open file descriptor, or the argument *cmd* is `F_SETLK` or `F_SETLKW`, the type of lock, *l\_type*, is a shared lock (`F_RDLCK`), and *fildev* is not a valid file descriptor open for reading, or the type of lock *l\_type*, is an exclusive lock (`F_WRLCK`), and *fildev* is not a valid file descriptor open for writing.
- 10879
- 10880
- 10881
- 10882
- 10883           [EINTR]        The *cmd* argument is `F_SETLKW` and the function was interrupted by a signal.
- 10884           [EINVAL]        The *cmd* argument is invalid, or the *cmd* argument is `F_DUPFD` and *arg* is negative or greater than or equal to `{OPEN_MAX}`, or the *cmd* argument is `F_GETLK`, `F_SETLK`, or `F_SETLKW` and the data pointed to by *arg* is not valid, or *fildev* refers to a file that does not support locking.
- 10885
- 10886
- 10887
- 10888           [EMFILE]        The argument *cmd* is `F_DUPFD` and `{OPEN_MAX}` file descriptors are currently open in the calling process, or no file descriptors greater than or equal to *arg* are available.
- 10889
- 10890
- 10891           [ENOLCK]        The argument *cmd* is `F_SETLK` or `F_SETLKW` and satisfying the lock or unlock request would result in the number of locked regions in the system exceeding a system-imposed limit.
- 10892
- 10893
- 10894           [EOVERFLOW]    One of the values to be returned cannot be represented correctly.
- 10895           [EOVERFLOW]    The *cmd* argument is `F_GETLK`, `F_SETLK`, or `F_SETLKW` and the smallest or, if *l\_len* is non-zero, the largest offset of any byte in the requested segment cannot be represented correctly in an object of type `off_t`.
- 10896
- 10897
- 10898           The *fcntl()* function may fail if:
- 10899           [EDEADLK]        The *cmd* argument is `F_SETLKW`, the lock is blocked by some lock from another process and putting the calling process to sleep, waiting for that lock to become free would cause a deadlock.
- 10900
- 10901

10902 **EXAMPLES**

10903           None.

10904 **APPLICATION USAGE**

10905           None.

10906 **RATIONALE**

10907           The ellipsis in the SYNOPSIS is the syntax specified by the ISO C standard for a variable number of arguments. It is used because System V uses pointers for the implementation of file locking functions.

10908

10909

10910           The *arg* values to `F_GETFD`, `F_SETFD`, `F_GETFL`, and `F_SETFL` all represent flag values to allow for future growth. Applications using these functions should do a read-modify-write operation

10911

10912 on them, rather than assuming that only the values defined by this volume of  
10913 IEEE Std 1003.1-200x are valid. It is a common error to forget this, particularly in the case of  
10914 F\_SETFD.

10915 This volume of IEEE Std 1003.1-200x permits concurrent read and write access to file data using  
10916 the *fcntl()* function; this is a change from the 1984 /usr/group standard and early proposals.  
10917 Without concurrency controls, this feature may not be fully utilized without occasional loss of  
10918 data.

10919 Data losses occur in several ways. One case occurs when several processes try to update the  
10920 same record, without sequencing controls; several updates may occur in parallel and the last  
10921 writer “wins”. Another case is a bit-tree or other internal list-based database that is undergoing  
10922 reorganization. Without exclusive use to the tree segment by the updating process, other reading  
10923 processes chance getting lost in the database when the index blocks are split, condensed,  
10924 inserted, or deleted. While *fcntl()* is useful for many applications, it is not intended to be overly  
10925 general and does not handle the bit-tree example well.

10926 This facility is only required for regular files because it is not appropriate for many devices such  
10927 as terminals and network connections.

10928 Since *fcntl()* works with “any file descriptor associated with that file, however it is obtained”,  
10929 the file descriptor may have been inherited through a *fork()* or *exec* operation and thus may  
10930 affect a file that another process also has open.

10931 The use of the open file description to identify what to lock requires extra calls and presents  
10932 problems if several processes are sharing an open file description, but there are too many  
10933 implementations of the existing mechanism for this volume of IEEE Std 1003.1-200x to use  
10934 different specifications.

10935 Another consequence of this model is that closing any file descriptor for a given file (whether or  
10936 not it is the same open file description that created the lock) causes the locks on that file to be  
10937 relinquished for that process. Equivalently, any close for any file/process pair relinquishes the  
10938 locks owned on that file for that process. But note that while an open file description may be  
10939 shared through *fork()*, locks are not inherited through *fork()*. Yet locks may be inherited through  
10940 one of the *exec* functions.

10941 The identification of a machine in a network environment is outside of the scope of this volume  
10942 of IEEE Std 1003.1-200x. Thus, an *l\_sysid* member, such as found in System V, is not included in  
10943 the locking structure.

10944 Changing of lock types can result in a previously locked region being split into smaller regions. |

10945 Mandatory locking was a major feature of the 1984 /usr/group standard.

10946 For advisory file record locking to be effective, all processes that have access to a file must  
10947 cooperate and use the advisory mechanism before doing I/O on the file. Enforcement-mode  
10948 record locking is important when it cannot be assumed that all processes are cooperating. For  
10949 example, if one user uses an editor to update a file at the same time that a second user executes  
10950 another process that updates the same file and if only one of the two processes is using advisory  
10951 locking, the processes are not cooperating. Enforcement-mode record locking would protect  
10952 against accidental collisions.

10953 Secondly, advisory record locking requires a process using locking to bracket each I/O operation  
10954 with lock (or test) and unlock operations. With enforcement-mode file and record locking, a  
10955 process can lock the file once and unlock when all I/O operations have been completed.  
10956 Enforcement-mode record locking provides a base that can be enhanced; for example, with  
10957 sharable locks. That is, the mechanism could be enhanced to allow a process to lock a file so  
10958 other processes could read it, but none of them could write it.

10959 Mandatory locks were omitted for several reasons:

- 10960 1. Mandatory lock setting was done by multiplexing the set-group-ID bit in most  
10961 implementations; this was confusing, at best.
- 10962 2. The relationship to file truncation as supported in 4.2 BSD was not well specified.
- 10963 3. Any publicly readable file could be locked by anyone. Many historical implementations  
10964 keep the password database in a publicly readable file. A malicious user could thus  
10965 prohibit logins. Another possibility would be to hold open a long-distance telephone line.
- 10966 4. Some demand-paged historical implementations offer memory mapped files, and  
10967 enforcement cannot be done on that type of file.

10968 Since sleeping on a region is interrupted with any signal, *alarm()* may be used to provide a  
10969 timeout facility in applications requiring it. This is useful in deadlock detection. Since |  
10970 implementation of full deadlock detection is not always feasible, the [EDEADLK] error was |  
10971 made optional.

#### 10972 FUTURE DIRECTIONS

10973 None.

#### 10974 SEE ALSO

10975 *close()*, *exec*, *open()*, *sigaction()*, the Base Definitions volume of IEEE Std 1003.1-200x, <*fcntl.h*>,  
10976 <*signal.h*>, <*unistd.h*>

#### 10977 CHANGE HISTORY

10978 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 10979 Issue 5

10980 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX  
10981 Threads Extension.

10982 Large File Summit extensions are added.

#### 10983 Issue 6

10984 In the SYNOPSIS, the optional include of the <*sys/types.h*> header is removed.

10985 The following new requirements on POSIX implementations derive from alignment with the  
10986 Single UNIX Specification:

- 10987 • The requirement to include <*sys/types.h*> has been removed. Although <*sys/types.h*> was  
10988 required for conforming implementations of previous POSIX specifications, it was not  
10989 required for UNIX applications.
- 10990 • In the DESCRIPTION, sentences describing behavior when *L\_len* is negative are now  
10991 mandated, and the description of unlock (F\_UNLOCK) when *L\_len* is non-negative is  
10992 mandated.
- 10993 • In the ERRORS section, the [EINVAL] error condition has the case mandated when the *cmd* is  
10994 invalid, and two [E\_OVERFLOW] error conditions are added.

10995 The F\_GETOWN and F\_SETOWN values are added for sockets.

10996 The following changes were made to align with the IEEE P1003.1a draft standard:

- 10997 • Clarification is added that the extent of the bytes locked is determined prior to the blocking  
10998 action.

10999 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that  
11000 *fcntl()* results are unspecified for typed memory objects.

11001

The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

11002 **NAME**11003       fcvt — convert a floating-point number to a string (**LEGACY**)11004 **SYNOPSIS**

11005 xSI     #include &lt;stdlib.h&gt;

11006       char \*fcvt(double *value*, int *ndigit*, int \*restrict *decpt*,  
11007           int \*restrict *sign*);

11008

11009 **DESCRIPTION**11010       Refer to *ecvt()*.

11011 **NAME**11012 fdatasync — synchronize the data of a file (**REALTIME**)11013 **SYNOPSIS**11014 SIO `#include <unistd.h>`11015 `int fdatasync(int fildes);`

11016

11017 **DESCRIPTION**11018 The *fdatasync()* function shall force all currently queued I/O operations associated with the file  
11019 indicated by file descriptor *fil*des to the synchronized I/O completion state.11020 The functionality shall be equivalent to *fsync()* with the symbol `_POSIX_SYNCHRONIZED_IO` |  
11021 defined, with the exception that all I/O operations shall be completed as defined for |  
11022 synchronized I/O data integrity completion.11023 **RETURN VALUE**11024 If successful, the *fdatasync()* function shall return the value 0; otherwise, the function shall return  
11025 the value `-1` and set *errno* to indicate the error. If the *fdatasync()* function fails, outstanding I/O  
11026 operations are not guaranteed to have been completed.11027 **ERRORS**11028 The *fdatasync()* function shall fail if:11029 [EBADF] The *fil*des argument is not a valid file descriptor open for writing.

11030 [EINVAL] This implementation does not support synchronized I/O for this file.

11031 In the event that any of the queued I/O operations fail, *fdatasync()* shall return the error  
11032 conditions defined for *read()* and *write()*.11033 **EXAMPLES**

11034 None.

11035 **APPLICATION USAGE**

11036 None.

11037 **RATIONALE**

11038 None.

11039 **FUTURE DIRECTIONS**

11040 None.

11041 **SEE ALSO**11042 *ai*o\_11043 *fsync()*, *fcntl()*, *fsync()*, *open()*, *read()*, *write()*, the Base Definitions volume of  
IEEE Std 1003.1-200x, `<unistd.h>`11044 **CHANGE HISTORY**

11045 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

11046 **Issue 6**11047 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
11048 implementation does not support the Synchronized Input and Output option.11049 The *fdatasync()* function is marked as part of the Synchronized Input and Output option.

11050 **NAME**11051 fdetach — detach a name from a STREAMS-based file descriptor (**STREAMS**)11052 **SYNOPSIS**

11053 XSR #include &lt;stropts.h&gt;

11054 int fdetach(const char \*path);

11055

11056 **DESCRIPTION**

11057 The *fdetach()* function shall detach a STREAMS-based file from the file to which it was attached |  
 11058 by a previous call to *fattach()*. The *path* argument points to the pathname of the attached |  
 11059 STREAMS file. The process shall have appropriate privileges or be the owner of the file. A |  
 11060 successful call to *fdetach()* shall cause all pathnames that named the attached STREAMS file to |  
 11061 again name the file to which the STREAMS file was attached. All subsequent operations on *path* |  
 11062 shall operate on the underlying file and not on the STREAMS file.

11063 All open file descriptions established while the STREAMS file was attached to the file referenced |  
 11064 by *path* shall still refer to the STREAMS file after the *fdetach()* has taken effect.

11065 If there are no open file descriptors or other references to the STREAMS file, then a successful |  
 11066 call to *fdetach()* shall have be equivalent to performing the last *close()* on the attached file. |

11067 **RETURN VALUE**

11068 Upon successful completion, *fdetach()* shall return 0; otherwise, it shall return -1 and set *errno* to |  
 11069 indicate the error.

11070 **ERRORS**11071 The *fdetach()* function shall fail if:

11072 [EACCES] Search permission is denied on a component of the path prefix.

11073 [EINVAL] The *path* argument names a file that is not currently attached.

11074 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path* |  
 11075 argument.

11076 [ENAMETOOLONG]

11077 The size of a pathname exceeds {PATH\_MAX} or a pathname component is |  
 11078 longer than {NAME\_MAX}. |

11079 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

11080 [ENOTDIR] A component of the path prefix is not a directory.

11081 [EPERM] The effective user ID is not the owner of *path* and the process does not have |  
 11082 appropriate privileges.

11083 The *fdetach()* function may fail if:

11084 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during |  
 11085 resolution of the *path* argument.

11086 [ENAMETOOLONG]

11087 Pathname resolution of a symbolic link produced an intermediate result |  
 11088 whose length exceeds {PATH\_MAX}.

11089 **EXAMPLES**11090 **Detaching a File**

11091 The following example detaches the STREAMS-based file **/tmp/named-STREAM** from the file to  
11092 which it was attached by a previous, successful call to *fattach()*. Subsequent calls to open this  
11093 file refer to the underlying file, not to the STREAMS file.

```
11094 #include <stropts.h>
11095 ...
11096     char *filename = "/tmp/named-STREAM";
11097     int ret;
11098     ret = fdetach(filename);
```

11099 **APPLICATION USAGE**

11100 None.

11101 **RATIONALE**

11102 None.

11103 **FUTURE DIRECTIONS**

11104 None.

11105 **SEE ALSO**

11106 *fattach()*, the Base Definitions volume of IEEE Std 1003.1-200x, <stropts.h>

11107 **CHANGE HISTORY**

11108 First released in Issue 4, Version 2.

11109 **Issue 5**

11110 Moved from X/OPEN UNIX extension to BASE.

11111 **Issue 6**

11112 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

11113 The wording of the mandatory [ELOOP] error condition is updated, and a second optional  
11114 [ELOOP] error condition is added.

11115 **NAME**

11116 `fdim`, `fdimf`, `fdiml` — compute positive difference between two floating-point numbers

11117 **SYNOPSIS**

11118 `#include <math.h>`

11119 `double fdim(double x, double y);`

11120 `float fdimf(float x, float y);`

11121 `long double fdiml(long double x, long double y);`

11122 **DESCRIPTION**

11123 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 11124 conflict between the requirements described here and the ISO C standard is unintentional. This  
 11125 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

11126 These functions shall determine the positive difference between their arguments. If  $x$  is greater  
 11127 than  $y$ ,  $x-y$  is returned. If  $x$  is less than or equal to  $y$ ,  $+0$  is returned.

11128 An application wishing to check for error situations should set `errno` to zero and call  
 11129 `feclearexcept(FE_ALL_EXCEPT)` before calling these functions. On return, if `errno` is non-zero or  
 11130 `fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW)` is non-  
 11131 zero, an error has occurred.

11132 **RETURN VALUE**

11133 Upon successful completion, these functions shall return the positive difference value.

11134 If  $x-y$  is positive and overflows, a range error shall occur and `fdim()`, `fdimf()`, and `fdiml()` shall  
 11135 return the value of the macro `HUGE_VAL`, `HUGE_VALF`, and `HUGE_VALL`, respectively.

11136 **XSI** If  $x-y$  is positive and underflows, a range error may occur, and either  $(x-y)$  (if representable), or  
 11137 `0.0` (if supported), or an implementation-defined value shall be returned.

11138 **MX** If  $x$  or  $y$  is NaN, a NaN shall be returned.

11139 **ERRORS**

11140 The `fdim()` function shall fail if:

11141 **Range Error** The result overflows.

11142 If the integer expression `(math_errhandling & MATH_ERRNO)` is non-zero, |  
 11143 then `errno` shall be set to `[ERANGE]`. If the integer expression |  
 11144 `(math_errhandling & MATH_ERREXCEPT)` is non-zero, then the overflow |  
 11145 floating-point exception shall be raised. |

11146 The `fdim()` function may fail if:

11147 **Range Error** The result underflows.

11148 If the integer expression `(math_errhandling & MATH_ERRNO)` is non-zero, |  
 11149 then `errno` shall be set to `[ERANGE]`. If the integer expression |  
 11150 `(math_errhandling & MATH_ERREXCEPT)` is non-zero, then the underflow |  
 11151 floating-point exception shall be raised. |

11152 **EXAMPLES**

11153           None.

11154 **APPLICATION USAGE**

11155           On implementations supporting IEEE Std 754-1985,  $x-y$  cannot underflow, and hence the 0.0  
11156           return value is shaded as an extension for implementations supporting the XSI extension rather  
11157           than an MX extension.

11158           On error, the expressions (math\_errhandling & MATH\_ERRNO) and (math\_errhandling &  
11159           MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

11160 **RATIONALE**

11161           None.

11162 **FUTURE DIRECTIONS**

11163           None.

11164 **SEE ALSO**

11165           *feclearexcept()*, *fetestexcept()*, *fmax()*, *fmin()*, the Base Definitions volume of IEEE Std 1003.1-200x, |  
11166           Section 4.18, Treatment of Error Conditions for Mathematical Functions, <math.h> |

11167 **CHANGE HISTORY**

11168           First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

11169 **NAME**

11170 fdopen — associate a stream with a file descriptor

11171 **SYNOPSIS**

11172 cx #include &lt;stdio.h&gt;

11173 FILE \*fdopen(int *fil-des*, const char \**mode*);

11174

11175 **DESCRIPTION**11176 The *fdopen()* function shall associate a stream with a file descriptor.11177 The *mode* argument is a character string having one of the following values:11178 *r* or *rb* Open a file for reading.11179 *w* or *wb* Open a file for writing.11180 *a* or *ab* Open a file for writing at end of file.11181 *r+* or *rb+* or *r+b* Open a file for update (reading and writing).11182 *w+* or *wb+* or *w+b* Open a file for update (reading and writing).11183 *a+* or *ab+* or *a+b* Open a file for update (reading and writing) at end of file.11184 The meaning of these flags is exactly as specified in *fopen()*, except that modes beginning with *w* |  
11185 shall not cause truncation of the file. |11186 Additional values for the *mode* argument may be supported by an implementation.11187 The application shall ensure that the mode of the stream as expressed by the *mode* argument is  
11188 allowed by the file access mode of the open file description to which *fil-des* refers. The file  
11189 position indicator associated with the new stream is set to the position indicated by the file  
11190 offset associated with the file descriptor.11191 The error and end-of-file indicators for the stream shall be cleared. The *fdopen()* function may  
11192 cause the *st\_atime* field of the underlying file to be marked for update.11193 SHM If *fil-des* refers to a shared memory object, the result of the *fdopen()* function is unspecified.11194 TYM If *fil-des* refers to a typed memory object, the result of the *fdopen()* function is unspecified.11195 The *fdopen()* function shall preserve the offset maximum previously set for the open file  
11196 description corresponding to *fil-des*.11197 **RETURN VALUE**11198 Upon successful completion, *fdopen()* shall return a pointer to a stream; otherwise, a null pointer  
11199 shall be returned and *errno* set to indicate the error.11200 **ERRORS**11201 The *fdopen()* function may fail if:11202 [EBADF] The *fil-des* argument is not a valid file descriptor.11203 [EINVAL] The *mode* argument is not a valid mode.

11204 [EMFILE] {FOPEN\_MAX} streams are currently open in the calling process.

11205 [EMFILE] {STREAM\_MAX} streams are currently open in the calling process.

11206 [ENOMEM] Insufficient space to allocate a buffer.

11207 **EXAMPLES**

11208 None.

11209 **APPLICATION USAGE**

11210 File descriptors are obtained from calls like *open()*, *dup()*, *creat()*, or *pipe()*, which open files but  
11211 do not return streams.

11212 **RATIONALE**

11213 The file descriptor may have been obtained from *open()*, *creat()*, *pipe()*, *dup()*, or *fcntl()*;  
11214 inherited through *fork()* or *exec*; or perhaps obtained by implementation-defined means, such as  
11215 the 4.3 BSD *socket()* call.

11216 The meanings of the *mode* arguments of *fdopen()* and *fopen()* differ. With *fdopen()*, open for write  
11217 (*w* or *w+*) does not truncate, and append (*a* or *a+*) cannot create for writing. The *mode* argument  
11218 formats that include *a b* are allowed for consistency with the ISO C standard function *fopen()*.  
11219 The *b* has no effect on the resulting stream. Although not explicitly required by this volume of  
11220 IEEE Std 1003.1-200x, a good implementation of append (*a*) mode would cause the *O\_APPEND*  
11221 flag to be set.

11222 **FUTURE DIRECTIONS**

11223 None.

11224 **SEE ALSO**

11225 *fclose()*, *fopen()*, *open()*, the Base Definitions volume of IEEE Std 1003.1-200x, <*stdio.h*>, Section  
11226 2.5.1 (on page 485)

11227 **CHANGE HISTORY**

11228 First released in Issue 1. Derived from Issue 1 of the SVID.

11229 **Issue 5**

11230 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension.

11231 Large File Summit extensions are added.

11232 **Issue 6**

11233 The following new requirements on POSIX implementations derive from alignment with the  
11234 Single UNIX Specification:

- 11235 • In the DESCRIPTION, the use and setting of the *mode* argument are changed to include  
11236 binary streams.
- 11237 • In the DESCRIPTION, text is added for large file support to indicate setting of the offset  
11238 maximum in the open file description.
- 11239 • All errors identified in the ERRORS section are added.
- 11240 • In the DESCRIPTION, text is added that the *fdopen()* function may cause *st\_atime* to be  
11241 updated.

11242 The following changes were made to align with the IEEE P1003.1a draft standard:

- 11243 • Clarification is added that it is the responsibility of the application to ensure that the mode is  
11244 compatible with the open file descriptor.

11245 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that  
11246 *fdopen()* results are unspecified for typed memory objects.

11247 **NAME**

11248           feclearexcept — clear floating-point exception

11249 **SYNOPSIS**

11250           #include &lt;fenv.h&gt;

11251           int feclearexcept(int *excepts*);11252 **DESCRIPTION**

11253 *cx*       The functionality described on this reference page is aligned with the ISO C standard. Any  
11254       conflict between the requirements described here and the ISO C standard is unintentional. This  
11255       volume of IEEE Std 1003.1-200x defers to the ISO C standard.

11256       The *feclearexcept()* function shall attempt to clear the supported floating-point exceptions |  
11257       represented by *excepts*. |

11258 **RETURN VALUE**

11259       If the argument is zero or if all the specified exceptions were successfully cleared, *feclearexcept()* |  
11260       shall return zero. Otherwise, it shall return a non-zero value. |

11261 **ERRORS**

11262       No errors are defined.

11263 **EXAMPLES**

11264       None.

11265 **APPLICATION USAGE**

11266       None.

11267 **RATIONALE**

11268       None.

11269 **FUTURE DIRECTIONS**

11270       None.

11271 **SEE ALSO**

11272       *fegetexceptflag()*, *feraiseexcept()*, *fesetexceptflag()*, *fetestexcept()*, the Base Definitions volume of  
11273       IEEE Std 1003.1-200x, <**fenv.h**>

11274 **CHANGE HISTORY**

11275       First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard. |

11276       ISO/IEC 9899:1999 standard, Technical Corrigendum No. 1 is incorporated. |

11277 **NAME**

11278 fegetenv, fesetenv — get and set current floating-point environment

11279 **SYNOPSIS**

11280 #include <fenv.h>

11281 int fegetenv(fenv\_t \*envp);

11282 int fesetenv(const fenv\_t \*envp);

11283 **DESCRIPTION**

11284 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
11285 conflict between the requirements described here and the ISO C standard is unintentional. This  
11286 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

11287 The *fegetenv()* function shall attempt to store the current floating-point environment in the object  
11288 pointed to by *envp*.

11289 The *fesetenv()* function shall attempt to establish the floating-point environment represented by  
11290 the object pointed to by *envp*. The argument *envp* shall point to an object set by a call to  
11291 *fegetenv()* or *feholdexcept()*, or equal a floating-point environment macro. The *fesetenv()* function  
11292 does not raise floating-point exceptions, but only installs the state of the floating-point status  
11293 flags represented through its argument.

11294 **RETURN VALUE**

11295 If the representation was successfully stored, *fegetenv()* shall return zero. Otherwise, it shall  
11296 return a non-zero value. If the environment was successfully established, *fesetenv()* shall return  
11297 zero. Otherwise, it shall return a non-zero value.

11298 **ERRORS**

11299 No errors are defined.

11300 **EXAMPLES**

11301 None.

11302 **APPLICATION USAGE**

11303 None.

11304 **RATIONALE**

11305 None.

11306 **FUTURE DIRECTIONS**

11307 None.

11308 **SEE ALSO**

11309 *feholdexcept()*, *feupdateenv()*, the Base Definitions volume of IEEE Std 1003.1-200x, <fenv.h>

11310 **CHANGE HISTORY**

11311 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

11312 ISO/IEC 9899:1999 standard, Technical Corrigendum No. 1 is incorporated.

11313 **NAME**

11314 fegetexceptflag, fesetexceptflag — get and set floating-point status flags

11315 **SYNOPSIS**

11316 #include &lt;fenv.h&gt;

11317 int fegetexceptflag(fexcept\_t \*flagp, int excepts);

11318 int fesetexceptflag(const fexcept\_t \*flagp, int excepts);

11319 **DESCRIPTION**

11320 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 11321 conflict between the requirements described here and the ISO C standard is unintentional. This  
 11322 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

11323 The *fegetexceptflag()* function shall attempt to store an implementation-defined representation of  
 11324 the states of the floating-point status flags indicated by the argument *excepts* in the object  
 11325 pointed to by the argument *flagp*.

11326 The *fesetexceptflag()* function shall attempt to set the floating-point status flags indicated by the  
 11327 argument *excepts* to the states stored in the object pointed to by *flagp*. The value pointed to by  
 11328 *flagp* shall have been set by a previous call to *fegetexceptflag()* whose second argument  
 11329 represented at least those floating-point exceptions represented by the argument *excepts*. This  
 11330 function does not raise floating-point exceptions, but only sets the state of the flags.

11331 **RETURN VALUE**

11332 If the representation was successfully stored, *fegetexceptflag()* shall return zero. Otherwise, it  
 11333 shall return a non-zero value. If the *excepts* argument is zero or if all the specified exceptions  
 11334 were successfully set, *fesetexceptflag()* shall return zero. Otherwise, it shall return a non-zero  
 11335 value.

11336 **ERRORS**

11337 No errors are defined.

11338 **EXAMPLES**

11339 None.

11340 **APPLICATION USAGE**

11341 None.

11342 **RATIONALE**

11343 None.

11344 **FUTURE DIRECTIONS**

11345 None.

11346 **SEE ALSO**

11347 *feclearexcept()*, *feraiseexcept()*, *fetestexcept()*, the Base Definitions volume of IEEE Std 1003.1-200x,  
 11348 <fenv.h>

11349 **CHANGE HISTORY**

11350 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

11351 ISO/IEC 9899:1999 standard, Technical Corrigendum No. 1 is incorporated.

11352 **NAME**

11353 fegetround, fesetround — get and set current rounding direction

11354 **SYNOPSIS**

```
11355 #include <fenv.h>
11356 int fegetround(void);
11357 int fesetround(int round);
```

11358 **DESCRIPTION**

11359 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
11360 conflict between the requirements described here and the ISO C standard is unintentional. This  
11361 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

11362 The *fegetround()* function shall get the current rounding direction.

11363 The *fesetround()* function shall establish the rounding direction represented by its argument  
11364 *round*. If the argument is not equal to the value of a rounding direction macro, the rounding  
11365 direction is not changed.

11366 **RETURN VALUE**

11367 The *fegetround()* function shall return the value of the rounding direction macro representing the  
11368 current rounding direction or a negative value if there is no such rounding direction macro or  
11369 the current rounding direction is not determinable.

11370 The *fesetround()* function shall return a zero value if and only if the requested rounding direction  
11371 was established.

11372 **ERRORS**

11373 No errors are defined.

11374 **EXAMPLES**

11375 The following example saves, sets, and restores the rounding direction, reporting an error and  
11376 aborting if setting the rounding direction fails:

```
11377 #include <fenv.h>
11378 #include <assert.h>
11379 void f(int round_dir)
11380 {
11381     #pragma STDC FENV_ACCESS ON
11382     int save_round;
11383     int setround_ok;
11384     save_round = fegetround();
11385     setround_ok = fesetround(round_dir);
11386     assert(setround_ok == 0);
11387     /* ... */
11388     fesetround(save_round);
11389     /* ... */
11390 }
```

11391 **APPLICATION USAGE**

11392 None.

11393 **RATIONALE**

11394 None.

11395 **FUTURE DIRECTIONS**

11396           None.

11397 **SEE ALSO**

11398           The Base Definitions volume of IEEE Std 1003.1-200x, <fenv.h>

11399 **CHANGE HISTORY**

11400           First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard. |

11401           ISO/IEC 9899: 1999 standard, Technical Corrigendum No. 1 is incorporated. |

11402 **NAME**

11403 feholdexcept — save current floating-point environment

11404 **SYNOPSIS**

11405 #include <fenv.h>

11406 int feholdexcept(fenv\_t \*envp);

11407 **DESCRIPTION**

11408 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
11409 conflict between the requirements described here and the ISO C standard is unintentional. This  
11410 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

11411 The *feholdexcept()* function shall save the current floating-point environment in the object  
11412 pointed to by *envp*, clear the floating-point status flags, and then install a non-stop (continue on  
11413 floating-point exceptions) mode, if available, for all floating-point exceptions.

11414 **RETURN VALUE**

11415 The *feholdexcept()* function shall return zero if and only if non-stop floating-point exception  
11416 handling was successfully installed.

11417 **ERRORS**

11418 No errors are defined.

11419 **EXAMPLES**

11420 None.

11421 **APPLICATION USAGE**

11422 None.

11423 **RATIONALE**

11424 The *feholdexcept()* function should be effective on typical IEC 60559:1989 standard  
11425 implementations which have the default non-stop mode and at least one other mode for trap  
11426 handling or aborting. If the implementation provides only the non-stop mode, then installing the  
11427 non-stop mode is trivial.

11428 **FUTURE DIRECTIONS**

11429 None.

11430 **SEE ALSO**

11431 *fegetenv()*, *fesetenv()*, *feupdateenv()*, the Base Definitions volume of IEEE Std 1003.1-200x,  
11432 <fenv.h>

11433 **CHANGE HISTORY**

11434 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

11435 **NAME**

11436 feof — test end-of-file indicator on a stream

11437 **SYNOPSIS**

11438 #include &lt;stdio.h&gt;

11439 int feof(FILE \*stream);

11440 **DESCRIPTION**

11441 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
11442 conflict between the requirements described here and the ISO C standard is unintentional. This  
11443 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

11444 The *feof()* function shall test the end-of-file indicator for the stream pointed to by *stream*.11445 **RETURN VALUE**11446 The *feof()* function shall return non-zero if and only if the end-of-file indicator is set for *stream*.11447 **ERRORS**

11448 No errors are defined.

11449 **EXAMPLES**

11450 None.

11451 **APPLICATION USAGE**

11452 None.

11453 **RATIONALE**

11454 None.

11455 **FUTURE DIRECTIONS**

11456 None.

11457 **SEE ALSO**11458 *clearerr()*, *ferror()*, *fopen()*, the Base Definitions volume of IEEE Std 1003.1-200x, <stdio.h>11459 **CHANGE HISTORY**

11460 First released in Issue 1. Derived from Issue 1 of the SVID.

11461 **NAME**11462        `feraiseexcept` — raise floating-point exception11463 **SYNOPSIS**11464        `#include <fenv.h>`11465        `int feraiseexcept(int excepts);`11466 **DESCRIPTION**11467 **cx**        The functionality described on this reference page is aligned with the ISO C standard. Any  
11468        conflict between the requirements described here and the ISO C standard is unintentional. This  
11469        volume of IEEE Std 1003.1-200x defers to the ISO C standard.11470        The `feraiseexcept()` function shall attempt to raise the supported floating-point exceptions |  
11471        represented by the argument `excepts`. The order in which these floating-point exceptions are |  
11472        raised is unspecified. Whether the `feraiseexcept()` function additionally raises the inexact |  
11473        floating-point exception whenever it raises the overflow or underflow floating-point exception is |  
11474        implementation-defined.11475 **RETURN VALUE**11476        If the argument is zero or if all the specified exceptions were successfully raised, `feraiseexcept()` |  
11477        shall return zero. Otherwise, it shall return a non-zero value. |11478 **ERRORS**

11479        No errors are defined.

11480 **EXAMPLES**

11481        None.

11482 **APPLICATION USAGE**11483        The effect is intended to be similar to that of floating-point exceptions raised by arithmetic  
11484        operations. Hence, enabled traps for floating-point exceptions raised by this function are taken.11485 **RATIONALE**11486        Raising overflow or underflow is allowed to also raise inexact because on some architectures the  
11487        only practical way to raise an exception is to execute an instruction that has the exception as a  
11488        side effect. The function is not restricted to accept only valid coincident expressions for atomic  
11489        operations, so the function can be used to raise exceptions accrued over several operations.11490 **FUTURE DIRECTIONS**

11491        None.

11492 **SEE ALSO**11493        `feclearexcept()`, `fegetexceptflag()`, `fesetexceptflag()`, `fetestexcept()`, the Base Definitions volume of  
11494        IEEE Std 1003.1-200x, `<fenv.h>`11495 **CHANGE HISTORY**

11496        First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard. |

11497        ISO/IEC 9899:1999 standard, Technical Corrigendum No. 1 is incorporated. |

11498 **NAME**

11499           ferror — test error indicator on a stream

11500 **SYNOPSIS**

11501           #include &lt;stdio.h&gt;

11502           int ferror(FILE \**stream*);11503 **DESCRIPTION**

11504 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
11505           conflict between the requirements described here and the ISO C standard is unintentional. This  
11506           volume of IEEE Std 1003.1-200x defers to the ISO C standard.

11507           The *ferror()* function shall test the error indicator for the stream pointed to by *stream*.11508 **RETURN VALUE**11509           The *ferror()* function shall return non-zero if and only if the error indicator is set for *stream*.11510 **ERRORS**

11511           No errors are defined.

11512 **EXAMPLES**

11513           None.

11514 **APPLICATION USAGE**

11515           None.

11516 **RATIONALE**

11517           None.

11518 **FUTURE DIRECTIONS**

11519           None.

11520 **SEE ALSO**11521           *clearerr()*, *feof()*, *fopen()*, the Base Definitions volume of IEEE Std 1003.1-200x, <stdio.h>11522 **CHANGE HISTORY**

11523           First released in Issue 1. Derived from Issue 1 of the SVID.

11524 **NAME**

11525       fesetenv — set current floating-point environment

11526 **SYNOPSIS**

11527       #include <fenv.h>

11528       int fesetenv(const fenv\_t \*envp);

11529 **DESCRIPTION**

11530       Refer to *fegetenv()*.

11531 **NAME**

11532       fesetexceptflag — set floating-point status flags

11533 **SYNOPSIS**

11534       #include &lt;fenv.h&gt;

11535       int fesetexceptflag(const fexcept\_t \*flagp, int excepts);

11536 **DESCRIPTION**11537       Refer to *fegetexceptflag()*.

11538 **NAME**

11539       fesetround — set current rounding direction

11540 **SYNOPSIS**

11541       #include <fenv.h>

11542       int fesetround(int *round*);

11543 **DESCRIPTION**

11544       Refer to *fegetround()*.

11545 **NAME**

11546       fetestexcept — test floating-point exception flags

11547 **SYNOPSIS**

11548       #include &lt;fenv.h&gt;

11549       int fetestexcept(int *excepts*);11550 **DESCRIPTION**

11551 *cx*     The functionality described on this reference page is aligned with the ISO C standard. Any  
 11552 conflict between the requirements described here and the ISO C standard is unintentional. This  
 11553 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

11554     The *fetestexcept()* function shall determine which of a specified subset of the floating-point  
 11555 exception flags are currently set. The *excepts* argument specifies the floating-point status flags to  
 11556 be queried.

11557 **RETURN VALUE**

11558     The *fetestexcept()* function shall return the value of the bitwise-inclusive OR of the floating-point  
 11559 exception macros corresponding to the currently set floating-point exceptions included in  
 11560 *excepts*.

11561 **ERRORS**

11562     No errors are defined.

11563 **EXAMPLES**

11564     The following example calls function *f()* if an invalid exception is set, and then function *g()* if an  
 11565 overflow exception is set:

```

11566     #include <fenv.h>
11567     /* ... */
11568     {
11569         #pragma STDC FENV_ACCESS ON
11570         int set_excepts;
11571         feclearexcept(FE_INVALID | FE_OVERFLOW);
11572         // maybe raise exceptions
11573         set_excepts = fetestexcept(FE_INVALID | FE_OVERFLOW);
11574         if (set_excepts & FE_INVALID) f();
11575         if (set_excepts & FE_OVERFLOW) g();
11576         /* ... */
11577     }

```

11578 **APPLICATION USAGE**

11579     None.

11580 **RATIONALE**

11581     None.

11582 **FUTURE DIRECTIONS**

11583     None.

11584 **SEE ALSO**

11585     *feclearexcept()*, *fegetexceptflag()*, *feraiseexcept()*, the Base Definitions volume of  
 11586 IEEE Std 1003.1-200x, <fenv.h>

11587 **CHANGE HISTORY**

11588 First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

11589 **NAME**

11590 feupdateenv — update floating-point environment

11591 **SYNOPSIS**

11592 #include &lt;fenv.h&gt;

11593 int feupdateenv(const fenv\_t \*envp);

11594 **DESCRIPTION**

11595 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
 11596 conflict between the requirements described here and the ISO C standard is unintentional. This  
 11597 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

11598 The *feupdateenv()* function shall attempt to save the currently raised floating-point exceptions in  
 11599 its automatic storage, attempt to install the floating-point environment represented by the object  
 11600 pointed to by *envp*, and then attempt to raise the saved floating-point exceptions. The argument  
 11601 *envp* shall point to an object set by a call to *feholdexcept()* or *fegetenv()*, or equal a floating-point  
 11602 environment macro.

11603 **RETURN VALUE**

11604 The *feupdateenv()* function shall return a zero value if and only if all the required actions were  
 11605 successfully carried out.

11606 **ERRORS**

11607 No errors are defined.

11608 **EXAMPLES**

11609 The following example shows sample code to hide spurious underflow floating-point  
 11610 exceptions:

```
11611 #include <fenv.h>
11612 double f(double x)
11613 {
11614     #pragma STDC FENV_ACCESS ON
11615     double result;
11616     fenv_t save_env;
11617     feholdexcept(&save_env);
11618     // compute result
11619     if (/* test spurious underflow */)
11620         feclearexcept(FE_UNDERFLOW);
11621     feupdateenv(&save_env);
11622     return result;
11623 }
```

11624 **APPLICATION USAGE**

11625 None.

11626 **RATIONALE**

11627 None.

11628 **FUTURE DIRECTIONS**

11629 None.

11630 **SEE ALSO**11631 *fegetenv()*, *feholdexcept()*, the Base Definitions volume of IEEE Std 1003.1-200x, <fenv.h>

11632 **CHANGE HISTORY**

- 11633       First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard. |
- 11634       ISO/IEC 9899: 1999 standard, Technical Corrigendum No. 1 is incorporated. |

11635 **NAME**

11636 fflush — flush a stream

11637 **SYNOPSIS**

11638 #include &lt;stdio.h&gt;

11639 int fflush(FILE \*stream);

11640 **DESCRIPTION**

11641 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 11642 conflict between the requirements described here and the ISO C standard is unintentional. This  
 11643 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

11644 If *stream* points to an output stream or an update stream in which the most recent operation was  
 11645 CX not input, *fflush()* shall cause any unwritten data for that stream to be written to the file, and the  
 11646 *st\_ctime* and *st\_mtime* fields of the underlying file shall be marked for update.

11647 If *stream* is a null pointer, *fflush()* shall perform this flushing action on all streams for which the  
 11648 behavior is defined above.

11649 **RETURN VALUE**

11650 Upon successful completion, *fflush()* shall return 0; otherwise, it shall set the error indicator for  
 11651 CX the stream, return EOF, and set *errno* to indicate the error.

11652 **ERRORS**11653 The *fflush()* function shall fail if:

11654 CX [EAGAIN] The O\_NONBLOCK flag is set for the file descriptor underlying *stream* and the  
 11655 process would be delayed in the write operation.

11656 CX [EBADF] The file descriptor underlying *stream* is not valid.

11657 CX [EFBIG] An attempt was made to write a file that exceeds the maximum file size.

11658 XSI [EFBIG] An attempt was made to write a file that exceeds the process' file size limit.

11659 CX [EFBIG] The file is a regular file and an attempt was made to write at or beyond the  
 11660 offset maximum associated with the corresponding stream.

11661 CX [EINTR] The *fflush()* function was interrupted by a signal.

11662 CX [EIO] The process is a member of a background process group attempting to write  
 11663 to its controlling terminal, TOSTOP is set, the process is neither ignoring nor  
 11664 blocking SIGTTOU, and the process group of the process is orphaned. This  
 11665 error may also be returned under implementation-defined conditions.

11666 CX [ENOSPC] There was no free space remaining on the device containing the file.

11667 CX [EPIPE] An attempt is made to write to a pipe or FIFO that is not open for reading by  
 11668 any process. A SIGPIPE signal shall also be sent to the thread.

11669 The *fflush()* function may fail if:

11670 CX [ENXIO] A request was made of a nonexistent device, or the request was outside the  
 11671 capabilities of the device.

11672 **EXAMPLES**11673 **Sending Prompts to Standard Output**

11674 The following example uses *printf()* calls to print a series of prompts for information the user  
11675 must enter from standard input. The *flush()* calls force the output to standard output. The  
11676 *fflush()* function is used because standard output is usually buffered and the prompt may not  
11677 immediately be printed on the output or terminal. The *gets()* calls read strings from standard  
11678 input and place the results in variables, for use later in the program

```
11679 #include <stdio.h>
11680 ...
11681 char user[100];
11682 char oldpasswd[100];
11683 char newpasswd[100];
11684 ...
11685 printf("User name: ");
11686 fflush(stdout);
11687 gets(user);

11688 printf("Old password: ");
11689 fflush(stdout);
11690 gets(oldpasswd);

11691 printf("New password: ");
11692 fflush(stdout);
11693 gets(newpasswd);
11694 ...
```

11695 **APPLICATION USAGE**

11696 None.

11697 **RATIONALE**

11698 Data buffered by the system may make determining the validity of the position of the current  
11699 file descriptor impractical. Thus, enforcing the repositioning of the file descriptor after *fflush()*  
11700 on streams open for *read()* is not mandated by IEEE Std 1003.1-200x.

11701 **FUTURE DIRECTIONS**

11702 None.

11703 **SEE ALSO**

11704 *getrlimit()*, *ulimit()*, the Base Definitions volume of IEEE Std 1003.1-200x, **<stdio.h>**

11705 **CHANGE HISTORY**

11706 First released in Issue 1. Derived from Issue 1 of the SVID.

11707 **Issue 5**

11708 Large File Summit extensions are added.

11709 **Issue 6**

11710 Extensions beyond the ISO C standard are now marked.

11711 The following new requirements on POSIX implementations derive from alignment with the  
11712 Single UNIX Specification:

- 11713 • The [EFBIG] error is added as part of the large file support extensions.
- 11714 • The [ENXIO] optional error condition is added.

11715 The RETURN VALUE section is updated to note that the error indicator shall be set for the  
11716 stream. This is for alignment with the ISO/IEC 9899:1999 standard.

11717 **NAME**

11718       ffs — find first set bit

11719 **SYNOPSIS**

11720 xSI       #include &lt;strings.h&gt;

11721       int ffs(int i);

11722

11723 **DESCRIPTION**11724       The *ffs()* function shall find the first bit set (beginning with the least significant bit) in *i*, and  
11725       return the index of that bit. Bits are numbered starting at one (the least significant bit).11726 **RETURN VALUE**11727       The *ffs()* function shall return the index of the first bit set. If *i* is 0, then *ffs()* shall return 0.11728 **ERRORS**

11729       No errors are defined.

11730 **EXAMPLES**

11731       None.

11732 **APPLICATION USAGE**

11733       None.

11734 **RATIONALE**

11735       None.

11736 **FUTURE DIRECTIONS**

11737       None.

11738 **SEE ALSO**

11739       The Base Definitions volume of IEEE Std 1003.1-200x, &lt;strings.h&gt;

11740 **CHANGE HISTORY**

11741       First released in Issue 4, Version 2.

11742 **Issue 5**

11743       Moved from X/OPEN UNIX extension to BASE.

## 11744 NAME

11745 fgetc — get a byte from a stream

## 11746 SYNOPSIS

11747 #include <stdio.h>

11748 int fgetc(FILE \*stream);

## 11749 DESCRIPTION

11750 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
11751 conflict between the requirements described here and the ISO C standard is unintentional. This  
11752 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

11753 If the end-of-file indicator for the input stream pointed to by *stream* is not set and a next byte is  
11754 present, the *fgetc()* function shall obtain the next byte as an **unsigned char** converted to an **int**,  
11755 from the input stream pointed to by *stream*, and advance the associated file position indicator for  
11756 the stream (if defined). Since *fgetc()* operates on bytes, reading a character consisting of multiple  
11757 bytes (or “a multi-byte character”) may require multiple calls to *fgetc()*.

11758 CX The *fgetc()* function may mark the *st\_atime* field of the file associated with *stream* for update. The  
11759 *st\_atime* field shall be marked for update by the first successful execution of *fgetc()*, *fgets()*,  
11760 *fgetwc()*, *fgetws()*, *fread()*, *fscanf()*, *getc()*, *getchar()*, *gets()*, or *scanf()* using *stream* that returns  
11761 data not supplied by a prior call to *ungetc()* or *ungetwc()*.

## 11762 RETURN VALUE

11763 Upon successful completion, *fgetc()* shall return the next byte from the input stream pointed to  
11764 by *stream*. If the end-of-file indicator for the stream is set, or if the stream is at end-of-file, the  
11765 end-of-file indicator for the stream shall be set and *fgetc()* shall return EOF. If a read error occurs,  
11766 CX the error indicator for the stream shall be set, *fgetc()* shall return EOF, and shall set *errno* to  
11767 indicate the error.

## 11768 ERRORS

11769 The *fgetc()* function shall fail if data needs to be read and:

11770 CX [EAGAIN] The O\_NONBLOCK flag is set for the file descriptor underlying *stream* and the  
11771 process would be delayed in the *fgetc()* operation.

11772 CX [EBADF] The file descriptor underlying *stream* is not a valid file descriptor open for  
11773 reading.

11774 CX [EINTR] The read operation was terminated due to the receipt of a signal, and no data  
11775 was transferred.

11776 CX [EIO] A physical I/O error has occurred, or the process is in a background process  
11777 group attempting to read from its controlling terminal, and either the process  
11778 is ignoring or blocking the SIGTTIN signal or the process group is orphaned.  
11779 This error may also be generated for implementation-defined reasons.

11780 CX [E\_OVERFLOW] The file is a regular file and an attempt was made to read at or beyond the  
11781 offset maximum associated with the corresponding stream.

11782 The *fgetc()* function may fail if:

11783 CX [ENOMEM] Insufficient storage space is available.

11784 CX [ENXIO] A request was made of a nonexistent device, or the request was outside the  
11785 capabilities of the device.

11786 **EXAMPLES**

11787           None.

11788 **APPLICATION USAGE**

11789           If the integer value returned by *fgetc()* is stored into a variable of type **char** and then compared  
11790           against the integer constant EOF, the comparison may never succeed, because sign-extension of  
11791           a variable of type **char** on widening to integer is implementation-defined.

11792           The *ferror()* or *feof()* functions must be used to distinguish between an error condition and an  
11793           end-of-file condition.

11794 **RATIONALE**

11795           None.

11796 **FUTURE DIRECTIONS**

11797           None.

11798 **SEE ALSO**

11799           *feof()*, *ferror()*, *fopen()*, *getchar()*, *getc()*, the Base Definitions volume of IEEE Std 1003.1-200x,  
11800           <**stdio.h**>

11801 **CHANGE HISTORY**

11802           First released in Issue 1. Derived from Issue 1 of the SVID.

11803 **Issue 5**

11804           Large File Summit extensions are added.

11805 **Issue 6**

11806           Extensions beyond the ISO C standard are now marked.

11807           The following new requirements on POSIX implementations derive from alignment with the  
11808           Single UNIX Specification:

- 11809           • The [EIO] and [EOVERFLOW] mandatory error conditions are added.
- 11810           • The [ENOMEM] and [ENXIO] optional error conditions are added.

11811           The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 11812           • The DESCRIPTION is updated to clarify the behavior when the end-of-file indicator for the  
11813           input stream is not set.
- 11814           • The RETURN VALUE section is updated to note that the error indicator shall be set for the  
11815           stream.

11816 **NAME**

11817 `fgetpos` — get current file position information

11818 **SYNOPSIS**

11819 `#include <stdio.h>`

11820 `int fgetpos(FILE *restrict stream, fpos_t *restrict pos);`

11821 **DESCRIPTION**

11822 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
11823 conflict between the requirements described here and the ISO C standard is unintentional. This  
11824 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

11825 The `fgetpos()` function shall store the current values of the parse state (if any) and file position  
11826 indicator for the stream pointed to by `stream` in the object pointed to by `pos`. The value stored  
11827 contains unspecified information usable by `fsetpos()` for repositioning the stream to its position  
11828 at the time of the call to `fgetpos()`.

11829 **RETURN VALUE**

11830 Upon successful completion, `fgetpos()` shall return 0; otherwise, it shall return a non-zero value  
11831 and set `errno` to indicate the error.

11832 **ERRORS**

11833 The `fgetpos()` function shall fail if:

11834 CX [EOVERFLOW] The current value of the file position cannot be represented correctly in an  
11835 object of type `fpos_t`.

11836 The `fgetpos()` function may fail if:

11837 CX [EBADF] The file descriptor underlying `stream` is not valid.

11838 CX [ESPIPE] The file descriptor underlying `stream` is associated with a pipe, FIFO, or socket.  
11839

11840 **EXAMPLES**

11841 None.

11842 **APPLICATION USAGE**

11843 None.

11844 **RATIONALE**

11845 None.

11846 **FUTURE DIRECTIONS**

11847 None.

11848 **SEE ALSO**

11849 `fopen()`, `ftell()`, `rewind()`, `ungetc()`, the Base Definitions volume of IEEE Std 1003.1-200x, `<stdio.h>`

11850 **CHANGE HISTORY**

11851 First released in Issue 4. Derived from the ISO C standard.

11852 **Issue 5**

11853 Large File Summit extensions are added.

11854 **Issue 6**

11855 Extensions beyond the ISO C standard are now marked.

11856 The following new requirements on POSIX implementations derive from alignment with the  
11857 Single UNIX Specification:

- 11858       • The [EIO] mandatory error condition is added.
- 11859       • The [EBADF] and [ESPIPE] optional error conditions are added.
- 11860       An additional [ESPIPE] error condition is added for sockets.
- 11861       The prototype for *fgetpos()* is changed for alignment with the ISO/IEC 9899:1999 standard.

11862 **NAME**

11863        fgets — get a string from a stream

11864 **SYNOPSIS**

11865        #include &lt;stdio.h&gt;

11866        char \*fgets(char \*restrict *s*, int *n*, FILE \*restrict *stream*);11867 **DESCRIPTION**

11868 cx     The functionality described on this reference page is aligned with the ISO C standard. Any  
 11869     conflict between the requirements described here and the ISO C standard is unintentional. This  
 11870     volume of IEEE Std 1003.1-200x defers to the ISO C standard.

11871     The *fgets()* function shall read bytes from *stream* into the array pointed to by *s*, until *n*−1 bytes  
 11872     are read, or a <newline> is read and transferred to *s*, or an end-of-file condition is encountered.  
 11873     The string is then terminated with a null byte.

11874 cx     The *fgets()* function may mark the *st\_atime* field of the file associated with *stream* for update. The  
 11875     *st\_atime* field shall be marked for update by the first successful execution of *fgetc()*, *fgets()*,  
 11876     *fgetwc()*, *fgetws()*, *fread()*, *fscanf()*, *getc()*, *getchar()*, *gets()*, or *scanf()* using *stream* that returns  
 11877     data not supplied by a prior call to *ungetc()* or *ungetwc()*.

11878 **RETURN VALUE**

11879     Upon successful completion, *fgets()* shall return *s*. If the stream is at end-of-file, the end-of-file  
 11880     indicator for the stream shall be set and *fgets()* shall return a null pointer. If a read error occurs,  
 11881 cx     the error indicator for the stream shall be set, *fgets()* shall return a null pointer, and shall set  
 11882     *errno* to indicate the error.

11883 **ERRORS**11884        Refer to *fgetc()*.11885 **EXAMPLES**11886        **Reading Input**

11887     The following example uses *fgets()* to read each line of input. {LINE\_MAX}, which defines the  
 11888     maximum size of the input line, is defined in the <limits.h> header.

```

11889     #include <stdio.h>
11890     ...
11891     char line[LINE_MAX];
11892     ...
11893     while (fgets(line, LINE_MAX, fp) != NULL) {
11894         ...
11895     }
11896     ...
```

11897 **APPLICATION USAGE**

11898        None.

11899 **RATIONALE**

11900        None.

11901 **FUTURE DIRECTIONS**

11902        None.

11903 **SEE ALSO**

11904 *fopen()*, *fread()*, *gets()*, the Base Definitions volume of IEEE Std 1003.1-200x, <**stdio.h**>

11905 **CHANGE HISTORY**

11906 First released in Issue 1. Derived from Issue 1 of the SVID.

11907 **Issue 6**

11908 Extensions beyond the ISO C standard are now marked.

11909 The prototype for *fgets()* is changed for alignment with the ISO/IEC 9899:1999 standard.

11910 **NAME**11911 `fgetwc` — get a wide-character code from a stream11912 **SYNOPSIS**11913 `#include <stdio.h>`11914 `#include <wchar.h>`11915 `wint_t fgetwc(FILE *stream);`11916 **DESCRIPTION**

11917 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 11918 conflict between the requirements described here and the ISO C standard is unintentional. This  
 11919 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

11920 The `fgetwc()` function shall obtain the next character (if present) from the input stream pointed to  
 11921 by `stream`, convert that to the corresponding wide-character code, and advance the associated  
 11922 file position indicator for the stream (if defined).

11923 If an error occurs, the resulting value of the file position indicator for the stream is unspecified. |

11924 CX The `fgetwc()` function may mark the `st_atime` field of the file associated with `stream` for update.  
 11925 The `st_atime` field shall be marked for update by the first successful execution of `fgetc()`, `fgets()`,  
 11926 `fgetwc()`, `fgetws()`, `fread()`, `fscanf()`, `getc()`, `getchar()`, `gets()`, or `scanf()` using `stream` that returns  
 11927 data not supplied by a prior call to `ungetc()` or `ungetwc()`.

11928 **RETURN VALUE**

11929 Upon successful completion, the `fgetwc()` function shall return the wide-character code of the  
 11930 character read from the input stream pointed to by `stream` converted to a type `wint_t`. If the  
 11931 stream is at end-of-file, the end-of-file indicator for the stream shall be set and `fgetwc()` shall  
 11932 return WEOF. If a read error occurs, the error indicator for the stream shall be set, `fgetwc()` shall  
 11933 CX return WEOF, and shall set `errno` to indicate the error. If an encoding error occurs, the error |  
 11934 indicator for the stream shall be set, `fgetwc()` shall return WEOF, and shall set `errno` to indicate |  
 11935 the error. |

11936 **ERRORS**11937 The `fgetwc()` function shall fail if data needs to be read and:

11938 CX [EAGAIN] The O\_NONBLOCK flag is set for the file descriptor underlying `stream` and the  
 11939 process would be delayed in the `fgetwc()` operation.

11940 CX [EBADF] The file descriptor underlying `stream` is not a valid file descriptor open for  
 11941 reading. |

11942 [EILSEQ] The data obtained from the input stream does not form a valid character. |

11943 CX [EINTR] The read operation was terminated due to the receipt of a signal, and no data  
 11944 was transferred.

11945 CX [EIO] A physical I/O error has occurred, or the process is in a background process  
 11946 group attempting to read from its controlling terminal, and either the process  
 11947 is ignoring or blocking the SIGTTIN signal or the process group is orphaned.  
 11948 This error may also be generated for implementation-defined reasons.

11949 CX [EOVERFLOW] The file is a regular file and an attempt was made to read at or beyond the  
 11950 offset maximum associated with the corresponding stream.

11951 The `fgetwc()` function may fail if:

11952 CX [ENOMEM] Insufficient storage space is available.

11953 CX [ENXIO] A request was made of a nonexistent device, or the request was outside the  
11954 capabilities of the device.

#### 11955 EXAMPLES

11956 None.

#### 11957 APPLICATION USAGE

11958 The *ferror()* or *feof()* functions must be used to distinguish between an error condition and an  
11959 end-of-file condition.

#### 11960 RATIONALE

11961 None.

#### 11962 FUTURE DIRECTIONS

11963 None.

#### 11964 SEE ALSO

11965 *feof()*, *ferror()*, *fopen()*, the Base Definitions volume of IEEE Std 1003.1-200x, `<stdio.h>`,  
11966 `<wchar.h>`

#### 11967 CHANGE HISTORY

11968 First released in Issue 4. Derived from the MSE working draft.

#### 11969 Issue 5

11970 The Optional Header (OH) marking is removed from `<stdio.h>`.

11971 Large File Summit extensions are added.

#### 11972 Issue 6

11973 Extensions beyond the ISO C standard are now marked.

11974 The following new requirements on POSIX implementations derive from alignment with the  
11975 Single UNIX Specification:

- 11976 • The [EIO] and [EOVERFLOW] mandatory error conditions are added.
- 11977 • The [ENOMEM] and [ENXIO] optional error conditions are added.

11978 **NAME**11979 `fgetws` — get a wide-character string from a stream11980 **SYNOPSIS**11981 `#include <stdio.h>`11982 `#include <wchar.h>`11983 `wchar_t *fgetws(wchar_t *restrict ws, int n,`11984 `FILE *restrict stream);`11985 **DESCRIPTION**

11986 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 11987 conflict between the requirements described here and the ISO C standard is unintentional. This  
 11988 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

11989 The `fgetws()` function shall read characters from the *stream*, convert these to the corresponding  
 11990 wide-character codes, place them in the `wchar_t` array pointed to by *ws*, until *n*−1 characters are  
 11991 read, or a <newline> is read, converted, and transferred to *ws*, or an end-of-file condition is  
 11992 encountered. The wide-character string, *ws*, shall then be terminated with a null wide-character  
 11993 code.

11994 If an error occurs, the resulting value of the file position indicator for the stream is unspecified.

11995 **CX** The `fgetws()` function may mark the `st_atime` field of the file associated with *stream* for update.  
 11996 The `st_atime` field shall be marked for update by the first successful execution of `fgetc()`, `fgets()`,  
 11997 `fgetwc()`, `fgetws()`, `fread()`, `fscanf()`, `getc()`, `getchar()`, `gets()`, or `scanf()` using *stream* that returns  
 11998 data not supplied by a prior call to `ungetc()` or `ungetwc()`.

11999 **RETURN VALUE**

12000 Upon successful completion, `fgetws()` shall return *ws*. If the stream is at end-of-file, the end-of-  
 12001 file indicator for the stream shall be set and `fgetws()` shall return a null pointer. If a read error  
 12002 **CX** occurs, the error indicator for the stream shall be set, `fgetws()` shall return a null pointer, and  
 12003 shall set `errno` to indicate the error.

12004 **ERRORS**12005 Refer to `fgetwc()`.12006 **EXAMPLES**

12007 None.

12008 **APPLICATION USAGE**

12009 None.

12010 **RATIONALE**

12011 None.

12012 **FUTURE DIRECTIONS**

12013 None.

12014 **SEE ALSO**12015 `fopen()`, `fread()`, the Base Definitions volume of IEEE Std 1003.1-200x, `<stdio.h>`, `<wchar.h>`12016 **CHANGE HISTORY**

12017 First released in Issue 4. Derived from the MSE working draft.

12018 **Issue 5**12019 The Optional Header (OH) marking is removed from `<stdio.h>`.

12020 **Issue 6**

12021 Extensions beyond the ISO C standard are now marked.

12022 The prototype for *fgetws()* is changed for alignment with the ISO/IEC 9899:1999 standard.

12023 **NAME**

12024           fileno — map a stream pointer to a file descriptor

12025 **SYNOPSIS**

12026 **cx**       #include <stdio.h>

12027           int fileno(FILE \*stream);

12028

12029 **DESCRIPTION**

12030           The *fileno()* function shall return the integer file descriptor associated with the stream pointed to  
12031           by *stream*.

12032 **RETURN VALUE**

12033           Upon successful completion, *fileno()* shall return the integer value of the file descriptor  
12034           associated with *stream*. Otherwise, the value  $-1$  shall be returned and *errno* set to indicate the  
12035           error.

12036 **ERRORS**

12037           The *fileno()* function may fail if:

12038           [EBADF]           The *stream* argument is not a valid stream.

12039 **EXAMPLES**

12040           None.

12041 **APPLICATION USAGE**

12042           None.

12043 **RATIONALE**

12044           Without some specification of which file descriptors are associated with these streams, it is  
12045           impossible for an application to set up the streams for another application it starts with *fork()*  
12046           and *exec*. In particular, it would not be possible to write a portable version of the *sh* command  
12047           interpreter (although there may be other constraints that would prevent that portability).

12048 **FUTURE DIRECTIONS**

12049           None.

12050 **SEE ALSO**

12051           *fdopen()*, *fopen()*, *stdin*, the Base Definitions volume of IEEE Std 1003.1-200x, <stdio.h>, Section  
12052           2.5.1 (on page 485)

12053 **CHANGE HISTORY**

12054           First released in Issue 1. Derived from Issue 1 of the SVID.

12055 **Issue 6**

12056           The following new requirements on POSIX implementations derive from alignment with the  
12057           Single UNIX Specification:

- 12058           • The [EBADF] optional error condition is added.

12059 **NAME**

12060 flockfile, ftrylockfile, funlockfile — stdio locking functions

12061 **SYNOPSIS**

12062 TSF #include &lt;stdio.h&gt;

```
12063 void flockfile(FILE *file);
12064 int ftrylockfile(FILE *file);
12065 void funlockfile(FILE *file);
12066
```

12067 **DESCRIPTION**

12068 These functions shall provide for explicit application-level locking of stdio (**FILE \***) objects. |  
12069 These functions can be used by a thread to delineate a sequence of I/O statements that are |  
12070 executed as a unit.

12071 The *flockfile()* function shall acquire for a thread ownership of a (**FILE \***) object. |

12072 The *ftrylockfile()* function shall acquire for a thread ownership of a (**FILE \***) object if the object is |  
12073 available; *ftrylockfile()* is a non-blocking version of *flockfile()*.

12074 The *funlockfile()* function shall relinquish the ownership granted to the thread. The behavior is |  
12075 undefined if a thread other than the current owner calls the *funlockfile()* function.

12076 The functions shall behave as if there is a lock count associated with each (**FILE \***) object. This |  
12077 count is implicitly initialized to zero when the (**FILE \***) object is created. The (**FILE \***) object is |  
12078 unlocked when the count is zero. When the count is positive, a single thread owns the (**FILE \***) |  
12079 object. When the *flockfile()* function is called, if the count is zero or if the count is positive and |  
12080 the caller owns the (**FILE \***) object, the count shall be incremented. Otherwise, the calling thread |  
12081 shall be suspended, waiting for the count to return to zero. Each call to *funlockfile()* shall |  
12082 decrement the count. This allows matching calls to *flockfile()* (or successful calls to *ftrylockfile()*) |  
12083 and *funlockfile()* to be nested.

12084 All functions that reference (**FILE \***) objects shall behave as if they use *flockfile()* and *funlockfile()* |  
12085 internally to obtain ownership of these (**FILE \***) objects.

12086 **RETURN VALUE**

12087 None for *flockfile()* and *funlockfile()*. The *ftrylockfile()* function shall return zero for success and |  
12088 non-zero to indicate that the lock cannot be acquired.

12089 **ERRORS**

12090 No errors are defined.

12091 **EXAMPLES**

12092 None.

12093 **APPLICATION USAGE**

12094 Applications using these functions may be subject to priority inversion, as discussed in the Base |  
12095 Definitions volume of IEEE Std 1003.1-200x, Section 3.285, Priority Inversion.

12096 **RATIONALE**

12097 The *flockfile()* and *funlockfile()* functions provide an orthogonal mutual exclusion lock for each |  
12098 **FILE**. The *ftrylockfile()* function provides a non-blocking attempt to acquire a file lock, |  
12099 analogous to *pthread\_mutex\_trylock()*.

12100 These locks behave as if they are the same as those used internally by *stdio* for thread-safety. |  
12101 This both provides thread-safety of these functions without requiring a second level of internal |  
12102 locking and allows functions in *stdio* to be implemented in terms of other *stdio* functions.

12103 Application writers and implementors should be aware that there are potential deadlock  
12104 problems on **FILE** objects. For example, the line-buffered flushing semantics of *stdio* (requested  
12105 via `{_IOLBF}`) require that certain input operations sometimes cause the buffered contents of  
12106 implementation-defined line-buffered output streams to be flushed. If two threads each hold the  
12107 lock on the other's **FILE**, deadlock ensues. This type of deadlock can be avoided by acquiring  
12108 **FILE** locks in a consistent order. In particular, the line-buffered output stream deadlock can  
12109 typically be avoided by acquiring locks on input streams before locks on output streams if a  
12110 thread would be acquiring both.

12111 In summary, threads sharing *stdio* streams with other threads can use *flockfile()* and *funlockfile()*  
12112 to cause sequences of I/O performed by a single thread to be kept bundled. The only case where  
12113 the use of *flockfile()* and *funlockfile()* is required is to provide a scope protecting uses of the  
12114 `*_unlocked()` functions/macros. This moves the cost/performance tradeoff to the optimal point.

#### 12115 **FUTURE DIRECTIONS**

12116 None.

#### 12117 **SEE ALSO**

12118 *getc\_unlocked()*, *putc\_unlocked()*, the Base Definitions volume of IEEE Std 1003.1-200x, `<stdio.h>`

#### 12119 **CHANGE HISTORY**

12120 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

#### 12121 **Issue 6**

12122 These functions are marked as part of the Thread-Safe Functions option.

## 12123 NAME

12124 floor, floorf, floorl — floor function

## 12125 SYNOPSIS

12126 #include &lt;math.h&gt;

12127 double floor(double x);

12128 float floorf(float x);

12129 long double floorl(long double x);

## 12130 DESCRIPTION

12131 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 12132 conflict between the requirements described here and the ISO C standard is unintentional. This  
 12133 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

12134 These functions shall compute the largest integral value not greater than *x*.

12135 An application wishing to check for error situations should set *errno* to zero and call  
 12136 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 12137 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 12138 zero, an error has occurred.

## 12139 RETURN VALUE

12140 Upon successful completion, these functions shall return the largest integral value not greater  
 12141 than *x*, expressed as a **double**, **float**, or **long double**, as appropriate for the return type of the  
 12142 function.

12143 **MX** If *x* is NaN, a NaN shall be returned.12144 If *x* is  $\pm 0$  or  $\pm \text{Inf}$ , *x* shall be returned.

12145 **XSI** If the correct value would cause overflow, a range error shall occur and *floor*(*x*), *floorf*(*x*), and  
 12146 *floorl*(*x*) shall return the value of the macro HUGE\_VAL, HUGE\_VALF, and HUGE\_VALL,  
 12147 respectively.

## 12148 ERRORS

12149 These functions shall fail if:

12150 **XSI** Range Error The result would cause an overflow.

12151 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero, |  
 12152 then *errno* shall be set to [ERANGE]. If the integer expression |  
 12153 (math\_errhandling & MATH\_ERREXCEPT) is non-zero, then the overflow |  
 12154 floating-point exception shall be raised. |

## 12155 EXAMPLES

12156 None.

## 12157 APPLICATION USAGE

12158 The integral value returned by these functions might not be expressible as an **int** or **long**. The  
 12159 return value should be tested before assigning it to an integer type to avoid the undefined results  
 12160 of an integer overflow.

12161 The *floor*(*x*) function can only overflow when the floating-point representation has  
 12162 DBL\_MANT\_DIG > DBL\_MAX\_EXP.

12163 On error, the expressions (math\_errhandling & MATH\_ERRNO) and (math\_errhandling &  
 12164 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

12165 **RATIONALE**

12166 None.

12167 **FUTURE DIRECTIONS**

12168 None.

12169 **SEE ALSO**12170 *ceil()*, *feclearexcept()*, *fetestexcept()*, *isnan()*, the Base Definitions volume of IEEE Std 1003.1-200x, |

12171 Section 4.18, Treatment of Error Conditions for Mathematical Functions, &lt;math.h&gt; |

12172 **CHANGE HISTORY**

12173 First released in Issue 1. Derived from Issue 1 of the SVID.

12174 **Issue 5**12175 The DESCRIPTION is updated to indicate how an application should check for an error. This  
12176 text was previously published in the APPLICATION USAGE section.12177 **Issue 6**12178 The *floorf()* and *floorl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.12179 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
12180 revised to align with the ISO/IEC 9899:1999 standard.12181 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
12182 marked.

## 12183 NAME

12184 fma, fmaf, fmal — floating-point multiply-add

## 12185 SYNOPSIS

12186 #include &lt;math.h&gt;

12187 double fma(double x, double y, double z);

12188 float fmaf(float x, float y, float z);

12189 long double fmal(long double x, long double y, long double z);

## 12190 DESCRIPTION

12191 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 12192 conflict between the requirements described here and the ISO C standard is unintentional. This  
 12193 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

12194 These functions shall compute  $(x * y) + z$ , rounded as one ternary operation: they shall compute  
 12195 the value (as if) to infinite precision and round once to the result format, according to the  
 12196 rounding mode characterized by the value of FLT\_ROUNDS.

12197 An application wishing to check for error situations should set *errno* to zero and call  
 12198 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 12199 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 12200 zero, an error has occurred.

## 12201 RETURN VALUE

12202 Upon successful completion, these functions shall return  $(x * y) + z$ , rounded as one ternary  
 12203 operation.

12204 **MX** If *x* or *y* are NaN, a NaN shall be returned.

12205 If *x* multiplied by *y* is an exact infinity and *z* is also an infinity but with the opposite sign, a  
 12206 domain error shall occur, and either a NaN (if supported), or an implementation-defined value  
 12207 shall be returned.

12208 If one of *x* and *y* is infinite, the other is zero, and *z* is not a NaN, a domain error shall occur, and  
 12209 either a NaN (if supported), or an implementation-defined value shall be returned.

12210 If one of *x* and *y* is infinite, the other is zero, and *z* is a NaN, a NaN shall be returned and a  
 12211 domain error may occur.

12212 If  $x*y$  is not  $0*Inf$  nor  $Inf*0$  and *z* is a NaN, a NaN shall be returned.

## 12213 ERRORS

12214 These functions shall fail if:

12215 **MX** Domain Error The value of  $x*y+z$  is invalid, or the value  $x*y$  is invalid and *z* is not a NaN.

12216 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero, |  
 12217 then *errno* shall be set to [EDOM]. If the integer expression (math\_errhandling |  
 12218 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception |  
 12219 shall be raised. |

12220 **MX** Range Error The result overflows.

12221 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero, |  
 12222 then *errno* shall be set to [ERANGE]. If the integer expression |  
 12223 (math\_errhandling & MATH\_ERREXCEPT) is non-zero, then the overflow |  
 12224 floating-point exception shall be raised. |

12225 These functions may fail if:

12226	MX	<b>Domain Error</b>	The value $x*y$ is invalid and $z$ is a NaN.
12227			If the integer expression ( <code>math_errhandling &amp; MATH_ERRNO</code> ) is non-zero,
12228			then <i>errno</i> shall be set to [EDOM]. If the integer expression ( <code>math_errhandling</code>
12229			& <code>MATH_ERREXCEPT</code> ) is non-zero, then the invalid floating-point exception
12230			shall be raised.
12231	MX	<b>Range Error</b>	The result underflows.
12232			If the integer expression ( <code>math_errhandling &amp; MATH_ERRNO</code> ) is non-zero,
12233			then <i>errno</i> shall be set to [ERANGE]. If the integer expression
12234			( <code>math_errhandling &amp; MATH_ERREXCEPT</code> ) is non-zero, then the underflow
12235			floating-point exception shall be raised.
12236		<b>EXAMPLES</b>	
12237			None.
12238		<b>APPLICATION USAGE</b>	
12239			On error, the expressions ( <code>math_errhandling &amp; MATH_ERRNO</code> ) and ( <code>math_errhandling &amp;</code>
12240			<code>MATH_ERREXCEPT</code> ) are independent of each other, but at least one of them must be non-zero.
12241		<b>RATIONALE</b>	
12242			In many cases, clever use of floating ( <i>fused</i> ) multiply-add leads to much improved code; but its
12243			unexpected use by the compiler can undermine carefully written code. The <code>FP_CONTRACT</code>
12244			macro can be used to disallow use of floating multiply-add; and the <i>fma()</i> function guarantees
12245			its use where desired. Many current machines provide hardware floating multiply-add
12246			instructions; software implementation can be used for others.
12247		<b>FUTURE DIRECTIONS</b>	
12248			None.
12249		<b>SEE ALSO</b>	
12250			<i>feclearexcept()</i> , <i>fetestexcept()</i> , the Base Definitions volume of IEEE Std 1003.1-200x, Section 4.18,
12251			Treatment of Error Conditions for Mathematical Functions, < <b>math.h</b> >
12252		<b>CHANGE HISTORY</b>	
12253			First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

12254 **NAME**

12255 fmax, fmaxf, fmaxl — determine maximum numeric value of two floating-point numbers

12256 **SYNOPSIS**

12257 #include <math.h>

12258 double fmax(double x, double y);

12259 float fmaxf(float x, float y);

12260 long double fmaxl(long double x, long double y);

12261 **DESCRIPTION**

12262 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
12263 conflict between the requirements described here and the ISO C standard is unintentional. This  
12264 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

12265 These functions shall determine the maximum numeric value of their arguments. NaN  
12266 arguments shall be treated as missing data: if one argument is a NaN and the other numeric,  
12267 then these functions shall choose the numeric value.

12268 **RETURN VALUE**

12269 Upon successful completion, these functions shall return the maximum numeric value of their  
12270 arguments.

12271 If just one argument is a NaN, the other argument shall be returned.

12272 **MX** If *x* and *y* are NaN, a NaN shall be returned.

12273 **ERRORS**

12274 No errors are defined.

12275 **EXAMPLES**

12276 None.

12277 **APPLICATION USAGE**

12278 None.

12279 **RATIONALE**

12280 None.

12281 **FUTURE DIRECTIONS**

12282 None.

12283 **SEE ALSO**

12284 *fdim()*, *fmin()*, the Base Definitions volume of IEEE Std 1003.1-200x, <math.h>

12285 **CHANGE HISTORY**

12286 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

12287 **NAME**

12288       fmin, fminf, fminl — determine minimum numeric value of two floating-point numbers

12289 **SYNOPSIS**

12290       #include <math.h>

12291       double fmin(double x, double y);

12292       float fminf(float x, float y);

12293       long double fminl(long double x, long double y);

12294 **DESCRIPTION**

12295 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
12296       conflict between the requirements described here and the ISO C standard is unintentional. This  
12297       volume of IEEE Std 1003.1-200x defers to the ISO C standard.

12298       These functions shall determine the minimum numeric value of their arguments. NaN  
12299       arguments shall be treated as missing data: if one argument is a NaN and the other numeric,  
12300       then these functions shall choose the numeric value.

12301 **RETURN VALUE**

12302       Upon successful completion, these functions shall return the minimum numeric value of their  
12303       arguments.

12304       If just one argument is a NaN, the other argument shall be returned.

12305 **MX**       If *x* and *y* are NaN, a NaN shall be returned.

12306 **ERRORS**

12307       No errors are defined.

12308 **EXAMPLES**

12309       None.

12310 **APPLICATION USAGE**

12311       None.

12312 **RATIONALE**

12313       None.

12314 **FUTURE DIRECTIONS**

12315       None.

12316 **SEE ALSO**

12317       *fdim()*, *fmax()*, the Base Definitions volume of IEEE Std 1003.1-200x, <math.h>

12318 **CHANGE HISTORY**

12319       First released in Issue 6. Derived from ISO/IEC 9899:1999 standard.

## 12320 NAME

12321 fmod, fmodf, fmodl — floating-point remainder value function

## 12322 SYNOPSIS

12323 #include <math.h>

12324 double fmod(double x, double y);

12325 float fmodf(float x, float y);

12326 long double fmodl(long double x, long double y);

## 12327 DESCRIPTION

12328 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
12329 conflict between the requirements described here and the ISO C standard is unintentional. This  
12330 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

12331 These functions shall return the floating-point remainder of the division of  $x$  by  $y$ .

12332 An application wishing to check for error situations should set *errno* to zero and call  
12333 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
12334 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
12335 zero, an error has occurred.

## 12336 RETURN VALUE

12337 These functions shall return the value  $x-i*y$ , for some integer  $i$  such that, if  $y$  is non-zero, the  
12338 result has the same sign as  $x$  and magnitude less than the magnitude of  $y$ .

12339 If the correct value would cause underflow, and is not representable, a range error may occur,  
12340 MX and either 0.0 (if supported), or an implementation-defined value shall be returned.

12341 MX If  $x$  or  $y$  is NaN, a NaN shall be returned.

12342 If  $y$  is zero, a domain error shall occur, and either a NaN (if supported), or an implementation-  
12343 defined value shall be returned.

12344 If  $x$  is infinite, a domain error shall occur, and either a NaN (if supported), or an  
12345 implementation-defined value shall be returned.

12346 If  $x$  is  $\pm 0$  and  $y$  is not zero,  $\pm 0$  shall be returned.

12347 If  $x$  is not infinite and  $y$  is  $\pm \text{Inf}$ ,  $x$  shall be returned.

12348 If the correct value would cause underflow, and is representable, a range error may occur and  
12349 the correct value shall be returned.

## 12350 ERRORS

12351 These functions shall fail if:

12352 MX Domain Error The  $x$  argument is infinite or  $y$  is zero.

12353 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero, |  
12354 then *errno* shall be set to [EDOM]. If the integer expression (math\_errhandling |  
12355 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception |  
12356 shall be raised. |

12357 These functions may fail if:

12358 Range Error The result underflows.

12359 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero, |  
12360 then *errno* shall be set to [ERANGE]. If the integer expression |  
12361 (math\_errhandling & MATH\_ERREXCEPT) is non-zero, then the underflow |  
12362 floating-point exception shall be raised. |

12363 **EXAMPLES**

12364 None.

12365 **APPLICATION USAGE**

12366 On error, the expressions (math\_errhandling & MATH\_ERRNO) and (math\_errhandling &  
12367 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

12368 **RATIONALE**

12369 None.

12370 **FUTURE DIRECTIONS**

12371 None.

12372 **SEE ALSO**

12373 *feclearexcept()*, *fetetestexcept()*, *isnan()*, the Base Definitions volume of IEEE Std 1003.1-200x, |  
12374 Section 4.18, Treatment of Error Conditions for Mathematical Functions, <math.h> |

12375 **CHANGE HISTORY**

12376 First released in Issue 1. Derived from Issue 1 of the SVID.

12377 **Issue 5**

12378 The DESCRIPTION is updated to indicate how an application should check for an error. This  
12379 text was previously published in the APPLICATION USAGE section.

12380 **Issue 6**12381 The behavior for when the *y* argument is zero is now defined.

12382 The *fmodf()* and *fmodl()* functions are added for alignment with the ISO/IEC 9899:1999  
12383 standard.

12384 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
12385 revised to align with the ISO/IEC 9899:1999 standard.

12386 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
12387 marked.

12388 **NAME**12389 `fmtmsg` — display a message in the specified format on standard error and/or a system console12390 **SYNOPSIS**12391 XSI 

```
#include <fmtmsg.h>
```

12392 

```
int fmtmsg(long classification, const char *label, int severity,  
12393 const char *text, const char *action, const char *tag);
```

12394

12395 **DESCRIPTION**12396 The `fmtmsg()` function shall display messages in a specified format instead of the traditional  
12397 `printf()` function.12398 Based on a message's classification component, `fmtmsg()` shall write a formatted message either  
12399 to standard error, to the console, or to both.12400 A formatted message consists of up to five components as defined below. The component  
12401 *classification* is not part of a message displayed to the user, but defines the source of the message  
12402 and directs the display of the formatted message.12403 *classification* Contains the sum of identifying values constructed from the constants defined  
12404 below. Any one identifier from a subclass may be used in combination with a  
12405 single identifier from a different subclass. Two or more identifiers from the  
12406 same subclass should not be used together, with the exception of identifiers  
12407 from the display subclass. (Both display subclass identifiers may be used so  
12408 that messages can be displayed to both standard error and the system  
12409 console).12410 **Major Classifications**12411 Identifies the source of the condition. Identifiers are: MM\_HARD  
12412 (hardware), MM\_SOFT (software), and MM\_FIRM (firmware).12413 **Message Source Subclassifications**12414 Identifies the type of software in which the problem is detected.  
12415 Identifiers are: MM\_APPL (application), MM\_UTIL (utility), and  
12416 MM\_OPYSYS (operating system).12417 **Display Subclassifications**12418 Indicates where the message is to be displayed. Identifiers are:  
12419 MM\_PRINT to display the message on the standard error stream,  
12420 MM\_CONSOLE to display the message on the system console. One or  
12421 both identifiers may be used.12422 **Status Subclassifications**12423 Indicates whether the application can recover from the condition.  
12424 Identifiers are: MM\_RECOVER (recoverable) and MM\_NRECOV (non-  
12425 recoverable).12426 An additional identifier, MM\_NULLMC, indicates that no classification  
12427 component is supplied for the message.12428 *label* Identifies the source of the message. The format is two fields separated by a  
12429 colon. The first field is up to 10 bytes, the second is up to 14 bytes.12430 *severity* Indicates the seriousness of the condition. Identifiers for the levels of *severity*  
12431 are:

12432		MM_HALT	Indicates that the application has encountered a severe fault and is halting. Produces the string "HALT".
12433			
12434		MM_ERROR	Indicates that the application has detected a fault. Produces the string "ERROR".
12435			
12436		MM_WARNING	Indicates a condition that is out of the ordinary, that might be a problem, and should be watched. Produces the string "WARNING".
12437			
12438			
12439		MM_INFO	Provides information about a condition that is not in error. Produces the string "INFO".
12440			
12441		MM_NOSEV	Indicates that no severity level is supplied for the message.
12442	<i>text</i>		Describes the error condition that produced the message. The character string is not limited to a specific size. If the character string is empty, then the text produced is unspecified.
12443			
12444			
12445	<i>action</i>		Describes the first step to be taken in the error-recovery process. The <i>fmtmsg()</i> function precedes the action string with the prefix: "TO FIX:". The <i>action</i> string is not limited to a specific size.
12446			
12447			
12448	<i>tag</i>		An identifier that references on-line documentation for the message. Suggested usage is that <i>tag</i> includes the <i>label</i> and a unique identifying number. A sample <i>tag</i> is "XSI:cat:146".
12449			
12450			
12451			The <i>MSGVERB</i> environment variable (for message verbosity) shall determine for <i>fmtmsg()</i>
12452			which message components it is to select when writing messages to standard error. The value of
12453			<i>MSGVERB</i> shall be a colon-separated list of optional keywords. Valid <i>keywords</i> are: <i>label</i> , <i>severity</i> ,
12454			<i>text</i> , <i>action</i> , and <i>tag</i> . If <i>MSGVERB</i> contains a keyword for a component and the component's
12455			value is not the component's null value, <i>fmtmsg()</i> shall include that component in the message
12456			when writing the message to standard error. If <i>MSGVERB</i> does not include a keyword for a
12457			message component, that component is shall not be included in the display of the message. The
12458			keywords may appear in any order. If <i>MSGVERB</i> is not defined, if its value is the null string, if
12459			its value is not of the correct format, or if it contains keywords other than the valid ones listed
12460			above, <i>fmtmsg()</i> shall select all components.
12461			<i>MSGVERB</i> shall determine which components are selected for display to standard error. All
12462			message components shall be included in console messages.
12463	<b>RETURN VALUE</b>		
12464			The <i>fmtmsg()</i> function shall return one of the following values:
12465	MM_OK		The function succeeded.
12466	MM_NOTOK		The function failed completely.
12467	MM_NOMSG		The function was unable to generate a message on standard error, but
12468			otherwise succeeded.
12469	MM_NOCON		The function was unable to generate a console message, but otherwise
12470			succeeded.
12471	<b>ERRORS</b>		
12472			None.

12473 **EXAMPLES**

12474 1. The following example of *fmtmsg()*:

```
12475     fmtmsg(MM_PRINT, "XSI:cat", MM_ERROR, "illegal option",  
12476     "refer to cat in user's reference manual", "XSI:cat:001")
```

12477 produces a complete message in the specified message format:

```
12478     XSI:cat: ERROR: illegal option  
12479     TO FIX: refer to cat in user's reference manual XSI:cat:001
```

12480 2. When the environment variable *MSGVERB* is set as follows:

```
12481     MSGVERB=severity:text:action
```

12482 and Example 1 is used, *fmtmsg()* produces:

```
12483     ERROR: illegal option  
12484     TO FIX: refer to cat in user's reference manual
```

12485 **APPLICATION USAGE**

12486 One or more message components may be systematically omitted from messages generated by  
12487 an application by using the null value of the argument for that component.

12488 **RATIONALE**

12489 None.

12490 **FUTURE DIRECTIONS**

12491 None.

12492 **SEE ALSO**

12493 *printf()*, the Base Definitions volume of IEEE Std 1003.1-200x, <*fmtmsg.h*>

12494 **CHANGE HISTORY**

12495 First released in Issue 4, Version 2.

12496 **Issue 5**

12497 Moved from X/OPEN UNIX extension to BASE.

12498 **NAME**

12499 fnmatch — match a filename or a pathname |

12500 **SYNOPSIS**

12501 #include &lt;fnmatch.h&gt;

12502 int fnmatch(const char \**pattern*, const char \**string*, int *flags*);12503 **DESCRIPTION**

12504 The *fnmatch()* function shall match patterns as described in the Shell and Utilities volume of  
 12505 IEEE Std 1003.1-200x, Section 2.13.1, Patterns Matching a Single Character, and Section 2.13.2,  
 12506 Patterns Matching Multiple Characters. It checks the string specified by the *string* argument to  
 12507 see if it matches the pattern specified by the *pattern* argument.

12508 The *flags* argument shall modify the interpretation of *pattern* and *string*. It is the bitwise-inclusive  
 12509 OR of zero or more of the flags defined in <fnmatch.h>. If the FNM\_PATHNAME flag is set in  
 12510 *flags*, then a slash character ( '/' ) in *string* shall be explicitly matched by a slash in *pattern*; it shall  
 12511 not be matched by either the asterisk or question-mark special characters, nor by a bracket  
 12512 expression. If the FNM\_PATHNAME flag is not set, the slash character shall be treated as an  
 12513 ordinary character. |

12514 If FNM\_NOESCAPE is not set in *flags*, a backslash character ( '\ ' ) in *pattern* followed by any  
 12515 other character shall match that second character in *string*. In particular, "\\\" shall match a  
 12516 backslash in *string*. If FNM\_NOESCAPE is set, a backslash character shall be treated as an  
 12517 ordinary character.

12518 If FNM\_PERIOD is set in *flags*, then a leading period ( '.' ) in *string* shall match a period in  
 12519 *pattern*; as described by rule 2 in the Shell and Utilities volume of IEEE Std 1003.1-200x, Section  
 12520 2.13.3, Patterns Used for Filename Expansion where the location of “leading” is indicated by the  
 12521 value of FNM\_PATHNAME:

- 12522 • If FNM\_PATHNAME is set, a period is “leading” if it is the first character in *string* or if it  
 12523 immediately follows a slash.

- 12524 • If FNM\_PATHNAME is not set, a period is “leading” only if it is the first character of *string*.

12525 If FNM\_PERIOD is not set, then no special restrictions are placed on matching a period.

12526 **RETURN VALUE**

12527 If *string* matches the pattern specified by *pattern*, then *fnmatch()* shall return 0. If there is no  
 12528 match, *fnmatch()* shall return FNM\_NOMATCH, which is defined in <fnmatch.h>. If an error  
 12529 occurs, *fnmatch()* shall return another non-zero value. |

12530 **ERRORS**

12531 No errors are defined.

12532 **EXAMPLES**

12533 None.

12534 **APPLICATION USAGE**

12535 The *fnmatch()* function has two major uses. It could be used by an application or utility that  
 12536 needs to read a directory and apply a pattern against each entry. The *find* utility is an example of  
 12537 this. It can also be used by the *pax* utility to process its *pattern* operands, or by applications that  
 12538 need to match strings in a similar manner.

12539 The name *fnmatch()* is intended to imply *filename* match, rather than *pathname* match. The default  
 12540 action of this function is to match filenames, rather than pathnames, since it gives no special  
 12541 significance to the slash character. With the FNM\_PATHNAME flag, *fnmatch()* does match  
 12542 pathnames, but without tilde expansion, parameter expansion, or special treatment for a period |

12543 at the beginning of a filename.

12544 **RATIONALE**

12545 This function replaced the REG\_FILENAME flag of *regcomp()* in early proposals of this volume  
12546 of IEEE Std 1003.1-200x. It provides virtually the same functionality as the *regcomp()* and  
12547 *regexexec()* functions using the REG\_FILENAME and REG\_FSLASH flags (the REG\_FSLASH flag  
12548 was proposed for *regcomp()*, and would have had the opposite effect from FNM\_PATHNAME),  
12549 but with a simpler function and less system overhead.

12550 **FUTURE DIRECTIONS**

12551 None.

12552 **SEE ALSO**

12553 *glob()*, *wordexp()*, the Base Definitions volume of IEEE Std 1003.1-200x, <fnmatch.h>, the Shell  
12554 and Utilities volume of IEEE Std 1003.1-200x

12555 **CHANGE HISTORY**

12556 First released in Issue 4. Derived from the ISO POSIX-2 standard.

12557 **Issue 5**

12558 Moved from POSIX2 C-language Binding to BASE.

## 12559 NAME

12560 fopen — open a stream

## 12561 SYNOPSIS

12562 #include &lt;stdio.h&gt;

12563 FILE \*fopen(const char \*restrict filename, const char \*restrict mode);

## 12564 DESCRIPTION

12565 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 12566 conflict between the requirements described here and the ISO C standard is unintentional. This  
 12567 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

12568 The *fopen()* function shall open the file whose pathname is the string pointed to by *filename*, and  
 12569 associates a stream with it.

12570 The *mode* argument points to a string. If the string is one of the following, the file shall be opened  
 12571 in the indicated mode. Otherwise, the behavior is undefined.

12572 *r* or *rb* Open file for reading.

12573 *w* or *wb* Truncate to zero length or create file for writing.

12574 *a* or *ab* Append; open or create file for writing at end-of-file.

12575 *r+* or *rb+* or *r+b* Open file for update (reading and writing).

12576 *w+* or *wb+* or *w+b* Truncate to zero length or create file for update.

12577 *a+* or *ab+* or *a+b* Append; open or create file for update, writing at end-of-file.

12578 CX The character 'b' shall have no effect, but is allowed for ISO C standard conformance. Opening  
 12579 a file with read mode (*r* as the first character in the *mode* argument) shall fail if the file does not  
 12580 exist or cannot be read.

12581 Opening a file with append mode (*a* as the first character in the *mode* argument) shall cause all  
 12582 subsequent writes to the file to be forced to the then current end-of-file, regardless of intervening  
 12583 calls to *fseek()*.

12584 When a file is opened with update mode ('+' as the second or third character in the *mode*  
 12585 argument), both input and output may be performed on the associated stream. However, the  
 12586 application shall ensure that output is not directly followed by input without an intervening call  
 12587 to *fflush()* or to a file positioning function (*fseek()*, *fsetpos()*, or *rewind()*), and input is not directly  
 12588 followed by output without an intervening call to a file positioning function, unless the input  
 12589 operation encounters end-of-file.

12590 When opened, a stream is fully buffered if and only if it can be determined not to refer to an  
 12591 interactive device. The error and end-of-file indicators for the stream shall be cleared.

12592 CX If *mode* is *w*, *wb*, *a*, *ab*, *w+*, *wb+*, *w+b*, *a+*, *ab+*, or *a+b*, and the file did not previously exist, upon  
 12593 successful completion, *fopen()* function shall mark for update the *st\_atime*, *st\_ctime*, and *st\_mtime*  
 12594 fields of the file and the *st\_ctime* and *st\_mtime* fields of the parent directory.

12595 If *mode* is *w*, *wb*, *w+*, *wb+*, or *w+b*, and the file did previously exist, upon successful completion,  
 12596 *fopen()* shall mark for update the *st\_ctime* and *st\_mtime* fields of the file. The *fopen()* function  
 12597 shall allocate a file descriptor as *open()* does.

12598 XSI After a successful call to the *fopen()* function, the orientation of the stream shall be cleared, the  
 12599 encoding rule shall be cleared, and the associated *mbstate\_t* object shall be set to describe an  
 12600 initial conversion state.

12601 CX The largest value that can be represented correctly in an object of type `off_t` shall be established  
 12602 as the offset maximum in the open file description.

### 12603 RETURN VALUE

12604 Upon successful completion, `fopen()` shall return a pointer to the object controlling the stream.  
 12605 CX Otherwise, a null pointer shall be returned, and `errno` shall be set to indicate the error.

### 12606 ERRORS

12607 The `fopen()` function shall fail if:

12608 CX [EACCES] Search permission is denied on a component of the path prefix, or the file  
 12609 exists and the permissions specified by `mode` are denied, or the file does not  
 12610 exist and write permission is denied for the parent directory of the file to be  
 12611 created.

12612 CX [EINTR] A signal was caught during `fopen()`.

12613 CX [EISDIR] The named file is a directory and `mode` requires write access.

12614 CX [ELOOP] A loop exists in symbolic links encountered during resolution of the `path`  
 12615 argument.

12616 CX [EMFILE] {OPEN\_MAX} file descriptors are currently open in the calling process.

12617 CX [ENAMETOOLONG]

12618 The length of the `filename` argument exceeds {PATH\_MAX} or a pathname  
 12619 component is longer than {NAME\_MAX}.

12620 CX [ENFILE] The maximum allowable number of files is currently open in the system.

12621 CX [ENOENT] A component of `filename` does not name an existing file or `filename` is an empty  
 12622 string.

12623 CX [ENOSPC] The directory or file system that would contain the new file cannot be  
 12624 expanded, the file does not exist, and it was to be created.

12625 CX [ENOTDIR] A component of the path prefix is not a directory.

12626 CX [ENXIO] The named file is a character special or block special file, and the device  
 12627 associated with this special file does not exist.

12628 CX [EOVERFLOW] The named file is a regular file and the size of the file cannot be represented  
 12629 correctly in an object of type `off_t`.

12630 CX [EROFS] The named file resides on a read-only file system and `mode` requires write  
 12631 access.

12632 The `fopen()` function may fail if:

12633 CX [EINVAL] The value of the `mode` argument is not valid.

12634 CX [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
 12635 resolution of the `path` argument.

12636 CX [EMFILE] {FOPEN\_MAX} streams are currently open in the calling process.

12637 CX [EMFILE] {STREAM\_MAX} streams are currently open in the calling process.

12638 CX [ENAMETOOLONG]

12639 Pathname resolution of a symbolic link produced an intermediate result  
 12640 whose length exceeds {PATH\_MAX}.

- 12641 CX [ENOMEM] Insufficient storage space is available.
- 12642 CX [ETXTBSY] The file is a pure procedure (shared text) file that is being executed and *mode* requires write access.
- 12643

12644 **EXAMPLES**12645 **Opening a File**

12646 The following example tries to open the file named **file** for reading. The *fopen()* function returns  
 12647 a file pointer that is used in subsequent *fgets()* and *fclose()* calls. If the program cannot open the  
 12648 file, it just ignores it.

```
12649 #include <stdio.h>
12650 ...
12651 FILE *fp;
12652 ...
12653 void rgrep(const char *file)
12654 {
12655     ...
12656     if ((fp = fopen(file, "r")) == NULL)
12657         return;
12658     ...
12659 }
```

12660 **APPLICATION USAGE**

12661 None.

12662 **RATIONALE**

12663 None.

12664 **FUTURE DIRECTIONS**

12665 None.

12666 **SEE ALSO**

12667 *fclose()*, *fdopen()*, *freopen()*, the Base Definitions volume of IEEE Std 1003.1-200x, <stdio.h>

12668 **CHANGE HISTORY**

12669 First released in Issue 1. Derived from Issue 1 of the SVID.

12670 **Issue 5**

12671 Large File Summit extensions are added.

12672 **Issue 6**

12673 Extensions beyond the ISO C standard are now marked.

12674 The following new requirements on POSIX implementations derive from alignment with the  
 12675 Single UNIX Specification:

- 12676 • In the DESCRIPTION, text is added to indicate setting of the offset maximum in the open file  
 12677 description. This change is to support large files.
- 12678 • In the ERRORS section, the [Eoverflow] condition is added. This change is to support  
 12679 large files.
- 12680 • The [ELOOP] mandatory error condition is added.
- 12681 • The [EINVAL], [EMFILE], [ENAMETOOLONG], [ENOMEM], and [ETXTBSY] optional error  
 12682 conditions are added.

- 12683 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.
- 12684 The following changes are made for alignment with the ISO/IEC 9899: 1999 standard:
- 12685 • The prototype for *fopen()* is updated.
  - 12686 • The DESCRIPTION is updated to note that if the argument *mode* points to a string other than  
12687 those listed, then the behavior is undefined.
- 12688 The wording of the mandatory [ELOOP] error condition is updated, and a second optional  
12689 [ELOOP] error condition is added.

## 12690 NAME

12691 fork — create a new process

## 12692 SYNOPSIS

12693 #include &lt;unistd.h&gt;

12694 pid\_t fork(void);

## 12695 DESCRIPTION

12696 The *fork()* function shall create a new process. The new process (child process) shall be an exact  
 12697 copy of the calling process (parent process) except as detailed below:

- 12698 • The child process shall have a unique process ID. |
- 12699 • The child process ID also shall not match any active process group ID. |
- 12700 • The child process shall have a different parent process ID, which shall be the process ID of |
- 12701 the calling process. |
- 12702 • The child process shall have its own copy of the parent's file descriptors. Each of the child's |
- 12703 file descriptors shall refer to the same open file description with the corresponding file |
- 12704 descriptor of the parent. |
- 12705 • The child process shall have its own copy of the parent's open directory streams. Each open |
- 12706 directory stream in the child process may share directory stream positioning with the |
- 12707 corresponding directory stream of the parent. |
- 12708 XSI • The child process shall have its own copy of the parent's message catalog descriptors. |
- 12709 • The child process' values of *tms\_utime*, *tms\_stime*, *tms\_cutime*, and *tms\_cstime* shall be set to 0. |
- 12710 • The time left until an alarm clock signal shall be reset to zero, and the alarm, if any, shall be |
- 12711 canceled; see *alarm()*. |
- 12712 XSI • All *semadj* values shall be cleared. |
- 12713 • File locks set by the parent process shall not be inherited by the child process. |
- 12714 • The set of signals pending for the child process shall be initialized to the empty set. |
- 12715 XSI • Interval timers shall be reset in the child process. |
- 12716 SEM • Any semaphores that are open in the parent process shall also be open in the child process. |
- 12717 ML • The child process shall not inherit any address space memory locks established by the parent |
- 12718 process via calls to *mlockall()* or *mlock()*. |
- 12719 MF|SHM • Memory mappings created in the parent shall be retained in the child process. |
- 12720 MAP\_PRIVATE mappings inherited from the parent shall also be MAP\_PRIVATE mappings |
- 12721 in the child, and any modifications to the data in these mappings made by the parent prior to |
- 12722 calling *fork()* shall be visible to the child. Any modifications to the data in MAP\_PRIVATE |
- 12723 mappings made by the parent after *fork()* returns shall be visible only to the parent. |
- 12724 Modifications to the data in MAP\_PRIVATE mappings made by the child shall be visible only |
- 12725 to the child. |
- 12726 PS • For the SCHED\_FIFO and SCHED\_RR scheduling policies, the child process shall inherit the |
- 12727 policy and priority settings of the parent process during a *fork()* function. For other |
- 12728 scheduling policies, the policy and priority settings on *fork()* are implementation-defined. |
- 12729 TMR • Per-process timers created by the parent shall not be inherited by the child process. |
- 12730 MSG • The child process shall have its own copy of the message queue descriptors of the parent. |
- 12731 Each of the message descriptors of the child shall refer to the same open message queue |

12732		description as the corresponding message descriptor of the parent.	
12733	AIO	• No asynchronous input or asynchronous output operations shall be inherited by the child	
12734		process.	
12735		• A process shall be created with a single thread. If a multi-threaded process calls <i>fork()</i> , the	
12736		new process shall contain a replica of the calling thread and its entire address space, possibly	
12737		including the states of mutexes and other resources. Consequently, to avoid errors, the child	
12738		process may only execute async-signal-safe operations until such time as one of the <i>exec</i>	
12739	THR	functions is called. Fork handlers may be established by means of the <i>pthread_atfork()</i>	
12740		function in order to maintain application invariants across <i>fork()</i> calls.	
12741	TRC TRI	• If the Trace option and the Trace Inherit option are both supported:	
12742		If the calling process was being traced in a trace stream that had its inheritance policy set to	
12743		POSIX_TRACE_INHERITED, the child process shall be traced into that trace stream, and the	
12744		child process shall inherit the parent's mapping of trace event names to trace event type	
12745		identifiers. If the trace stream in which the calling process was being traced had its	
12746		inheritance policy set to POSIX_TRACE_CLOSE_FOR_CHILD, the child process shall not be	
12747		traced into that trace stream. The inheritance policy is set by a call to the	
12748		<i>posix_trace_attr_setinherited()</i> function.	
12749	TRC	• If the Trace option is supported, but the Trace Inherit option is not supported:	
12750		The child process shall not be traced into any of the trace streams of its parent process.	
12751	TRC	• If the Trace option is supported, the child process of a trace controller process shall not	
12752		control the trace streams controlled by its parent process.	
12753	CPT	• The initial value of the CPU-time clock of the child process shall be set to zero.	
12754	TCT	• The initial value of the CPU-time clock of the single thread of the child process shall be set to	
12755		zero.	
12756		All other process characteristics defined by IEEE Std 1003.1-200x shall be the same in the parent	
12757		and child processes. The inheritance of process characteristics not defined by	
12758		IEEE Std 1003.1-200x is unspecified by IEEE Std 1003.1-200x.	
12759		After <i>fork()</i> , both the parent and the child processes shall be capable of executing independently	
12760		before either one terminates.	
12761		<b>RETURN VALUE</b>	
12762		Upon successful completion, <i>fork()</i> shall return 0 to the child process and shall return the	
12763		process ID of the child process to the parent process. Both processes shall continue to execute	
12764		from the <i>fork()</i> function. Otherwise, -1 shall be returned to the parent process, no child process	
12765		shall be created, and <i>errno</i> shall be set to indicate the error.	
12766		<b>ERRORS</b>	
12767		The <i>fork()</i> function shall fail if:	
12768	[EAGAIN]	The system lacked the necessary resources to create another process, or the	
12769		system-imposed limit on the total number of processes under execution	
12770		system-wide or by a single user {CHILD_MAX} would be exceeded.	
12771		The <i>fork()</i> function may fail if:	
12772	[ENOMEM]	Insufficient storage space is available.	

12773 **EXAMPLES**

12774 None.

12775 **APPLICATION USAGE**

12776 None.

12777 **RATIONALE**

12778 Many historical implementations have timing windows where a signal sent to a process group  
 12779 (for example, an interactive SIGINT) just prior to or during execution of *fork()* is delivered to the  
 12780 parent following the *fork()* but not to the child because the *fork()* code clears the child's set of  
 12781 pending signals. This volume of IEEE Std 1003.1-200x does not require, or even permit, this  
 12782 behavior. However, it is pragmatic to expect that problems of this nature may continue to exist  
 12783 in implementations that appear to conform to this volume of IEEE Std 1003.1-200x and pass  
 12784 available verification suites. This behavior is only a consequence of the implementation failing to  
 12785 make the interval between signal generation and delivery totally invisible. From the  
 12786 application's perspective, a *fork()* call should appear atomic. A signal that is generated prior to  
 12787 the *fork()* should be delivered prior to the *fork()*. A signal sent to the process group after the  
 12788 *fork()* should be delivered to both parent and child. The implementation may actually initialize  
 12789 internal data structures corresponding to the child's set of pending signals to include signals  
 12790 sent to the process group during the *fork()*. Since the *fork()* call can be considered as atomic  
 12791 from the application's perspective, the set would be initialized as empty and such signals would  
 12792 have arrived after the *fork()*; see also <signal.h>.

12793 One approach that has been suggested to address the problem of signal inheritance across *fork()*  
 12794 is to add an [EINTR] error, which would be returned when a signal is detected during the call.  
 12795 While this is preferable to losing signals, it was not considered an optimal solution. Although it  
 12796 is not recommended for this purpose, such an error would be an allowable extension for an  
 12797 implementation.

12798 The [ENOMEM] error value is reserved for those implementations that detect and distinguish  
 12799 such a condition. This condition occurs when an implementation detects that there is not enough  
 12800 memory to create the process. This is intended to be returned when [EAGAIN] is inappropriate  
 12801 because there can never be enough memory (either primary or secondary storage) to perform the  
 12802 operation. Since *fork()* duplicates an existing process, this must be a condition where there is  
 12803 sufficient memory for one such process, but not for two. Many historical implementations  
 12804 actually return [ENOMEM] due to temporary lack of memory, a case that is not generally  
 12805 distinct from [EAGAIN] from the perspective of a conforming application.

12806 Part of the reason for including the optional error [ENOMEM] is because the SVID specifies it  
 12807 and it should be reserved for the error condition specified there. The condition is not applicable  
 12808 on many implementations.

12809 IEEE Std 1003.1-1988 neglected to require concurrent execution of the parent and child of *fork()*.  
 12810 A system that single-threads processes was clearly not intended and is considered an  
 12811 unacceptable "toy implementation" of this volume of IEEE Std 1003.1-200x. The only objection  
 12812 anticipated to the phrase "executing independently" is testability, but this assertion should be  
 12813 testable. Such tests require that both the parent and child can block on a detectable action of the  
 12814 other, such as a write to a pipe or a signal. An interactive exchange of such actions should be  
 12815 possible for the system to conform to the intent of this volume of IEEE Std 1003.1-200x.

12816 The [EAGAIN] error exists to warn applications that such a condition might occur. Whether it  
 12817 occurs or not is not in any practical sense under the control of the application because the  
 12818 condition is usually a consequence of the user's use of the system, not of the application's code.  
 12819 Thus, no application can or should rely upon its occurrence under any circumstances, nor  
 12820 should the exact semantics of what concept of "user" is used be of concern to the application  
 12821 writer. Validation writers should be cognizant of this limitation.

12822 There are two reasons why POSIX programmers call *fork()*. One reason is to create a new thread  
 12823 of control within the same program (which was originally only possible in POSIX by creating a  
 12824 new process); the other is to create a new process running a different program. In the latter case,  
 12825 the call to *fork()* is soon followed by a call to one of the *exec* functions.

12826 The general problem with making *fork()* work in a multi-threaded world is what to do with all  
 12827 of the threads. There are two alternatives. One is to copy all of the threads into the new process.  
 12828 This causes the programmer or implementation to deal with threads that are suspended on  
 12829 system calls or that might be about to execute system calls that should not be executed in the  
 12830 new process. The other alternative is to copy only the thread that calls *fork()*. This creates the  
 12831 difficulty that the state of process-local resources is usually held in process memory. If a thread  
 12832 that is not calling *fork()* holds a resource, that resource is never released in the child process  
 12833 because the thread whose job it is to release the resource does not exist in the child process.

12834 When a programmer is writing a multi-threaded program, the first described use of *fork()*,  
 12835 creating new threads in the same program, is provided by the *pthread\_create()* function. The  
 12836 *fork()* function is thus used only to run new programs, and the effects of calling functions that  
 12837 require certain resources between the call to *fork()* and the call to an *exec* function are undefined.

12838 The addition of the *forkall()* function to the standard was considered and rejected. The *forkall()*  
 12839 function lets all the threads in the parent be duplicated in the child. This essentially duplicates  
 12840 the state of the parent in the child. This allows threads in the child to continue processing and  
 12841 allows locks and the state to be preserved without explicit *pthread\_atfork()* code. The calling  
 12842 process has to ensure that the threads processing state that is shared between the parent and  
 12843 child (that is, file descriptors or MAP\_SHARED memory) behaves properly after *forkall()*. For  
 12844 example, if a thread is reading a file descriptor in the parent when *forkall()* is called, then two  
 12845 threads (one in the parent and one in the child) are reading the file descriptor after the *forkall()*.  
 12846 If this is not desired behavior, the parent process has to synchronize with such threads before  
 12847 calling *forkall()*.

12848 When *forkall()* is called, threads, other than the calling thread, that are in POSIX System  
 12849 Interfaces functions that can return with an [EINTR] error may have those functions return  
 12850 [EINTR] if the implementation cannot ensure that the function behaves correctly in the parent  
 12851 and child. In particular, *pthread\_cond\_wait()* and *pthread\_cond\_timedwait()* need to return in order  
 12852 to ensure that the condition has not changed. These functions can be awakened by a spurious  
 12853 condition wakeup rather than returning [EINTR].

#### 12854 FUTURE DIRECTIONS

12855 None.

#### 12856 SEE ALSO

12857 *alarm()*, *exec*, *fcntl()*, *posix\_trace\_attr\_getinherited()*, *posix\_trace\_trid\_eventid\_open()*, *semop()*,  
 12858 *signal()*, *times()*, the Base Definitions volume of IEEE Std 1003.1-200x, <sys/types.h>, <unistd.h>

#### 12859 CHANGE HISTORY

12860 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 12861 Issue 5

12862 The DESCRIPTION is changed for alignment with the POSIX Realtime Extension and the POSIX  
 12863 Threads Extension.

#### 12864 Issue 6

12865 The following new requirements on POSIX implementations derive from alignment with the  
 12866 Single UNIX Specification:

- 12867 • The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was  
 12868 required for conforming implementations of previous POSIX specifications, it was not

12869 required for UNIX applications.

12870 The following changes were made to align with the IEEE P1003.1a draft standard:

12871 • The effect of *fork()* on a pending alarm call in the child process is clarified.

12872 The description of CPU-time clock semantics is added for alignment with IEEE Std 1003.1d-1999.

12873 The description of tracing semantics is added for alignment with IEEE Std 1003.1q-2000.

12874 **NAME**

12875 fpathconf, pathconf — get configurable pathname variables

12876 **SYNOPSIS**

12877 #include <unistd.h>

12878 long fpathconf(int *fildev*, int *name*);

12879 long pathconf(const char \**path*, int *name*);

12880 **DESCRIPTION**

12881 The *fpathconf()* and *pathconf()* functions shall determine the current value of a configurable limit  
12882 or option (*variable*) that is associated with a file or directory.

12883 For *pathconf()*, the *path* argument points to the pathname of a file or directory.

12884 For *fpathconf()*, the *fildev* argument is an open file descriptor.

12885 The *name* argument represents the variable to be queried relative to that file or directory.  
12886 Implementations shall support all of the variables listed in the following table and may support  
12887 others. The variables in the following table come from <limits.h> or <unistd.h> and the  
12888 symbolic constants, defined in <unistd.h>, are the corresponding values used for *name*. Support  
12889 for some pathname configuration variables is dependent on implementation options (see  
12890 shading and margin codes in the table below). Where an implementation option is not  
12891 supported, the variable need not be supported.

12892

12893

Variable	Value of <i>name</i>	Requirements
{FILESIZEBITS}	_PC_FILESIZEBITS	3, 4
{LINK_MAX}	_PC_LINK_MAX	1
{MAX_CANON}	_PC_MAX_CANON	2
{MAX_INPUT}	_PC_MAX_INPUT	2
{NAME_MAX}	_PC_NAME_MAX	3, 4
{PATH_MAX}	_PC_PATH_MAX	4, 5
{PIPE_BUF}	_PC_PIPE_BUF	6
{POSIX_ALLOC_SIZE_MIN}	_PC_ALLOC_SIZE_MIN	
{POSIX_REC_INCR_XFER_SIZE}	_PC_REC_INCR_XFER_SIZE	
{POSIX_REC_MAX_XFER_SIZE}	_PC_REC_MAX_XFER_SIZE	
{POSIX_REC_MIN_XFER_SIZE}	_PC_REC_MIN_XFER_SIZE	
{POSIX_REC_XFER_ALIGN}	_PC_REC_XFER_ALIGN	
{SYMLINK_MAX}	_PC_SYMLINK_MAX	4, 9
_POSIX_CHOWN_RESTRICTED	_PC_CHOWN_RESTRICTED	7
_POSIX_NO_TRUNC	_PC_NO_TRUNC	3, 4
_POSIX_VDISABLE	_PC_VDISABLE	2
_POSIX_ASYNC_IO	_PC_ASYNC_IO	8
_POSIX_PRIO_IO	_PC_PRIO_IO	8
_POSIX_SYNC_IO	_PC_SYNC_IO	8

12901 ADV

12902 ADV

12903 ADV

12904 ADV

12905 ADV

12906

12907

12908

12909

12910

12911

12912

12913	<b>Requirements</b>	
12914	1. If <i>path</i> or <i>fildev</i> refers to a directory, the value returned shall apply to the directory itself.	
12915	2. If <i>path</i> or <i>fildev</i> does not refer to a terminal file, it is unspecified whether an implementation	
12916	supports an association of the variable name with the specified file.	
12917	3. If <i>path</i> or <i>fildev</i> refers to a directory, the value returned shall apply to filenames within the	
12918	directory.	
12919	4. If <i>path</i> or <i>fildev</i> does not refer to a directory, it is unspecified whether an implementation	
12920	supports an association of the variable name with the specified file.	
12921	5. If <i>path</i> or <i>fildev</i> refers to a directory, the value returned shall be the maximum length of a	
12922	relative pathname when the specified directory is the working directory.	
12923	6. If <i>path</i> refers to a FIFO, or <i>fildev</i> refers to a pipe or FIFO, the value returned shall apply to	
12924	the referenced object. If <i>path</i> or <i>fildev</i> refers to a directory, the value returned shall apply to	
12925	any FIFO that exists or can be created within the directory. If <i>path</i> or <i>fildev</i> refers to any	
12926	other type of file, it is unspecified whether an implementation supports an association of	
12927	the variable name with the specified file.	
12928	7. If <i>path</i> or <i>fildev</i> refers to a directory, the value returned shall apply to any files, other than	
12929	directories, that exist or can be created within the directory.	
12930	8. If <i>path</i> or <i>fildev</i> refers to a directory, it is unspecified whether an implementation supports	
12931	an association of the variable name with the specified file.	
12932	9. If <i>path</i> or <i>fildev</i> refers to a directory, the value returned shall be the maximum length of the	
12933	string that a symbolic link in that directory can contain.	
12934	<b>RETURN VALUE</b>	
12935	If <i>name</i> is an invalid value, both <i>pathconf()</i> and <i>fpathconf()</i> shall return <code>-1</code> and set <i>errno</i> to	
12936	indicate the error.	
12937	If the variable corresponding to <i>name</i> has no limit for the <i>path</i> or file descriptor, both <i>pathconf()</i>	
12938	and <i>fpathconf()</i> shall return <code>-1</code> without changing <i>errno</i> . If the implementation needs to use <i>path</i>	
12939	to determine the value of <i>name</i> and the implementation does not support the association of <i>name</i>	
12940	with the file specified by <i>path</i> , or if the process did not have appropriate privileges to query the	
12941	file specified by <i>path</i> , or <i>path</i> does not exist, <i>pathconf()</i> shall return <code>-1</code> and set <i>errno</i> to indicate the	
12942	error.	
12943	If the implementation needs to use <i>fildev</i> to determine the value of <i>name</i> and the implementation	
12944	does not support the association of <i>name</i> with the file specified by <i>fildev</i> , or if <i>fildev</i> is an invalid	
12945	file descriptor, <i>fpathconf()</i> shall return <code>-1</code> and set <i>errno</i> to indicate the error.	
12946	Otherwise, <i>pathconf()</i> or <i>fpathconf()</i> shall return the current variable value for the file or	
12947	directory without changing <i>errno</i> . The value returned shall not be more restrictive than the	
12948	corresponding value available to the application when it was compiled with the	
12949	implementation's <code>&lt;limits.h&gt;</code> or <code>&lt;unistd.h&gt;</code> .	
12950	<b>ERRORS</b>	
12951	The <i>pathconf()</i> function shall fail if:	
12952	[EINVAL]           The value of <i>name</i> is not valid.	
12953	[ELOOP]            A loop exists in symbolic links encountered during resolution of the <i>path</i>	
12954	argument.	
12955	The <i>pathconf()</i> function may fail if:	

12956	[EACCES]	Search permission is denied for a component of the path prefix.
12957	[EINVAL]	The implementation does not support an association of the variable <i>name</i> with the specified file.
12958		
12959	[ELOOP]	More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the <i>path</i> argument.
12960		
12961	[ENAMETOOLONG]	
12962		The length of the <i>path</i> argument exceeds {PATH_MAX} or a pathname component is longer than {NAME_MAX}.
12963		
12964	[ENAMETOOLONG]	
12965		As a result of encountering a symbolic link in resolution of the <i>path</i> argument, the length of the substituted pathname string exceeded {PATH_MAX}.
12966		
12967	[ENOENT]	A component of <i>path</i> does not name an existing file or <i>path</i> is an empty string.
12968	[ENOTDIR]	A component of the path prefix is not a directory.
12969		The <i>fpathconf()</i> function shall fail if:
12970	[EINVAL]	The value of <i>name</i> is not valid.
12971		The <i>fpathconf()</i> function may fail if:
12972	[EBADF]	The <i>fdes</i> argument is not a valid file descriptor.
12973	[EINVAL]	The implementation does not support an association of the variable <i>name</i> with the specified file.
12974		
12975	<b>EXAMPLES</b>	
12976		None.
12977	<b>APPLICATION USAGE</b>	
12978		None.
12979	<b>RATIONALE</b>	
12980		The <i>pathconf()</i> function was proposed immediately after the <i>sysconf()</i> function when it was realized that some configurable values may differ across file system, directory, or device boundaries.
12981		
12982		
12983		For example, {NAME_MAX} frequently changes between System V and BSD-based file systems; System V uses a maximum of 14, BSD 255. On an implementation that provides both types of file systems, an application would be forced to limit all pathname components to 14 bytes, as this would be the value specified in <limits.h> on such a system.
12984		
12985		
12986		
12987		Therefore, various useful values can be queried on any pathname or file descriptor, assuming that the appropriate permissions are in place.
12988		
12989		The value returned for the variable {PATH_MAX} indicates the longest relative pathname that could be given if the specified directory is the process' current working directory. A process may not always be able to generate a name that long and use it if a subdirectory in the pathname crosses into a more restrictive file system.
12990		
12991		
12992		
12993		The value returned for the variable _POSIX_CHOWN_RESTRICTED also applies to directories that do not have file systems mounted on them. The value may change when crossing a mount point, so applications that need to know should check for each directory. (An even easier check is to try the <i>chown()</i> function and look for an error in case it happens.)
12994		
12995		
12996		
12997		Unlike the values returned by <i>sysconf()</i> , the pathname-oriented variables are potentially more volatile and are not guaranteed to remain constant throughout the process' lifetime. For
12998		

12999 example, in between two calls to *pathconf()*, the file system in question may have been  
 13000 unmounted and remounted with different characteristics.

13001 Also note that most of the errors are optional. If one of the variables always has the same value  
 13002 on an implementation, the implementation need not look at *path* or *fildev* to return that value and  
 13003 is, therefore, not required to detect any of the errors except the meaning of [EINVAL] that  
 13004 indicates that the value of *name* is not valid for that variable.

13005 If the value of any of the limits are unspecified (logically infinite), they will not be defined in  
 13006 <limits.h> and the *pathconf()* and *fpathconf()* functions return -1 without changing *errno*. This  
 13007 can be distinguished from the case of giving an unrecognized *name* argument because *errno* is set  
 13008 to [EINVAL] in this case.

13009 Since -1 is a valid return value for the *pathconf()* and *fpathconf()* functions, applications should  
 13010 set *errno* to zero before calling them and check *errno* only if the return value is -1.

13011 For the case of {SYMLINK\_MAX}, since both *pathconf()* and *open()* follow symbolic links, there  
 13012 is no way that *path* or *fildev* could refer to a symbolic link.

### 13013 FUTURE DIRECTIONS

13014 None.

### 13015 SEE ALSO

13016 *confstr()*, *sysconf()*, the Base Definitions volume of IEEE Std 1003.1-200x, <limits.h>, <unistd.h>,  
 13017 the Shell and Utilities volume of IEEE Std 1003.1-200x

### 13018 CHANGE HISTORY

13019 First released in Issue 3.

13020 Entry included for alignment with the POSIX.1-1988 standard.

### 13021 Issue 5

13022 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension.

13023 Large File Summit extensions are added.

### 13024 Issue 6

13025 The following new requirements on POSIX implementations derive from alignment with the  
 13026 Single UNIX Specification:

- 13027 • The DESCRIPTION is updated to include {FILESIZEBITS}.
- 13028 • The [ELOOP] mandatory error condition is added.
- 13029 • A second [ENAMETOOLONG] is added as an optional error condition.

13030 The following changes were made to align with the IEEE P1003.1a draft standard:

- 13031 • The \_PC\_SYMLINK\_MAX entry is added to the table in the DESCRIPTION.

13032 The *pathconf()* variables {POSIX\_ALLOC\_SIZE\_MIN}, {POSIX\_REC\_INCR\_XFER\_SIZE},  
 13033 {POSIX\_REC\_MAX\_XFER\_SIZE}, {POSIX\_REC\_MIN\_XFER\_SIZE},  
 13034 {POSIX\_REC\_XFER\_ALIGN} and their associated names are added for alignment with  
 13035 IEEE Std 1003.1d-1999.

13036 **NAME**

13037 fpclassify — classify real floating type

13038 **SYNOPSIS**

13039 #include &lt;math.h&gt;

13040 int fpclassify(real-floating x);

13041 **DESCRIPTION**

13042 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
13043 conflict between the requirements described here and the ISO C standard is unintentional. This  
13044 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

13045 The *fpclassify()* macro shall classify its argument value as NaN, infinite, normal, subnormal,  
13046 zero, or into another implementation-defined category. First, an argument represented in a  
13047 format wider than its semantic type is converted to its semantic type. Then classification is based  
13048 on the type of the argument.

13049 **RETURN VALUE**

13050 The *fpclassify()* macro shall return the value of the number classification macro appropriate to  
13051 the value of its argument.

13052 **ERRORS**

13053 No errors are defined.

13054 **EXAMPLES**

13055 None.

13056 **APPLICATION USAGE**

13057 None.

13058 **RATIONALE**

13059 None.

13060 **FUTURE DIRECTIONS**

13061 None.

13062 **SEE ALSO**

13063 *isfinite()*, *isinf()*, *isnan()*, *isnormal()*, *signbit()*, the Base Definitions volume of  
13064 IEEE Std 1003.1-200x, <math.h>

13065 **CHANGE HISTORY**

13066 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

13067 **NAME**

13068 fprintf, printf, snprintf, sprintf — print formatted output

13069 **SYNOPSIS**

13070 #include &lt;stdio.h&gt;

13071 int fprintf(FILE \*restrict *stream*, const char \*restrict *format*, ...);13072 int printf(const char \*restrict *format*, ...);13073 int snprintf(char \*restrict *s*, size\_t *n*,13074 const char \*restrict *format*, ...);13075 int sprintf(char \*restrict *s*, const char \*restrict *format*, ...);13076 **DESCRIPTION**

13077 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
 13078 conflict between the requirements described here and the ISO C standard is unintentional. This  
 13079 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

13080 The *fprintf()* function shall place output on the named output *stream*. The *printf()* function shall  
 13081 place output on the standard output stream *stdout*. The *sprintf()* function shall place output  
 13082 followed by the null byte, '\0', in consecutive bytes starting at *s*; it is the user's responsibility  
 13083 to ensure that enough space is available.

13084 The *snprintf()* function shall be equivalent to *sprintf()*, with the addition of the *n* argument  
 13085 which states the size of the buffer referred to by *s*. If *n* is zero, nothing shall be written and *s* may  
 13086 be a null pointer. Otherwise, output bytes beyond the *n*-1st shall be discarded instead of being  
 13087 written to the array, and a null byte is written at the end of the bytes actually written into the  
 13088 array.

13089 If copying takes place between objects that overlap as a result of a call to *sprintf()* or *snprintf()*,  
 13090 the results are undefined.

13091 Each of these functions converts, formats, and prints its arguments under control of the *format*.  
 13092 The *format* is a character string, beginning and ending in its initial shift state, if any. The *format* is  
 13093 composed of zero or more directives: *ordinary characters*, which are simply copied to the output  
 13094 stream, and *conversion specifications*, each of which shall result in the fetching of zero or more  
 13095 arguments. The results are undefined if there are insufficient arguments for the *format*. If the  
 13096 *format* is exhausted while arguments remain, the excess arguments shall be evaluated but are  
 13097 otherwise ignored.

13098 xsi Conversions can be applied to the *n*th argument after the *format* in the argument list, rather than  
 13099 to the next unused argument. In this case, the conversion specifier character % (see below) is  
 13100 replaced by the sequence "%n\$", where *n* is a decimal integer in the range [1,{NL\_ARGMAX}],  
 13101 giving the position of the argument in the argument list. This feature provides for the definition  
 13102 of format strings that select arguments in an order appropriate to specific languages (see the  
 13103 EXAMPLES section).

13104 The *format* can contain either numbered argument conversion specifications (that is, "%n\$" and  
 13105 "\*m\$"), or unnumbered argument conversion specifications (that is, % and \*), but not both. The  
 13106 only exception to this is that %% can be mixed with the "%n\$" form. The results of mixing  
 13107 numbered and unnumbered argument specifications in a *format* string are undefined. When  
 13108 numbered argument specifications are used, specifying the *N*th argument requires that all the  
 13109 leading arguments, from the first to the (*N*-1)th, are specified in the format string.

13110 In format strings containing the "%n\$" form of conversion specification, numbered arguments  
 13111 in the argument list can be referenced from the format string as many times as required.

13112 In format strings containing the % form of conversion specification, each argument in the  
 13113 argument list is used exactly once.

13114 CX All forms of the *fprintf()* functions allow for the insertion of a language-dependent radix character in the output string. The radix character is defined in the program's locale (category *LC\_NUMERIC*). In the POSIX locale, or in a locale where the radix character is not defined, the radix character shall default to a period ( '.' ).

13118 XSI Each conversion specification is introduced by the '%' character or by the character sequence "%n\$", after which the following appear in sequence:

- 13120 • Zero or more *flags* (in any order), which modify the meaning of the conversion specification.
- 13121 • An optional minimum *field width*. If the converted value has fewer bytes than the field width, it shall be padded with spaces by default on the left; it shall be padded on the right if the left-adjustment flag ('-'), described below, is given to the field width. The field width takes the form of an asterisk ('\*'), described below, or a decimal integer.
- 13122
- 13123
- 13124
- 13125 • An optional *precision* that gives the minimum number of digits to appear for the d, i, o, u, x, and X conversion specifiers; the number of digits to appear after the radix character for the a, A, e, E, f, and F conversion specifiers; the maximum number of significant digits for the g and G conversion specifiers; or the maximum number of bytes to be printed from a string in s and S conversion specifiers. The precision takes the form of a period ( '.' ) followed either by an asterisk ( '\*' ), described below, or an optional decimal digit string, where a null digit string is treated as zero. If a precision appears with any other conversion specifier, the behavior is undefined.
- 13126
- 13127
- 13128 XSI
- 13129
- 13130
- 13131
- 13132
- 13133 • An optional length modifier that specifies the size of the argument.
- 13134 • A *conversion specifier* character that indicates the type of conversion to be applied.

13135 A field width, or precision, or both, may be indicated by an asterisk ( '\*' ). In this case an argument of type *int* supplies the field width or precision. Applications shall ensure that arguments specifying field width, or precision, or both appear in that order before the argument, if any, to be converted. A negative field width is taken as a '-' flag followed by a positive field width. A negative precision is taken as if the precision were omitted. In format strings containing the "%n\$" form of a conversion specification, a field width or precision may be indicated by the sequence "\*m\$", where *m* is a decimal integer in the range [1,{NL\_ARGMAX}] giving the position in the argument list (after the format argument) of an integer argument containing the field width or precision, for example:

```
13144 printf("%1$d:%2$.*3$d:%4$.*3$d\n", hour, min, precision, sec);
```

13145 The flag characters and their meanings are:

- 13146 XSI ' The integer portion of the result of a decimal conversion (%i, %d, %u, %f, %F, %g, or %G) shall be formatted with thousands' grouping characters. For other conversions the behavior is undefined. The non-monetary grouping character is used.
- 13147
- 13148
- 13149 - The result of the conversion shall be left-justified within the field. The conversion is right-justified if this flag is not specified.
- 13150
- 13151 + The result of a signed conversion shall always begin with a sign ('+' or '-'). The conversion shall begin with a sign only when a negative value is converted if this flag is not specified.
- 13152
- 13153
- 13154 <space> If the first character of a signed conversion is not a sign or if a signed conversion results in no characters, a <space> shall be prefixed to the result. This means that if the <space> and '+' flags both appear, the <space> flag shall be ignored.
- 13155
- 13156
- 13157 # Specifies that the value is to be converted to an alternative form. For o conversion, it increases the precision (if necessary) to force the first digit of the result to be zero. For x
- 13158

13159 or X conversion specifiers, a non-zero result shall have 0x (or 0X) prefixed to it. For a, A,  
 13160 e, E, f, F, g, and G conversion specifiers, the result shall always contain a radix  
 13161 character, even if no digits follow the radix character. Without this flag, a radix  
 13162 character appears in the result of these conversions only if a digit follows it. For g and G  
 13163 conversion specifiers, trailing zeros shall *not* be removed from the result as they  
 13164 normally are. For other conversion specifiers, the behavior is undefined.

13165 0 For d, i, o, u, x, X, a, A, e, E, f, F, g, and G conversion specifiers, leading zeros  
 13166 (following any indication of sign or base) are used to pad to the field width; no space  
 13167 padding is performed. If the '0' and '-' flags both appear, the '0' flag is ignored. For  
 13168 d, i, o, u, x, and X conversion specifiers, if a precision is specified, the '0' flag is  
 13169 XSI ignored. If the '0' and '\'' flags both appear, the grouping characters are inserted  
 13170 before zero padding. For other conversions, the behavior is undefined.

13171 The length modifiers and their meanings are:

13172 hh Specifies that a following d, i, o, u, x, or X conversion specifier applies to a **signed char**  
 13173 or **unsigned char** argument (the argument will have been promoted according to the  
 13174 integer promotions, but its value shall be converted to **signed char** or **unsigned char**  
 13175 before printing); or that a following n conversion specifier applies to a pointer to a  
 13176 **signed char** argument.

13177 h Specifies that a following d, i, o, u, x, or X conversion specifier applies to a **short** or  
 13178 **unsigned short** argument (the argument will have been promoted according to the  
 13179 integer promotions, but its value shall be converted to **short** or **unsigned short** before  
 13180 printing); or that a following n conversion specifier applies to a pointer to a **short**  
 13181 argument.

13182 l (ell) Specifies that a following d, i, o, u, x, or X conversion specifier applies to a **long** or  
 13183 **unsigned long** argument; that a following n conversion specifier applies to a pointer to  
 13184 a **long** argument; that a following c conversion specifier applies to a **wint\_t** argument;  
 13185 that a following s conversion specifier applies to a pointer to a **wchar\_t** argument; or  
 13186 has no effect on a following a, A, e, E, f, F, g, or G conversion specifier.

13187 ll (ell-ell) Specifies that a following d, i, o, u, x, or X conversion specifier applies to a **long long** or  
 13188 **unsigned long long** argument; or that a following n conversion specifier applies to a  
 13189 pointer to a **long long** argument.

13191 j Specifies that a following d, i, o, u, x, or X conversion specifier applies to an **intmax\_t**  
 13192 or **uintmax\_t** argument; or that a following n conversion specifier applies to a pointer  
 13193 to an **intmax\_t** argument.

13194 z Specifies that a following d, i, o, u, x, or X conversion specifier applies to a **size\_t** or the  
 13195 corresponding signed integer type argument; or that a following n conversion specifier  
 13196 applies to a pointer to a signed integer type corresponding to **size\_t** argument.

13197 t Specifies that a following d, i, o, u, x, or X conversion specifier applies to a **ptrdiff\_t** or  
 13198 the corresponding **unsigned** type argument; or that a following n conversion specifier  
 13199 applies to a pointer to a **ptrdiff\_t** argument.

13200 L Specifies that a following a, A, e, E, f, F, g, or G conversion specifier applies to a **long**  
 13201 **double** argument.

13202 If a length modifier appears with any conversion specifier other than as specified above, the  
 13203 behavior is undefined.

13204		The conversion specifiers and their meanings are:
13205	d, i	The <b>int</b> argument shall be converted to a signed decimal in the style " $[-]ddd$ ". The
13206		precision specifies the minimum number of digits to appear; if the value being
13207		converted can be represented in fewer digits, it shall be expanded with leading zeros.
13208		The default precision is 1. The result of converting zero with an explicit precision of
13209		zero shall be no characters.
13210	o	The <b>unsigned</b> argument shall be converted to unsigned octal format in the style
13211		" $ddd$ ". The precision specifies the minimum number of digits to appear; if the value
13212		being converted can be represented in fewer digits, it shall be expanded with leading
13213		zeros. The default precision is 1. The result of converting zero with an explicit precision
13214		of zero shall be no characters.
13215	u	The <b>unsigned</b> argument shall be converted to unsigned decimal format in the style
13216		" $ddd$ ". The precision specifies the minimum number of digits to appear; if the value
13217		being converted can be represented in fewer digits, it shall be expanded with leading
13218		zeros. The default precision is 1. The result of converting zero with an explicit precision
13219		of zero shall be no characters.
13220	x	The <b>unsigned</b> argument shall be converted to unsigned hexadecimal format in the style
13221		" $ddd$ "; the letters "abcdef" are used. The precision specifies the minimum number
13222		of digits to appear; if the value being converted can be represented in fewer digits, it
13223		shall be expanded with leading zeros. The default precision is 1. The result of
13224		converting zero with an explicit precision of zero shall be no characters.
13225	X	Equivalent to the x conversion specifier, except that letters "ABCDEF" are used instead
13226		of "abcdef".
13227	f, F	The <b>double</b> argument shall be converted to decimal notation in the style
13228		" $[-]ddd.ddd$ ", where the number of digits after the radix character is equal to the
13229		precision specification. If the precision is missing, it shall be taken as 6; if the precision
13230		is explicitly zero and no '#' flag is present, no radix character shall appear. If a radix
13231		character appears, at least one digit appears before it. The low-order digit shall be
13232		rounded in an implementation-defined manner.
13233		A <b>double</b> argument representing an infinity shall be converted in one of the styles
13234		" $[-]inf$ " or " $[-]infinity$ "; which style is implementation-defined. A <b>double</b>
13235		argument representing a NaN shall be converted in one of the styles " $[-]nan(n-$
13236		<i>char-sequence</i> )"; or " $[-]nan$ " which style, and the meaning of any <i>n-char-sequence</i> ,
13237		is implementation-defined. The F conversion specifier produces "INF", "INFINITY",
13238		or "NAN" instead of "inf", "infinity", or "nan", respectively.
13239	e, E	The <b>double</b> argument shall be converted in the style " $[-]d.ddde±dd$ ", where there is
13240		one digit before the radix character (which is non-zero if the argument is non-zero) and
13241		the number of digits after it is equal to the precision; if the precision is missing, it shall
13242		be taken as 6; if the precision is zero and no '#' flag is present, no radix character shall
13243		appear. The low-order digit shall be rounded in an implementation-defined manner.
13244		The E conversion specifier shall produce a number with 'E' instead of 'e'
13245		introducing the exponent. The exponent shall always contain at least two digits. If the
13246		value is zero, the exponent shall be zero.
13247		A <b>double</b> argument representing an infinity or NaN shall be converted in the style of
13248		an f or F conversion specifier.
13249	g, G	The <b>double</b> argument shall be converted in the style f or e (or in the style E in the case
13250		of a G conversion specifier), with the precision specifying the number of significant

13251 digits. If an explicit precision is zero, it shall be taken as 1. The style used depends on |  
 13252 the value converted; style `e` (or `E`) shall be used only if the exponent resulting from |  
 13253 such a conversion is less than `-4` or greater than or equal to the precision. Trailing zeros |  
 13254 shall be removed from the fractional portion of the result; a radix character shall appear |  
 13255 only if it is followed by a digit.

13256 A **double** argument representing an infinity or NaN shall be converted in the style of |  
 13257 an `f` or `F` conversion specifier.

13258 a, A A **double** argument representing a floating-point number shall be converted in the |  
 13259 style "`[-]0xh.hhhhp±d`", where there is one hexadecimal digit (which shall be non- |  
 13260 zero if the argument is a normalized floating-point number and is otherwise |  
 13261 unspecified) before the decimal-point character and the number of hexadecimal digits |  
 13262 after it is equal to the precision; if the precision is missing and `FLT_RADIX` is a power |  
 13263 of 2, then the precision shall be sufficient for an exact representation of the value; if the |  
 13264 precision is missing and `FLT_RADIX` is not a power of 2, then the precision shall be |  
 13265 sufficient to distinguish values of type **double**, except that trailing zeros may be |  
 13266 omitted; if the precision is zero and the `'#'` flag is not specified, no decimal-point |  
 13267 character shall appear. The letters "`abcdef`" shall be used for a conversion and the |  
 13268 letters "`ABCDEF`" for A conversion. The A conversion specifier produces a number with |  
 13269 `'X'` and `'P'` instead of `'x'` and `'p'`. The exponent shall always contain at least one |  
 13270 digit, and only as many more digits as necessary to represent the decimal exponent of |  
 13271 2. If the value is zero, the exponent shall be zero.

13272 A **double** argument representing an infinity or NaN shall be converted in the style of |  
 13273 an `f` or `F` conversion specifier.

13274 c The **int** argument shall be converted to an **unsigned char**, and the resulting byte shall |  
 13275 be written.

13276 If an `l` (`ell`) qualifier is present, the **wint\_t** argument shall be converted as if by an `l` |  
 13277 conversion specification with no precision and an argument that points to a two- |  
 13278 element array of type **wchar\_t**, the first element of which contains the **wint\_t** argument |  
 13279 to the `l` conversion specification and the second element contains a null wide |  
 13280 character.

13281 s The argument shall be a pointer to an array of **char**. Bytes from the array shall be |  
 13282 written up to (but not including) any terminating null byte. If the precision is specified, |  
 13283 no more than that many bytes shall be written. If the precision is not specified or is |  
 13284 greater than the size of the array, the application shall ensure that the array contains a |  
 13285 null byte.

13286 If an `l` (`ell`) qualifier is present, the argument shall be a pointer to an array of type |  
 13287 **wchar\_t**. Wide characters from the array shall be converted to characters (each as if by |  
 13288 a call to the `wcrtomb()` function, with the conversion state described by an **mbstate\_t** |  
 13289 object initialized to zero before the first wide character is converted) up to and |  
 13290 including a terminating null wide character. The resulting characters shall be written |  
 13291 up to (but not including) the terminating null character (byte). If no precision is |  
 13292 specified, the application shall ensure that the array contains a null wide character. If a |  
 13293 precision is specified, no more than that many characters (bytes) shall be written |  
 13294 (including shift sequences, if any), and the array shall contain a null wide character if, |  
 13295 to equal the character sequence length given by the precision, the function would need |  
 13296 to access a wide character one past the end of the array. In no case shall a partial |  
 13297 character written.

13298	p	The argument shall be a pointer to <b>void</b> . The value of the pointer is converted to a sequence of printable characters, in an implementation-defined manner.	
13299			
13300	n	The argument shall be a pointer to an integer into which is written the number of bytes written to the output so far by this call to one of the <i>fprintf()</i> functions. No argument is converted.	
13301			
13302			
13303	XSI C	Equivalent to <code>lc</code> .	
13304	XSI S	Equivalent to <code>ls</code> .	
13305	%	Print a '%' character; no argument is converted. The complete conversion specification shall be <code>%%</code> .	
13306			
13307		If a conversion specification does not match one of the above forms, the behavior is undefined. If any argument is not the correct type for the corresponding conversion specification, the behavior is undefined.	
13308			
13309			
13310		In no case shall a nonexistent or small field width cause truncation of a field; if the result of a conversion is wider than the field width, the field shall be expanded to contain the conversion result. Characters generated by <i>fprintf()</i> and <i>printf()</i> are printed as if <i>fputc()</i> had been called.	
13311			
13312			
13313		For the <code>a</code> and <code>A</code> conversion specifiers, if <code>FLT_RADIX</code> is a power of 2, the value shall be correctly rounded to a hexadecimal floating number with the given precision.	
13314			
13315		For <code>a</code> and <code>A</code> conversions, if <code>FLT_RADIX</code> is not a power of 2 and the result is not exactly representable in the given precision, the result should be one of the two adjacent numbers in hexadecimal floating style with the given precision, with the extra stipulation that the error should have a correct sign for the current rounding direction.	
13316			
13317			
13318			
13319		For the <code>e</code> , <code>E</code> , <code>f</code> , <code>F</code> , <code>g</code> , and <code>G</code> conversion specifiers, if the number of significant decimal digits is at most <code>DECIMAL_DIG</code> , then the result should be correctly rounded. If the number of significant decimal digits is more than <code>DECIMAL_DIG</code> but the source value is exactly representable with <code>DECIMAL_DIG</code> digits, then the result should be an exact representation with trailing zeros. Otherwise, the source value is bounded by two adjacent decimal strings $L < U$ , both having <code>DECIMAL_DIG</code> significant digits; the value of the resultant decimal string $D$ should satisfy $L \leq D \leq U$ , with the extra stipulation that the error should have a correct sign for the current rounding direction.	
13320			
13321			
13322			
13323			
13324			
13325			
13326			
13327	CX	The <code>st_ctime</code> and <code>st_mtime</code> fields of the file shall be marked for update between the call to a successful execution of <i>fprintf()</i> or <i>printf()</i> and the next successful completion of a call to <i>fflush()</i> or <i>fclose()</i> on the same stream or a call to <i>exit()</i> or <i>abort()</i> .	
13328			
13329			
13330	<b>RETURN VALUE</b>		
13331		Upon successful completion, the <i>fprintf()</i> and <i>printf()</i> functions shall return the number of bytes transmitted.	
13332			
13333		Upon successful completion, the <i>sprintf()</i> function shall return the number of bytes written to <i>s</i> , excluding the terminating null byte.	
13334			
13335		Upon successful completion, the <i>snprintf()</i> function shall return the number of bytes that would be written to <i>s</i> had <i>n</i> been sufficiently large excluding the terminating null byte.	
13336			
13337		If an output error was encountered, these functions shall return a negative value.	
13338		If the value of <i>n</i> is zero on a call to <i>snprintf()</i> , nothing shall be written, the number of bytes that would have been written had <i>n</i> been sufficiently large excluding the terminating null shall be returned, and <i>s</i> may be a null pointer.	
13339			
13340			

13341 **ERRORS**

13342 For the conditions under which *fprintf()* and *printf()* fail and may fail, refer to *fputc()* or  
13343 *fputwc()*.

13344 In addition, all forms of *fprintf()* may fail if:

13345 XSI [EILSEQ] A wide-character code that does not correspond to a valid character has been  
13346 detected.

13347 XSI [EINVAL] There are insufficient arguments.

13348 The *printf()* and *fprintf()* functions may fail if:

13349 XSI [ENOMEM] Insufficient storage space is available.

13350 The *snprintf()* function shall fail if:

13351 XSI [EOVERFLOW] The value of *n* is greater than {INT\_MAX} or the number of bytes needed to  
13352 hold the output excluding the terminating null is greater than {INT\_MAX}.

13353 **EXAMPLES**13354 **Printing Language-Independent Date and Time**

13355 The following statement can be used to print date and time using a language-independent  
13356 format:

```
13357 printf(format, weekday, month, day, hour, min);
```

13358 For American usage, *format* could be a pointer to the following string:

```
13359 "%s, %s %d, %d:%.2d\n"
```

13360 This example would produce the following message:

```
13361 Sunday, July 3, 10:02
```

13362 For German usage, *format* could be a pointer to the following string:

```
13363 "%1$s, %3$d. %2$s, %4$d:%5$.2d\n"
```

13364 This definition of *format* would produce the following message:

```
13365 Sonntag, 3. Juli, 10:02
```

13366 **Printing File Information**

13367 The following example prints information about the type, permissions, and number of links of a  
13368 specific file in a directory.

13369 The first two calls to *printf()* use data decoded from a previous *stat()* call. The user-defined  
13370 *strperm()* function shall return a string similar to the one at the beginning of the output for the  
13371 following command:

```
13372 ls -l
```

13373 The next call to *printf()* outputs the owner's name if it is found using *getpwuid()*; the *getpwuid()*  
13374 function shall return a **passwd** structure from which the name of the user is extracted. If the user  
13375 name is not found, the program instead prints out the numeric value of the user ID.

13376 The next call prints out the group name if it is found using *getgrgid()*; *getgrgid()* is very similar to  
13377 *getpwuid()* except that it shall return group information based on the group number. Once  
13378 again, if the group is not found, the program prints the numeric value of the group for the entry.

```

13379     The final call to printf() prints the size of the file.
13380     #include <stdio.h>
13381     #include <sys/types.h>
13382     #include <pwd.h>
13383     #include <grp.h>
13384     char *strperm (mode_t);
13385     ...
13386     struct stat statbuf;
13387     struct passwd *pwd;
13388     struct group *grp;
13389     ...
13390     printf("%10.10s", strperm (statbuf.st_mode));
13391     printf("%4d", statbuf.st_nlink);
13392     if ((pwd = getpwuid(statbuf.st_uid)) != NULL)
13393         printf(" %-8.8s", pwd->pw_name);
13394     else
13395         printf(" %-8ld", (long) statbuf.st_uid);
13396     if ((grp = getgrgid(statbuf.st_gid)) != NULL)
13397         printf(" %-8.8s", grp->gr_name);
13398     else
13399         printf(" %-8ld", (long) statbuf.st_gid);
13400     printf("%9jd", (intmax_t) statbuf.st_size);
13401     ...

```

#### 13402 **Printing a Localized Date String**

13403 The following example gets a localized date string. The *nl\_langinfo()* function shall return the  
 13404 localized date string, which specifies the order and layout of the date. The *strftime()* function  
 13405 takes this information and, using the **tm** structure for values, places the date and time  
 13406 information into *datestring*. The *printf()* function then outputs *datestring* and the name of the  
 13407 entry.

```

13408     #include <stdio.h>
13409     #include <time.h>
13410     #include <langinfo.h>
13411     ...
13412     struct dirent *dp;
13413     struct tm *tm;
13414     char datestring[256];
13415     ...
13416     strftime(datestring, sizeof(datestring), nl_langinfo (D_T_FMT), tm);
13417     printf(" %s %s\n", datestring, dp->d_name);
13418     ...

```

13419 **Printing Error Information**

13420 The following example uses *fprintf()* to write error information to standard error.

13421 In the first group of calls, the program tries to open the password lock file named **LOCKFILE**. If  
 13422 the file already exists, this is an error, as indicated by the **O\_EXCL** flag on the *open()* function. If  
 13423 the call fails, the program assumes that someone else is updating the password file, and the  
 13424 program exits.

13425 The next group of calls saves a new password file as the current password file by creating a link  
 13426 between **LOCKFILE** and the new password file **PASSWDFILE**.

```

13427 #include <sys/types.h>
13428 #include <sys/stat.h>
13429 #include <fcntl.h>
13430 #include <stdio.h>
13431 #include <stdlib.h>
13432 #include <unistd.h>
13433 #include <string.h>
13434 #include <errno.h>

13435 #define LOCKFILE "/etc/ptmp"
13436 #define PASSWDFILE "/etc/passwd"
13437 ...
13438 int pfd;
13439 ...
13440 if ((pfd = open(LOCKFILE, O_WRONLY | O_CREAT | O_EXCL,
13441               S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH)) == -1)
13442 {
13443     fprintf(stderr, "Cannot open /etc/ptmp. Try again later.\n");
13444     exit(1);
13445 }
13446 ...
13447 if (link(LOCKFILE, PASSWDFILE) == -1) {
13448     fprintf(stderr, "Link error: %s\n", strerror(errno));
13449     exit(1);
13450 }
13451 ...

```

13452 **Printing Usage Information**

13453 The following example checks to make sure the program has the necessary arguments, and uses  
 13454 *fprintf()* to print usage information if the expected number of arguments is not present.

```

13455 #include <stdio.h>
13456 #include <stdlib.h>
13457 ...
13458 char *Options = "hdbt1";
13459 ...
13460 if (argc < 2) {
13461     fprintf(stderr, "Usage: %s -%s <file\n", argv[0], Options); exit(1);
13462 }
13463 ...

```

13464       **Formatting a Decimal String**

13465       The following example prints a key and data pair on *stdout*. Note use of the '\*' (asterisk) in the  
13466       format string; this ensures the correct number of decimal places for the element based on the  
13467       number of elements requested.

```
13468       #include <stdio.h>
13469       ...
13470       long i;
13471       char *keyst;
13472       int elementlen, len;
13473       ...
13474       while (len < elementlen) {
13475       ...
13476           printf("%s Element%0*ld\n", keyst, elementlen, i);
13477       ...
13478       }
```

13479       **Creating a Filename**

13480       The following example creates a filename using information from a previous *getpwnam()*  
13481       function that returned the HOME directory of the user.

```
13482       #include <stdio.h>
13483       #include <sys/types.h>
13484       #include <unistd.h>
13485       ...
13486       char filename[PATH_MAX+1];
13487       struct passwd *pw;
13488       ...
13489       sprintf(filename, "%s/%d.out", pw->pw_dir, getpid());
13490       ...
```

13491       **Reporting an Event**

13492       The following example loops until an event has timed out. The *pause()* function waits forever  
13493       unless it receives a signal. The *fprintf()* statement should never occur due to the possible return  
13494       values of *pause()*.

```
13495       #include <stdio.h>
13496       #include <unistd.h>
13497       #include <string.h>
13498       #include <errno.h>
13499       ...
13500       while (!event_complete) {
13501       ...
13502           if (pause() != -1 || errno != EINTR)
13503               fprintf(stderr, "pause: unknown error: %s\n", strerror(errno));
13504       }
13505       ...
```

13506 **Printing Monetary Information**

13507 The following example uses *strfmon()* to convert a number and store it as a formatted monetary  
 13508 string named *convbuf*. If the first number is printed, the program prints the format and the  
 13509 description; otherwise, it just prints the number.

```

13510 #include <monetary.h>
13511 #include <stdio.h>
13512 ...
13513 struct tblfmt {
13514     char *format;
13515     char *description;
13516 };
13517 struct tblfmt table[] = {
13518     { "%n", "default formatting" },
13519     { "%11n", "right align within an 11 character field" },
13520     { "%#5n", "aligned columns for values up to 99,999" },
13521     { "%=*#5n", "specify a fill character" },
13522     { "%=0#5n", "fill characters do not use grouping" },
13523     { "%^#5n", "disable the grouping separator" },
13524     { "%^#5.0n", "round off to whole units" },
13525     { "%^#5.4n", "increase the precision" },
13526     { "%(#5n", "use an alternative pos/neg style" },
13527     { "%!(#5n", "disable the currency symbol" },
13528 };
13529 ...
13530 float input[3];
13531 int i, j;
13532 char convbuf[100];
13533 ...
13534 strfmon(convbuf, sizeof(convbuf), table[i].format, input[j]);
13535
13536 if (j == 0) {
13537     printf("%s%s%s\n", table[i].format,
13538         convbuf, table[i].description);
13539 }
13540 else {
13541     printf("%s\n", convbuf);
13542 }
13543 ...

```

13543 **APPLICATION USAGE**

13544 If the application calling *fprintf()* has any objects of type **wint\_t** or **wchar\_t**, it must also include  
 13545 the **<wchar.h>** header to have these objects defined.

13546 **RATIONALE**

13547 None.

13548 **FUTURE DIRECTIONS**

13549 None.

13550 **SEE ALSO**

13551 *fputc()*, *fscanf()*, *setlocale()*, *wcrtomb()*, the Base Definitions volume of IEEE Std 1003.1-200x,  
 13552 **<stdio.h>**, **<wchar.h>**, the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 7, Locale

13553 **CHANGE HISTORY**

13554 First released in Issue 1. Derived from Issue 1 of the SVID.

13555 **Issue 5**

13556 Aligned with ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, the `l` (ell) qualifier can  
13557 now be used with `c` and `s` conversion specifiers.

13558 The *snprintf()* function is new in Issue 5.

13559 **Issue 6**

13560 Extensions beyond the ISO C standard are now marked.

13561 The DESCRIPTION is updated to avoid use of the term “must” for application requirements. |

13562 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

13563 • The prototypes for *fprintf()*, *printf()*, *snprintf()*, and *sprintf()* are updated, and the XSI  
13564 shading is removed from *snprintf()*.

13565 • The description of *snprintf()* is aligned with the ISO C standard. Note that this supersedes |  
13566 the *snprintf()* description in The Open Group Base Resolution bwg98-006, which changed the |  
13567 behavior from Issue 5. |

13568 • The DESCRIPTION is updated. |

13569 The DESCRIPTION is updated to use the terms “conversion specifier” and “conversion  
13570 specification” consistently. |

13571 ISO/IEC 9899:1999 standard, Technical Corrigendum No. 1 is incorporated. |

13572 **NAME**

13573 fputc — put a byte on a stream

13574 **SYNOPSIS**

13575 #include &lt;stdio.h&gt;

13576 int fputc(int *c*, FILE \**stream*);13577 **DESCRIPTION**

13578 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 13579 conflict between the requirements described here and the ISO C standard is unintentional. This  
 13580 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

13581 The *fputc()* function shall write the byte specified by *c* (converted to an **unsigned char**) to the  
 13582 output stream pointed to by *stream*, at the position indicated by the associated file-position  
 13583 indicator for the stream (if defined), and shall advance the indicator appropriately. If the file  
 13584 cannot support positioning requests, or if the stream was opened with append mode, the byte  
 13585 shall be appended to the output stream.

13586 CX The *st\_ctime* and *st\_mtime* fields of the file shall be marked for update between the successful  
 13587 execution of *fputc()* and the next successful completion of a call to *fflush()* or *fclose()* on the same  
 13588 stream or a call to *exit()* or *abort()*.

13589 **RETURN VALUE**

13590 Upon successful completion, *fputc()* shall return the value it has written. Otherwise, it shall  
 13591 CX return EOF, the error indicator for the stream shall be set, and *errno* shall be set to indicate the  
 13592 error.

13593 **ERRORS**

13594 The *fputc()* function shall fail if either the *stream* is unbuffered or the *stream*'s buffer needs to be  
 13595 flushed, and:

13596 CX [EAGAIN] The O\_NONBLOCK flag is set for the file descriptor underlying *stream* and the  
 13597 process would be delayed in the write operation.

13598 CX [EBADF] The file descriptor underlying *stream* is not a valid file descriptor open for  
 13599 writing.

13600 CX [EFBIG] An attempt was made to write to a file that exceeds the maximum file size.

13601 XSI [EFBIG] An attempt was made to write to a file that exceeds the process' file size limit.

13602 CX [EFBIG] The file is a regular file and an attempt was made to write at or beyond the  
 13603 offset maximum.

13604 CX [EINTR] The write operation was terminated due to the receipt of a signal, and no data  
 13605 was transferred.

13606 CX [EIO] A physical I/O error has occurred, or the process is a member of a  
 13607 background process group attempting to write to its controlling terminal,  
 13608 TOSTOP is set, the process is neither ignoring nor blocking SIGTTOU, and the  
 13609 process group of the process is orphaned. This error may also be returned  
 13610 under implementation-defined conditions.

13611 CX [ENOSPC] There was no free space remaining on the device containing the file.

13612 CX [EPIPE] An attempt is made to write to a pipe or FIFO that is not open for reading by  
 13613 any process. A SIGPIPE signal shall also be sent to the thread.

13614 The *fputc()* function may fail if:

13615 CX [ENOMEM] Insufficient storage space is available.

13616 CX [ENXIO] A request was made of a nonexistent device, or the request was outside the  
13617 capabilities of the device.

13618 **EXAMPLES**

13619 None.

13620 **APPLICATION USAGE**

13621 None.

13622 **RATIONALE**

13623 None.

13624 **FUTURE DIRECTIONS**

13625 None.

13626 **SEE ALSO**

13627 *ferror()*, *fopen()*, *getrlimit()*, *putc()*, *puts()*, *setbuf()*, *ulimit()*, the Base Definitions volume of  
13628 IEEE Std 1003.1-200x, <stdio.h>

13629 **CHANGE HISTORY**

13630 First released in Issue 1. Derived from Issue 1 of the SVID.

13631 **Issue 5**

13632 Large File Summit extensions are added.

13633 **Issue 6**

13634 Extensions beyond the ISO C standard are now marked.

13635 The following new requirements on POSIX implementations derive from alignment with the  
13636 Single UNIX Specification:

- 13637
- The [EIO] and [EFBIG] mandatory error conditions are added.
- 13638
- The [ENOMEM] and [ENXIO] optional error conditions are added.

13639 **NAME**

13640 fputs — put a string on a stream

13641 **SYNOPSIS**

13642 #include &lt;stdio.h&gt;

13643 int fputs(const char \*restrict *s*, FILE \*restrict *stream*);13644 **DESCRIPTION**

13645 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
 13646 conflict between the requirements described here and the ISO C standard is unintentional. This  
 13647 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

13648 The *fputs()* function shall write the null-terminated string pointed to by *s* to the stream pointed  
 13649 to by *stream*. The terminating null byte shall not be written.

13650 cx The *st\_ctime* and *st\_mtime* fields of the file shall be marked for update between the successful  
 13651 execution of *fputs()* and the next successful completion of a call to *fflush()* or *fclose()* on the same  
 13652 stream or a call to *exit()* or *abort()*.

13653 **RETURN VALUE**

13654 Upon successful completion, *fputs()* shall return a non-negative number. Otherwise, it shall  
 13655 cx return EOF, set an error indicator for the stream, and set *errno* to indicate the error.

13656 **ERRORS**13657 Refer to *fputc()*.13658 **EXAMPLES**13659 **Printing to Standard Output**

13660 The following example gets the current time, converts it to a string using *localtime()* and  
 13661 *asctime()*, and prints it to standard output using *fputs()*. It then prints the number of minutes to  
 13662 an event for which it is waiting.

```

13663 #include <time.h>
13664 #include <stdio.h>
13665 ...
13666 time_t now;
13667 int minutes_to_event;
13668 ...
13669 time(&now);
13670 printf("The time is ");
13671 fputs(asctime(localtime(&now)), stdout);
13672 printf("There are still %d minutes to the event.\n",
13673       minutes_to_event);
13674 ...

```

13675 **APPLICATION USAGE**13676 The *puts()* function appends a <newline> while *fputs()* does not.13677 **RATIONALE**

13678 None.

13679 **FUTURE DIRECTIONS**

13680 None.

13681 **SEE ALSO**

13682        *fopen()*, *putc()*, *puts()*, the Base Definitions volume of IEEE Std 1003.1-200x, <**stdio.h**>

13683 **CHANGE HISTORY**

13684        First released in Issue 1. Derived from Issue 1 of the SVID.

13685 **Issue 6**

13686        Extensions beyond the ISO C standard are now marked.

13687        The *fputs()* prototype is updated for alignment with the ISO/IEC 9899: 1999 standard.

13688 **NAME**13689 `fputc` — put a wide-character code on a stream13690 **SYNOPSIS**13691 `#include <stdio.h>`13692 `#include <wchar.h>`13693 `wint_t fputc(wchar_t wc, FILE *stream);`13694 **DESCRIPTION**

13695 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 13696 conflict between the requirements described here and the ISO C standard is unintentional. This  
 13697 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

13698 The `fputc()` function shall write the character corresponding to the wide-character code `wc` to  
 13699 the output stream pointed to by `stream`, at the position indicated by the associated file-position  
 13700 indicator for the stream (if defined), and advances the indicator appropriately. If the file cannot  
 13701 support positioning requests, or if the stream was opened with append mode, the character is  
 13702 appended to the output stream. If an error occurs while writing the character, the shift state of  
 13703 the output file is left in an undefined state.

13704 CX The `st_ctime` and `st_mtime` fields of the file shall be marked for update between the successful  
 13705 execution of `fputc()` and the next successful completion of a call to `fflush()` or `fclose()` on the  
 13706 same stream or a call to `exit()` or `abort()`.

13707 **RETURN VALUE**

13708 Upon successful completion, `fputc()` shall return `wc`. Otherwise, it shall return WEOF, the error  
 13709 CX indicator for the stream shall be set set, and `errno` shall be set to indicate the error.

13710 **ERRORS**

13711 The `fputc()` function shall fail if either the stream is unbuffered or data in the `stream`'s buffer  
 13712 needs to be written, and:

13713 CX [EAGAIN] The O\_NONBLOCK flag is set for the file descriptor underlying `stream` and the  
 13714 process would be delayed in the write operation.

13715 CX [EBADF] The file descriptor underlying `stream` is not a valid file descriptor open for  
 13716 writing.

13717 CX [EFBIG] An attempt was made to write to a file that exceeds the maximum file size or  
 13718 the process' file size limit.

13719 CX [EFBIG] The file is a regular file and an attempt was made to write at or beyond the  
 13720 offset maximum associated with the corresponding stream.

13721 [EILSEQ] The wide-character code `wc` does not correspond to a valid character.

13722 CX [EINTR] The write operation was terminated due to the receipt of a signal, and no data  
 13723 was transferred.

13724 CX [EIO] A physical I/O error has occurred, or the process is a member of a  
 13725 background process group attempting to write to its controlling terminal,  
 13726 TOSTOP is set, the process is neither ignoring nor blocking SIGTTOU, and the  
 13727 process group of the process is orphaned. This error may also be returned  
 13728 under implementation-defined conditions.

13729 CX [ENOSPC] There was no free space remaining on the device containing the file.

13730 CX [EPIPE] An attempt is made to write to a pipe or FIFO that is not open for reading by  
 13731 any process. A SIGPIPE signal shall also be sent to the thread.

13732 The *fputc()* function may fail if:

13733 CX [ENOMEM] Insufficient storage space is available.

13734 CX [ENXIO] A request was made of a nonexistent device, or the request was outside the  
13735 capabilities of the device.

#### 13736 EXAMPLES

13737 None.

#### 13738 APPLICATION USAGE

13739 None.

#### 13740 RATIONALE

13741 None.

#### 13742 FUTURE DIRECTIONS

13743 None.

#### 13744 SEE ALSO

13745 *ferror()*, *fopen()*, *setbuf()*, *ulimit()*, the Base Definitions volume of IEEE Std 1003.1-200x,  
13746 `<stdio.h>`, `<wchar.h>`

#### 13747 CHANGE HISTORY

13748 First released in Issue 4. Derived from the MSE working draft.

#### 13749 Issue 5

13750 Aligned with ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, the type of argument *wc*  
13751 is changed from `wint_t` to `wchar_t`.

13752 The Optional Header (OH) marking is removed from `<stdio.h>`.

13753 Large File Summit extensions are added.

#### 13754 Issue 6

13755 Extensions beyond the ISO C standard are now marked.

13756 The following new requirements on POSIX implementations derive from alignment with the  
13757 Single UNIX Specification:

- 13758 • The [EFBIG] and [EIO] mandatory error conditions are added.
- 13759 • The [ENOMEM] and [ENXIO] optional error conditions are added.

13760 **NAME**

13761 fputws — put a wide-character string on a stream

13762 **SYNOPSIS**

13763 #include &lt;stdio.h&gt;

13764 #include &lt;wchar.h&gt;

13765 int fputws(const wchar\_t \*restrict ws, FILE \*restrict stream);

13766 **DESCRIPTION**

13767 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 13768 conflict between the requirements described here and the ISO C standard is unintentional. This  
 13769 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

13770 The *fputws()* function shall write a character string corresponding to the (null-terminated)  
 13771 wide-character string pointed to by *ws* to the stream pointed to by *stream*. No character  
 13772 corresponding to the terminating null wide-character code shall be written.

13773 CX The *st\_ctime* and *st\_mtime* fields of the file shall be marked for update between the successful  
 13774 execution of *fputws()* and the next successful completion of a call to *flush()* or *fclose()* on the  
 13775 same stream or a call to *exit()* or *abort()*.

13776 **RETURN VALUE**

13777 Upon successful completion, *fputws()* shall return a non-negative number. Otherwise, it shall  
 13778 CX return  $-1$ , set an error indicator for the stream, and set *errno* to indicate the error.

13779 **ERRORS**13780 Refer to *fputwc()*.13781 **EXAMPLES**

13782 None.

13783 **APPLICATION USAGE**13784 The *fputws()* function does not append a <newline>.13785 **RATIONALE**

13786 None.

13787 **FUTURE DIRECTIONS**

13788 None.

13789 **SEE ALSO**13790 *fopen()*, the Base Definitions volume of IEEE Std 1003.1-200x, <stdio.h>, <wchar.h>13791 **CHANGE HISTORY**

13792 First released in Issue 4. Derived from the MSE working draft.

13793 **Issue 5**

13794 The Optional Header (OH) marking is removed from &lt;stdio.h&gt;.

13795 **Issue 6**

13796 Extensions beyond the ISO C standard are now marked.

13797 The *fputws()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

## 13798 NAME

13799 fread — binary input

## 13800 SYNOPSIS

13801 #include &lt;stdio.h&gt;

```
13802 size_t fread(void *restrict ptr, size_t size, size_t nitems,
13803 FILE *restrict stream);
```

## 13804 DESCRIPTION

13805 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 13806 conflict between the requirements described here and the ISO C standard is unintentional. This  
 13807 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

13808 The *fread()* function shall read into the array pointed to by *ptr* up to *nitems* elements whose size  
 13809 is specified by *size* in bytes, from the stream pointed to by *stream*. For each object, *size* calls shall  
 13810 be made to the *fgetc()* function and the results stored, in the order read, in an array of **unsigned**  
 13811 **char** exactly overlaying the object. The file position indicator for the stream (if defined) shall be  
 13812 advanced by the number of bytes successfully read. If an error occurs, the resulting value of the  
 13813 file position indicator for the stream is unspecified. If a partial element is read, its value is  
 13814 unspecified.

13815 CX The *fread()* function may mark the *st\_atime* field of the file associated with *stream* for update. The  
 13816 *st\_atime* field shall be marked for update by the first successful execution of *fgetc()*, *fgets()*,  
 13817 *fgetwc()*, *fgetws()*, *fread()*, *fscanf()*, *getc()*, *getchar()*, *gets()*, or *scanf()* using *stream* that returns  
 13818 data not supplied by a prior call to *ungetc()* or *ungetwc()*.

## 13819 RETURN VALUE

13820 Upon successful completion, *fread()* shall return the number of elements successfully read which  
 13821 is less than *nitems* only if a read error or end-of-file is encountered. If *size* or *nitems* is 0, *fread()*  
 13822 shall return 0 and the contents of the array and the state of the stream remain unchanged.  
 13823 CX Otherwise, if a read error occurs, the error indicator for the stream shall be set, and *errno* shall be  
 13824 set to indicate the error.

## 13825 ERRORS

13826 Refer to *fgetc()*.

## 13827 EXAMPLES

13828 **Reading from a Stream**13829 The following example reads a single element from the *fp* stream into the array pointed to by *buf*.

```
13830 #include <stdio.h>
13831 ...
13832 size_t bytes_read;
13833 char buf[100];
13834 FILE *fp;
13835 ...
13836 bytes_read = fread(buf, sizeof(buf), 1, fp);
13837 ...
```

## 13838 APPLICATION USAGE

13839 The *ferror()* or *feof()* functions must be used to distinguish between an error condition and an  
 13840 end-of-file condition.

13841 Because of possible differences in element length and byte ordering, files written using *fwrite()*  
 13842 are application-dependent, and possibly cannot be read using *fread()* by a different application

13843 or by the same application on a different processor.

13844 **RATIONALE**

13845 None.

13846 **FUTURE DIRECTIONS**

13847 None.

13848 **SEE ALSO**

13849 *feof()*, *ferror()*, *fgetc()*, *fopen()*, *getc()*, *gets()*, *scanf()*, the Base Definitions volume of  
13850 IEEE Std 1003.1-200x, <**stdio.h**>

13851 **CHANGE HISTORY**

13852 First released in Issue 1. Derived from Issue 1 of the SVID.

13853 **Issue 6**

13854 Extensions beyond the ISO C standard are now marked.

13855 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 13856
- The *fread()* prototype is updated.
- 13857
- The DESCRIPTION is updated to describe how the bytes from a call to *fgetc()* are stored.

13858 **NAME**

13859 free — free allocated memory

13860 **SYNOPSIS**

13861 #include &lt;stdlib.h&gt;

13862 void free(void \*ptr);

13863 **DESCRIPTION**

13864 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
13865 conflict between the requirements described here and the ISO C standard is unintentional. This  
13866 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

13867 The *free()* function shall cause the space pointed to by *ptr* to be deallocated; that is, made  
13868 available for further allocation. If *ptr* is a null pointer, no action shall occur. Otherwise, if the  
13869 ADV argument does not match a pointer earlier returned by the *calloc()*, *malloc()*, *posix\_memalign()*,  
13870 XSI *realloc()*, or *strdup()*, function, or if the space has been deallocated by a call to *free()* or *realloc()*,  
13871 the behavior is undefined.

13872 Any use of a pointer that refers to freed space results in undefined behavior.

13873 **RETURN VALUE**13874 The *free()* function shall not return a value.13875 **ERRORS**

13876 No errors are defined.

13877 **EXAMPLES**

13878 None.

13879 **APPLICATION USAGE**

13880 There is now no requirement for the implementation to support the inclusion of &lt;malloc.h&gt;.

13881 **RATIONALE**

13882 None.

13883 **FUTURE DIRECTIONS**

13884 None.

13885 **SEE ALSO**13886 *calloc()*, *malloc()*, *realloc()*, the Base Definitions volume of IEEE Std 1003.1-200x, <stdlib.h>13887 **CHANGE HISTORY**

13888 First released in Issue 1. Derived from Issue 1 of the SVID.

13889 **Issue 6**13890 Reference to the *valloc()* function is removed.

## 13891 NAME

13892 freeaddrinfo, getaddrinfo — get address information

## 13893 SYNOPSIS

```
13894 #include <sys/socket.h>
13895 #include <netdb.h>

13896 void freeaddrinfo(struct addrinfo *ai);
13897 int getaddrinfo(const char *restrict nodename,
13898                const char *restrict servname,
13899                const struct addrinfo *restrict hints,
13900                struct addrinfo **restrict res);
```

## 13901 DESCRIPTION

13902 The *freeaddrinfo()* function shall free one or more **addrinfo** structures returned by *getaddrinfo()*, |  
 13903 along with any additional storage associated with those structures. If the *ai\_next* field of the |  
 13904 structure is not null, the entire list of structures shall be freed. The *freeaddrinfo()* function shall |  
 13905 support the freeing of arbitrary sublists of an **addrinfo** list originally returned by *getaddrinfo()*.

13906 The *getaddrinfo()* function shall translate the name of a service location (for example, a host |  
 13907 name) and/or a service name and shall return a set of socket addresses and associated |  
 13908 information to be used in creating a socket with which to address the specified service.

13909 The *freeaddrinfo()* and *getaddrinfo()* functions shall be thread-safe.

13910 The *nodename* and *servname* arguments are either null pointers or pointers to null-terminated |  
 13911 strings. One or both of these two arguments shall be supplied by the application as a non-null |  
 13912 pointer.

13913 The format of a valid name depends on the address family or families. If a specific family is not |  
 13914 given and the name could be interpreted as valid within multiple supported families, the |  
 13915 implementation shall attempt to resolve the name in all supported families and, in absence of |  
 13916 errors, one or more results shall be returned.

13917 If the *nodename* argument is not null, it can be a descriptive name or can be an address string. If |  
 13918 IP6 the specified address family is AF\_INET, AF\_INET6, or AF\_UNSPEC, valid descriptive names |  
 13919 include host names. If the specified address family is AF\_INET or AF\_UNSPEC, address strings |  
 13920 using Internet standard dot notation as specified in *inet\_addr()* are valid.

13921 IP6 If the specified address family is AF\_INET6 or AF\_UNSPEC, standard IPv6 text forms described |  
 13922 in *inet\_ntop()* are valid.

13923 If *nodename* is not null, the requested service location is named by *nodename*; otherwise, the |  
 13924 requested service location is local to the caller.

13925 If *servname* is null, the call shall return network-level addresses for the specified *nodename*. If |  
 13926 *servname* is not null, it is a null-terminated character string identifying the requested service. This |  
 13927 can be either a descriptive name or a numeric representation suitable for use with the address |  
 13928 IP6 family or families. If the specified address family is AF\_INET, AF\_INET6, or AF\_UNSPEC, the |  
 13929 service can be specified as a string specifying a decimal port number.

13930 If the *hints* argument is not null, it refers to a structure containing input values that may direct |  
 13931 the operation by providing options and by limiting the returned information to a specific socket |  
 13932 type, address family and/or protocol. In this *hints* structure every member other than *ai\_flags*, |  
 13933 *ai\_family*, *ai\_socktype*, and *ai\_protocol* shall be set to zero or a null pointer. A value of |  
 13934 AF\_UNSPEC for *ai\_family* means that the caller shall accept any address family. A value of zero |  
 13935 for *ai\_socktype* means that the caller shall accept any socket type. A value of zero for *ai\_protocol* |  
 13936 means that the caller shall accept any protocol. If *hints* is a null pointer, the behavior shall be as if

13937 it referred to a structure containing the value zero for the *ai\_flags*, *ai\_socktype*, and *ai\_protocol*  
13938 fields, and AF\_UNSPEC for the *ai\_family* field.

13939 The *ai\_flags* field to which the *hints* parameter points shall be set to zero or be the bitwise-  
13940 inclusive OR of one or more of the values AI\_PASSIVE, AI\_CANONNAME,  
13941 AI\_NUMERICHOST, and AI\_NUMERICSERV.

13942 If the AI\_PASSIVE flag is specified, the returned address information shall be suitable for use in  
13943 binding a socket for accepting incoming connections for the specified service. In this case, if the  
13944 *nodename* argument is null, then the IP address portion of the socket address structure shall be  
13945 set to INADDR\_ANY for an IPv4 address or IN6ADDR\_ANY\_INIT for an IPv6 address. If the  
13946 AI\_PASSIVE flag is not specified, the returned address information shall be suitable for a call to  
13947 *connect()* (for a connection-mode protocol) or for a call to *connect()*, *sendto()*, or *sendmsg()* (for a  
13948 connectionless protocol). In this case, if the *nodename* argument is null, then the IP address  
13949 portion of the socket address structure shall be set to the loopback address.

13950 If the AI\_CANONNAME flag is specified and the *nodename* argument is not null, the function  
13951 shall attempt to determine the canonical name corresponding to *nodename* (for example, if  
13952 *nodename* is an alias or shorthand notation for a complete name).

13953 If the AI\_NUMERICHOST flag is specified, then a non-null *nodename* string supplied shall be a  
13954 numeric host address string. Otherwise, an [EAI\_NONAME] error is returned. This flag shall  
13955 prevent any type of name resolution service (for example, the DNS) from being invoked.

13956 If the AI\_NUMERICSERV flag is specified, then a non-null *servname* string supplied shall be a  
13957 numeric port string. Otherwise, an [EAI\_NONAME] error shall be returned. This flag shall  
13958 prevent any type of name resolution service (for example, NIS+) from being invoked.

13959 IP6 If the AI\_V4MAPPED flag is specified along with an *ai\_family* of AF\_INET6, then *getaddrinfo()*  
13960 shall return IPv4-mapped IPv6 addresses on finding no matching IPv6 addresses (*ai\_addrlen*  
13961 shall be 16). The AI\_V4MAPPED flag shall be ignored unless *ai\_family* equals AF\_INET6. If the  
13962 AI\_ALL flag is used with the AI\_V4MAPPED flag, then *getaddrinfo()* shall return all matching  
13963 IPv6 and IPv4 addresses. The AI\_ALL flag without the AI\_V4MAPPED flag is ignored.

13964 The *ai\_socktype* field to which argument *hints* points specifies the socket type for the service, as  
13965 defined in *socket()*. If a specific socket type is not given (for example, a value of zero) and the  
13966 service name could be interpreted as valid with multiple supported socket types, the  
13967 implementation shall attempt to resolve the service name for all supported socket types and, in  
13968 the absence of errors, all possible results shall be returned. A non-zero socket type value shall  
13969 limit the returned information to values with the specified socket type.

13970 If the *ai\_family* field to which *hints* points has the value AF\_UNSPEC, addresses shall be  
13971 returned for use with any address family that can be used with the specified *nodename* and/or  
13972 *servname*. Otherwise, addresses shall be returned for use only with the specified address family.  
13973 If *ai\_family* is not AF\_UNSPEC and *ai\_protocol* is not zero, then addresses are returned for use  
13974 only with the specified address family and protocol; the value of *ai\_protocol* shall be interpreted  
13975 as in a call to the *socket()* function with the corresponding values of *ai\_family* and *ai\_protocol*.

#### 13976 RETURN VALUE

13977 A zero return value for *getaddrinfo()* indicates successful completion; a non-zero return value  
13978 indicates failure. The possible values for the failures are listed in the ERRORS section.

13979 Upon successful return of *getaddrinfo()*, the location to which *res* points shall refer to a linked list  
13980 of **addrinfo** structures, each of which shall specify a socket address and information for use in  
13981 creating a socket with which to use that socket address. The list shall include at least one  
13982 **addrinfo** structure. The *ai\_next* field of each structure contains a pointer to the next structure on  
13983 the list, or a null pointer if it is the last structure on the list. Each structure on the list shall

13984 include values for use with a call to the *socket()* function, and a socket address for use with the  
 13985 *connect()* function or, if the AI\_PASSIVE flag was specified, for use with the *bind()* function. The  
 13986 fields *ai\_family*, *ai\_socktype*, and *ai\_protocol* shall be usable as the arguments to the *socket()*  
 13987 function to create a socket suitable for use with the returned address. The fields *ai\_addr* and  
 13988 *ai\_addrlen* are usable as the arguments to the *connect()* or *bind()* functions with such a socket,  
 13989 according to the AI\_PASSIVE flag.

13990 If *nodename* is not null, and if requested by the AI\_CANONNAME flag, the *ai\_canonname* field of  
 13991 the first returned **addrinfo** structure shall point to a null-terminated string containing the  
 13992 canonical name corresponding to the input *nodename*; if the canonical name is not available, then  
 13993 *ai\_canonname* shall refer to the *nodename* argument or a string with the same contents. The  
 13994 contents of the *ai\_flags* field of the returned structures are undefined.

13995 All fields in socket address structures returned by *getaddrinfo()* that are not filled in through an  
 13996 explicit argument (for example, *sin6\_flowinfo*) shall be set to zero.

13997 **Note:** This makes it easier to compare socket address structures.

#### 13998 **ERRORS**

13999 The *getaddrinfo()* function shall fail and return the corresponding value if:

14000 [EAI\_AGAIN] The name could not be resolved at this time. Future attempts may succeed.

14001 [EAI\_BADFLAGS]

14002 The *flags* parameter had an invalid value.

14003 [EAI\_FAIL] A non-recoverable error occurred when attempting to resolve the name.

14004 [EAI\_FAMILY] The address family was not recognized.

14005 [EAI\_MEMORY] There was a memory allocation failure when trying to allocate storage for the  
 14006 return value.

14007 [EAI\_NONAME] The name does not resolve for the supplied parameters.

14008 Neither *nodename* nor *servname* were supplied. At least one of these shall be  
 14009 supplied.

14010 [EAI\_SERVICE] The service passed was not recognized for the specified socket type.

14011 [EAI\_SOCKTYPE]

14012 The intended socket type was not recognized.

14013 [EAI\_SYSTEM] A system error occurred; the error code can be found in *errno*.

14014 [EAI\_OVERFLOW] An argument buffer overflowed.

#### 14015 **EXAMPLES**

14016 None.

#### 14017 **APPLICATION USAGE**

14018 If the caller handles only TCP and not UDP, for example, then the *ai\_protocol* member of the *hints*  
 14019 structure should be set to IPPROTO\_TCP when *getaddrinfo()* is called.

14020 If the caller handles only IPv4 and not IPv6, then the *ai\_family* member of the *hints* structure  
 14021 should be set to AF\_INET when *getaddrinfo()* is called.

#### 14022 **RATIONALE**

14023 None.

14024 **FUTURE DIRECTIONS**

14025       None.

14026 **SEE ALSO**

14027       *connect()*, *gai\_strerror()*, *gethostbyname()*, *getnameinfo()*, *getservbyname()*, *socket()*, the Base  
14028       Definitions volume of IEEE Std 1003.1-200x, <**netdb.h**>, <**sys/socket.h**>

14029 **CHANGE HISTORY**

14030       First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

14031       The **restrict** keyword is added to the *getaddrinfo()* prototype for alignment with the  
14032       ISO/IEC 9899:1999 standard.

## 14033 NAME

14034 freopen — open a stream

## 14035 SYNOPSIS

14036 #include &lt;stdio.h&gt;

```
14037 FILE *freopen(const char *restrict filename, const char *restrict mode,
14038 FILE *restrict stream);
```

## 14039 DESCRIPTION

14040 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 14041 conflict between the requirements described here and the ISO C standard is unintentional. This  
 14042 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

14043 The *freopen()* function shall first attempt to flush the stream and close any file descriptor  
 14044 associated with *stream*. Failure to flush or close the file descriptor successfully shall be ignored.  
 14045 The error and end-of-file indicators for the stream shall be cleared.

14046 The *freopen()* function shall open the file whose pathname is the string pointed to by *filename* and  
 14047 associate the stream pointed to by *stream* with it. The *mode* argument shall be used just as in  
 14048 *fopen()*.

14049 The original stream shall be closed regardless of whether the subsequent open succeeds.

14050 If *filename* is a null pointer, the *freopen()* function shall attempt to change the mode of the stream  
 14051 to that specified by *mode*, as if the name of the file currently associated with the stream had been  
 14052 used. It is implementation-defined which changes of mode are permitted (if any), and under  
 14053 what circumstances.

14054 XSI After a successful call to the *freopen()* function, the orientation of the stream shall be cleared, the  
 14055 encoding rule shall be cleared, and the associated **mbstate\_t** object shall be set to describe an  
 14056 initial conversion state.

14057 CX The largest value that can be represented correctly in an object of type **off\_t** shall be established  
 14058 as the offset maximum in the open file description.

## 14059 RETURN VALUE

14060 Upon successful completion, *freopen()* shall return the value of *stream*. Otherwise, a null pointer  
 14061 CX shall be returned, and *errno* shall be set to indicate the error.

## 14062 ERRORS

14063 The *freopen()* function shall fail if:

14064 CX [EACCES] Search permission is denied on a component of the path prefix, or the file  
 14065 exists and the permissions specified by *mode* are denied, or the file does not  
 14066 exist and write permission is denied for the parent directory of the file to be  
 14067 created.

14068 CX [EINTR] A signal was caught during *freopen()*.

14069 CX [EISDIR] The named file is a directory and *mode* requires write access.

14070 CX [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*  
 14071 argument.

14072 CX [EMFILE] {OPEN\_MAX} file descriptors are currently open in the calling process.

14073 CX [ENAMETOOLONG]

14074 The length of the *filename* argument exceeds {PATH\_MAX} or a pathname  
 14075 component is longer than {NAME\_MAX}.

14076 CX	[ENFILE]	The maximum allowable number of files is currently open in the system.
14077 CX 14078	[ENOENT]	A component of <i>filename</i> does not name an existing file or <i>filename</i> is an empty string.
14079 CX 14080	[ENOSPC]	The directory or file system that would contain the new file cannot be expanded, the file does not exist, and it was to be created.
14081 CX	[ENOTDIR]	A component of the path prefix is not a directory.
14082 CX 14083	[ENXIO]	The named file is a character special or block special file, and the device associated with this special file does not exist.
14084 CX 14085	[EOVERFLOW]	The named file is a regular file and the size of the file cannot be represented correctly in an object of type <code>off_t</code> .
14086 CX 14087	[EROFS]	The named file resides on a read-only file system and <i>mode</i> requires write access.
14088		The <i>freopen()</i> function may fail if:
14089 CX	[EINVAL]	The value of the <i>mode</i> argument is not valid.
14090 CX 14091	[ELOOP]	More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the <i>path</i> argument.
14092 CX 14093 14094	[ENAMETOOLONG]	Pathname resolution of a symbolic link produced an intermediate result whose length exceeds {PATH_MAX}.
14095 CX	[ENOMEM]	Insufficient storage space is available.
14096 CX 14097	[ENXIO]	A request was made of a nonexistent device, or the request was outside the capabilities of the device.
14098 CX 14099	[ETXTBSY]	The file is a pure procedure (shared text) file that is being executed and <i>mode</i> requires write access.

#### 14100 EXAMPLES

##### 14101 Directing Standard Output to a File

14102 The following example logs all standard output to the `/tmp/logfile` file.

```

14103 #include <stdio.h>
14104 ...
14105 FILE *fp;
14106 ...
14107 fp = freopen ("/tmp/logfile", "a+", stdout);
14108 ...

```

#### 14109 APPLICATION USAGE

14110 The *freopen()* function is typically used to attach the preopened *streams* associated with *stdin*,  
 14111 *stdout*, and *stderr* to other files.

#### 14112 RATIONALE

14113 None.

14114 **FUTURE DIRECTIONS**

14115 None.

14116 **SEE ALSO**14117 *fclose()*, *fopen()*, *fdopen()*, *mbsinit()*, the Base Definitions volume of IEEE Std 1003.1-200x,  
14118 **<stdio.h>**14119 **CHANGE HISTORY**

14120 First released in Issue 1. Derived from Issue 1 of the SVID.

14121 **Issue 5**14122 The DESCRIPTION is updated to indicate that the orientation of the stream is cleared and the  
14123 conversion state of the stream is set to an initial conversion state by a successful call to the  
14124 *freopen()* function.

14125 Large File Summit extensions are added.

14126 **Issue 6**

14127 Extensions beyond the ISO C standard are now marked.

14128 The following new requirements on POSIX implementations derive from alignment with the |  
14129 Single UNIX Specification:14130 • In the DESCRIPTION, text is added to indicate setting of the offset maximum in the open file  
14131 description. This change is to support large files.14132 • In the ERRORS section, the [E\_OVERFLOW] condition is added. This change is to support  
14133 large files.

14134 • The [ELOOP] mandatory error condition is added.

14135 • A second [ENAMETOOLONG] is added as an optional error condition.

14136 • The [EINVAL], [ENOMEM], [ENXIO], and [ETXTBSY] optional error conditions are added.

14137 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

14138 • The *freopen()* prototype is updated.

14139 • The DESCRIPTION is updated.

14140 The wording of the mandatory [ELOOP] error condition is updated, and a second optional  
14141 [ELOOP] error condition is added.

14142 The DESCRIPTION is updated regarding failure to close, changing the “file” to “file descriptor”.

14143 **NAME**

14144 frexp, frexpf, frexpl — extract mantissa and exponent from a double precision number

14145 **SYNOPSIS**

14146 #include <math.h>

14147 double frexp(double *num*, int \**exp*);

14148 float frexpf(float *num*, int \**exp*);

14149 long double frexpl(long double *num*, int \**exp*);

14150 **DESCRIPTION**

14151 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
14152 conflict between the requirements described here and the ISO C standard is unintentional. This  
14153 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

14154 These functions shall break a floating-point number *num* into a normalized fraction and an  
14155 integral power of 2. The integer exponent shall be stored in the **int** object pointed to by *exp*.

14156 **RETURN VALUE**

14157 For finite arguments, these functions shall return the value *x*, such that *x* has a magnitude in the  
14158 interval  $[\frac{1}{2}, 1)$  or 0, and *num* equals *x* times 2 raised to the power \**exp*.

14159 **MX** If *num* is NaN, a NaN shall be returned, and the value of \**exp* is unspecified.

14160 If *num* is  $\pm 0$ ,  $\pm 0$  shall be returned, and the value of \**exp* shall be 0.

14161 If *num* is  $\pm \text{Inf}$ , *num* shall be returned, and the value of \**exp* is unspecified.

14162 **ERRORS**

14163 No errors are defined.

14164 **EXAMPLES**

14165 None.

14166 **APPLICATION USAGE**

14167 None.

14168 **RATIONALE**

14169 None.

14170 **FUTURE DIRECTIONS**

14171 None.

14172 **SEE ALSO**

14173 *isnan()*, *ldexp()*, *modf()*, the Base Definitions volume of IEEE Std 1003.1-200x, <math.h>

14174 **CHANGE HISTORY**

14175 First released in Issue 1. Derived from Issue 1 of the SVID.

14176 **Issue 5**

14177 The DESCRIPTION is updated to indicate how an application should check for an error. This  
14178 text was previously published in the APPLICATION USAGE section.

14179 **Issue 6**

14180 The *frexpf()* and *frexpl()* functions are added for alignment with the ISO/IEC 9899:1999  
14181 standard.

14182 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are |  
14183 revised to align with the ISO/IEC 9899:1999 standard.  
14184 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
14185 marked.

## 14186 NAME

14187 fscanf, scanf, sscanf — convert formatted input

## 14188 SYNOPSIS

14189 #include &lt;stdio.h&gt;

14190 int fscanf(FILE \*restrict *stream*, const char \*restrict *format*, ... );14191 int scanf(const char \*restrict *format*, ... );14192 int sscanf(const char \*restrict *s*, const char \*restrict *format*, ... );

## 14193 DESCRIPTION

14194 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 14195 conflict between the requirements described here and the ISO C standard is unintentional. This  
 14196 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

14197 The *fscanf()* function shall read from the named input *stream*. The *scanf()* function shall read  
 14198 from the standard input stream *stdin*. The *sscanf()* function shall read from the string *s*. Each  
 14199 function reads bytes, interprets them according to a format, and stores the results in its  
 14200 arguments. Each expects, as arguments, a control string *format* described below, and a set of  
 14201 *pointer* arguments indicating where the converted input should be stored. The result is  
 14202 undefined if there are insufficient arguments for the format. If the format is exhausted while  
 14203 arguments remain, the excess arguments shall be evaluated but otherwise ignored.

14204 XSI Conversions can be applied to the *n*th argument after the *format* in the argument list, rather than  
 14205 to the next unused argument. In this case, the conversion specifier character % (see below) is  
 14206 replaced by the sequence "%n\$", where *n* is a decimal integer in the range [1,{NL\_ARGMAX}].  
 14207 This feature provides for the definition of format strings that select arguments in an order  
 14208 appropriate to specific languages. In format strings containing the "%n\$" form of conversion  
 14209 specifications, it is unspecified whether numbered arguments in the argument list can be  
 14210 referenced from the format string more than once.

14211 The *format* can contain either form of a conversion specification—that is, % or "%n\$"—but the  
 14212 two forms cannot be mixed within a single *format* string. The only exception to this is that %% or  
 14213 %\* can be mixed with the "%n\$" form. When numbered argument specifications are used,  
 14214 specifying the *N*th argument requires that all the leading arguments, from the first to the  
 14215 (*N*–1)th, are pointers.

14216 CX The *fscanf()* function in all its forms shall allow detection of a language-dependent radix  
 14217 character in the input string. The radix character is defined in the program's locale (category  
 14218 *LC\_NUMERIC*). In the POSIX locale, or in a locale where the radix character is not defined, the  
 14219 radix character shall default to a period ( '.' ).

14220 The format is a character string, beginning and ending in its initial shift state, if any, composed  
 14221 of zero or more directives. Each directive is composed of one of the following: one or more  
 14222 white-space characters (<space>*s*, <tab>*s*, <newline>*s*, <vertical-tab>*s*, or <form-feed>*s*); an  
 14223 ordinary character (neither '%' nor a white-space character); or a conversion specification. Each  
 14224 XSI conversion specification is introduced by the character '%' or the character sequence "%n\$",  
 14225 after which the following appear in sequence:

- 14226 • An optional assignment-suppressing character '\*'.
- 14227 • An optional non-zero decimal integer that specifies the maximum field width.
- 14228 • An option length modifier that specifies the size of the receiving object.
- 14229 • A *conversion specifier* character that specifies the type of conversion to be applied. The valid  
 14230 conversion specifiers are described below.

14231 The *fscanf()* functions shall execute each directive of the format in turn. If a directive fails, as  
 14232 detailed below, the function shall return. Failures are described as input failures (due to the  
 14233 unavailability of input bytes) or matching failures (due to inappropriate input).

14234 A directive composed of one or more white-space characters shall be executed by reading input  
 14235 until no more valid input can be read, or up to the first byte which is not a white-space character,  
 14236 which remains unread.

14237 A directive that is an ordinary character shall be executed as follows: the next byte shall be read  
 14238 from the input and compared with the byte that comprises the directive; if the comparison  
 14239 shows that they are not equivalent, the directive shall fail, and the differing and subsequent  
 14240 bytes shall remain unread. Similarly, if end-of-file, an encoding error, or a read error prevents a  
 14241 character from being read, the directive shall fail.

14242 A directive that is a conversion specification defines a set of matching input sequences, as  
 14243 described below for each conversion character. A conversion specification shall be executed in  
 14244 the following steps.

14245 Input white-space characters (as specified by *isspace()*) shall be skipped, unless the conversion  
 14246 specification includes a `[`, `c`, `C`, or `n` conversion specifier.

14247 An item shall be read from the input, unless the conversion specification includes an `n`  
 14248 conversion specifier. An input item shall be defined as the longest sequence of input bytes (up to  
 14249 any specified maximum field width, which may be measured in characters or bytes dependent  
 14250 on the conversion specifier) which is an initial subsequence of a matching sequence. The first  
 14251 byte, if any, after the input item shall remain unread. If the length of the input item is 0, the  
 14252 execution of the conversion specification shall fail; this condition is a matching failure, unless  
 14253 end-of-file, an encoding error, or a read error prevented input from the stream, in which case it is  
 14254 an input failure.

14255 Except in the case of a `%` conversion specifier, the input item (or, in the case of a `%n` conversion  
 14256 specification, the count of input bytes) shall be converted to a type appropriate to the conversion  
 14257 character. If the input item is not a matching sequence, the execution of the conversion  
 14258 specification fails; this condition is a matching failure. Unless assignment suppression was  
 14259 indicated by a `'*'`, the result of the conversion shall be placed in the object pointed to by the  
 14260 first argument following the *format* argument that has not already received a conversion result if  
 14261 XSI the conversion specification is introduced by `%`, or in the *n*th argument if introduced by the  
 14262 character sequence `"%n$"`. If this object does not have an appropriate type, or if the result of the  
 14263 conversion cannot be represented in the space provided, the behavior is undefined.

14264 The length modifiers and their meanings are:

14265 `hh` Specifies that a following `d`, `i`, `o`, `u`, `x`, `X`, or `n` conversion specifier applies to an  
 14266 argument with type pointer to **signed char** or **unsigned char**.

14267 `h` Specifies that a following `d`, `i`, `o`, `u`, `x`, `X`, or `n` conversion specifier applies to an  
 14268 argument with type pointer to **short** or **unsigned short**.

14269 `l` (`ell`) Specifies that a following `d`, `i`, `o`, `u`, `x`, `X`, or `n` conversion specifier applies to an  
 14270 argument with type pointer to **long** or **unsigned long**; that a following `a`, `A`, `e`, `E`, `f`, `F`, `g`,  
 14271 or `G` conversion specifier applies to an argument with type pointer to **double**; or that a  
 14272 following `c`, `s`, or `[` conversion specifier applies to an argument with type pointer to  
 14273 **wchar\_t**.

14274 `ll` (`ell-ell`)  
 14275 Specifies that a following `d`, `i`, `o`, `u`, `x`, `X`, or `n` conversion specifier applies to an  
 14276 argument with type pointer to **long long** or **unsigned long long**.

14277	j	Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an argument with type pointer to <b>intmax_t</b> or <b>uintmax_t</b> .
14278		
14279	z	Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an argument with type pointer to <b>size_t</b> or the corresponding signed integer type.
14280		
14281	t	Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an argument with type pointer to <b>ptrdiff_t</b> or the corresponding <b>unsigned</b> type.
14282		
14283	L	Specifies that a following a, A, e, E, f, F, g, or G conversion specifier applies to an argument with type pointer to <b>long double</b> .
14284		
14285		If a length modifier appears with any conversion specifier other than as specified above, the behavior is undefined.
14286		
14287		The following conversion specifiers are valid:
14288	d	Matches an optionally signed decimal integer, whose format is the same as expected for the subject sequence of <i>strtol()</i> with the value 10 for the <i>base</i> argument. In the absence of a size modifier, the application shall ensure that the corresponding argument is a pointer to <b>int</b> .
14289		
14290		
14291		
14292	i	Matches an optionally signed integer, whose format is the same as expected for the subject sequence of <i>strtol()</i> with 0 for the <i>base</i> argument. In the absence of a size modifier, the application shall ensure that the corresponding argument is a pointer to <b>int</b> .
14293		
14294		
14295		
14296	o	Matches an optionally signed octal integer, whose format is the same as expected for the subject sequence of <i>strtol()</i> with the value 8 for the <i>base</i> argument. In the absence of a size modifier, the application shall ensure that the corresponding argument is a pointer to <b>unsigned</b> .
14297		
14298		
14299		
14300	u	Matches an optionally signed decimal integer, whose format is the same as expected for the subject sequence of <i>strtoul()</i> with the value 10 for the <i>base</i> argument. In the absence of a size modifier, the application shall ensure that the corresponding argument is a pointer to <b>unsigned</b> .
14301		
14302		
14303		
14304	x	Matches an optionally signed hexadecimal integer, whose format is the same as expected for the subject sequence of <i>strtoul()</i> with the value 16 for the <i>base</i> argument. In the absence of a size modifier, the application shall ensure that the corresponding argument is a pointer to <b>unsigned</b> .
14305		
14306		
14307		
14308	a, e, f, g	
14309		Matches an optionally signed floating-point number, infinity, or NaN, whose format is the same as expected for the subject sequence of <i>strtod()</i> . In the absence of a size modifier, the application shall ensure that the corresponding argument is a pointer to <b>float</b> .
14310		
14311		
14312		
14313		If the <i>fprintf()</i> family of functions generates character string representations for infinity and NaN (a symbolic entity encoded in floating-point format) to support IEEE Std 754-1985, the <i>fscanf()</i> family of functions shall recognize them as input.
14314		
14315		
14316	s	Matches a sequence of bytes that are not white-space characters. The application shall ensure that the corresponding argument is a pointer to the initial byte of an array of <b>char</b> , <b>signed char</b> , or <b>unsigned char</b> large enough to accept the sequence and a terminating null character code, which shall be added automatically.
14317		
14318		
14319		
14320		If an l (ell) qualifier is present, the input is a sequence of characters that begins in the initial shift state. Each character shall be converted to a wide character as if by a call to
14321		

14322 the *mbrtowc()* function, with the conversion state described by an **mbstate\_t** object |  
14323 initialized to zero before the first character is converted. The application shall ensure that |  
14324 the corresponding argument is a pointer to an array of **wchar\_t** large enough to |  
14325 accept the sequence and the terminating null wide character, which shall be added |  
14326 automatically.

14327 [ Matches a non-empty sequence of bytes from a set of expected bytes (the *scanset*). The |  
14328 normal skip over white-space characters shall be suppressed in this case. The |  
14329 application shall ensure that the corresponding argument is a pointer to the initial byte |  
14330 of an array of **char**, **signed char**, or **unsigned char** large enough to accept the sequence |  
14331 and a terminating null byte, which shall be added automatically.

14332 If an **l** (ell) qualifier is present, the input is a sequence of characters that begins in the |  
14333 initial shift state. Each character in the sequence shall be converted to a wide character |  
14334 as if by a call to the *mbrtowc()* function, with the conversion state described by an |  
14335 **mbstate\_t** object initialized to zero before the first character is converted. The |  
14336 application shall ensure that the corresponding argument is a pointer to an array of |  
14337 **wchar\_t** large enough to accept the sequence and the terminating null wide character, |  
14338 which shall be added automatically.

14339 The conversion specification includes all subsequent bytes in the *format* string up to |  
14340 and including the matching right square bracket (']'). The bytes between the square |  
14341 brackets (the *scanlist*) comprise the scanset, unless the byte after the left square bracket |  
14342 is a circumflex (^), in which case the scanset contains all bytes that do not appear in |  
14343 the scanlist between the circumflex and the right square bracket. If the conversion |  
14344 specification begins with "[ ]" or "[ ^ ]", the right square bracket is included in the |  
14345 scanlist and the next right square bracket is the matching right square bracket that ends |  
14346 the conversion specification; otherwise, the first right square bracket is the one that |  
14347 ends the conversion specification. If a '-' is in the scanlist and is not the first character, |  
14348 nor the second where the first character is a '^', nor the last character, the behavior is |  
14349 implementation-defined.

14350 c Matches a sequence of bytes of the number specified by the field width (1 if no field |  
14351 width is present in the conversion specification). The application shall ensure that the |  
14352 corresponding argument is a pointer to the initial byte of an array of **char**, **signed char**, |  
14353 or **unsigned char** large enough to accept the sequence. No null byte is added. The |  
14354 normal skip over white-space characters shall be suppressed in this case.

14355 If an **l** (ell) qualifier is present, the input shall be a sequence of characters that begins in |  
14356 the initial shift state. Each character in the sequence is converted to a wide character as |  
14357 if by a call to the *mbrtowc()* function, with the conversion state described by an |  
14358 **mbstate\_t** object initialized to zero before the first character is converted. The |  
14359 application shall ensure that the corresponding argument is a pointer to an array of |  
14360 **wchar\_t** large enough to accept the resulting sequence of wide characters. No null wide |  
14361 character is added.

14362 p Matches an implementation-defined set of sequences, which shall be the same as the set |  
14363 of sequences that is produced by the %p conversion specification of the corresponding |  
14364 *fprintf()* functions. The application shall ensure that the corresponding argument is a |  
14365 pointer to a pointer to **void**. The interpretation of the input item is implementation- |  
14366 defined. If the input item is a value converted earlier during the same program |  
14367 execution, the pointer that results shall compare equal to that value; otherwise, the |  
14368 behavior of the %p conversion specification is undefined.

14369 n No input is consumed. The application shall ensure that the corresponding argument is |  
14370 a pointer to the integer into which shall be written the number of bytes read from the |

14371		input so far by this call to the <i>fscanf()</i> functions. Execution of a <code>%n</code> conversion	
14372		specification shall not increment the assignment count returned at the completion of	
14373		execution of the function. No argument shall be converted, but one shall be consumed.	
14374		If the conversion specification includes an assignment-suppressing character or a field	
14375		width, the behavior is undefined.	
14376	XSI	<b>C</b> Equivalent to <code>lc</code> .	
14377	XSI	<b>S</b> Equivalent to <code>ls</code> .	
14378		<code>%</code> Matches a single <code>'%'</code> character; no conversion or assignment occurs. The complete	
14379		conversion specification shall be <code>%%</code> .	
14380		If a conversion specification is invalid, the behavior is undefined.	
14381		The conversion specifiers <code>A</code> , <code>E</code> , <code>F</code> , <code>G</code> , and <code>X</code> are also valid and shall be equivalent to <code>a</code> , <code>e</code> , <code>f</code> , <code>g</code> , and	
14382		<code>x</code> , respectively.	
14383		If end-of-file is encountered during input, conversion shall be terminated. If end-of-file occurs	
14384		before any bytes matching the current conversion specification (except for <code>%n</code> ) have been read	
14385		(other than leading white-space characters, where permitted), execution of the current	
14386		conversion specification shall terminate with an input failure. Otherwise, unless execution of the	
14387		current conversion specification is terminated with a matching failure, execution of the	
14388		following conversion specification (if any) shall be terminated with an input failure.	
14389		Reaching the end of the string in <i>sscanf()</i> shall be equivalent to encountering end-of-file for	
14390		<i>fscanf()</i> .	
14391		If conversion terminates on a conflicting input, the offending input is left unread in the input.	
14392		Any trailing white space (including <code>&lt;newline&gt;</code> s) shall be left unread unless matched by a	
14393		conversion specification. The success of literal matches and suppressed assignments is only	
14394		directly determinable via the <code>%n</code> conversion specification.	
14395	CX	The <i>fscanf()</i> and <i>scanf()</i> functions may mark the <i>st_atime</i> field of the file associated with <i>stream</i>	
14396		for update. The <i>st_atime</i> field shall be marked for update by the first successful execution of	
14397		<i>fgetc()</i> , <i>fgets()</i> , <i>fread()</i> , <i>getc()</i> , <i>getchar()</i> , <i>gets()</i> , <i>fscanf()</i> , or <i>scanf()</i> using <i>stream</i> that returns data	
14398		not supplied by a prior call to <i>ungetc()</i> .	
14399		<b>RETURN VALUE</b>	
14400		Upon successful completion, these functions shall return the number of successfully matched	
14401		and assigned input items; this number can be zero in the event of an early matching failure. If	
14402		the input ends before the first matching failure or conversion, EOF shall be returned. If a read	
14403	CX	error occurs, the error indicator for the stream is set, EOF shall be returned, and <i>errno</i> shall be set	
14404		to indicate the error.	
14405		<b>ERRORS</b>	
14406		For the conditions under which the <i>fscanf()</i> functions fail and may fail, refer to <i>fgetc()</i> or	
14407		<i>fgetwc()</i> .	
14408		In addition, <i>fscanf()</i> may fail if:	
14409	XSI	<b>[EILSEQ]</b> Input byte sequence does not form a valid character.	
14410	XSI	<b>[EINVAL]</b> There are insufficient arguments.	

14411 **EXAMPLES**

14412 The call:

```
14413 int i, n; float x; char name[50];
14414 n = scanf("%d%f%s", &i, &x, name);
```

14415 with the input line:

14416 25 54.32E-1 Hamster

14417 assigns to *n* the value 3, to *i* the value 25, to *x* the value 5.432, and *name* contains the string  
14418 "Hamster".

14419 The call:

```
14420 int i; float x; char name[50];
14421 (void) scanf("%2d%f*d %[0123456789]", &i, &x, name);
```

14422 with input:

14423 56789 0123 56a72

14424 assigns 56 to *i*, 789.0 to *x*, skips 0123, and places the string "56\0" in *name*. The next call to  
14425 *getchar()* shall return the character 'a'.

14426 **Reading Data into an Array**

14427 The following call uses *fscanf()* to read three floating-point numbers from standard input into  
14428 the *input* array.

```
14429 float input[3]; fscanf (stdin, "%f %f %f", input, input+1, input+2);
```

14430 **APPLICATION USAGE**

14431 If the application calling *fscanf()* has any objects of type **wint\_t** or **wchar\_t**, it must also include  
14432 the **<wchar.h>** header to have these objects defined.

14433 **RATIONALE**

14434 This function is aligned with the ISO/IEC 9899:1999 standard, and in doing so a few "obvious"  
14435 things were not included. Specifically, the set of characters allowed in a scanset is limited to  
14436 single-byte characters. In other similar places, multi-byte characters have been permitted, but  
14437 for alignment with the ISO/IEC 9899:1999 standard, it has not been done here. Applications  
14438 needing this could use the corresponding wide-character functions to achieve the desired  
14439 results.

14440 **FUTURE DIRECTIONS**

14441 None.

14442 **SEE ALSO**

14443 *getc()*, *printf()*, *setlocale()*, *strtod()*, *strtol()*, *strtoul()*, *wcrtomb()*, the Base Definitions volume of  
14444 IEEE Std 1003.1-200x, **<langinfo.h>**, **<stdio.h>**, **<wchar.h>**, the Base Definitions volume of  
14445 IEEE Std 1003.1-200x, Chapter 7, Locale

14446 **CHANGE HISTORY**

14447 First released in Issue 1. Derived from Issue 1 of the SVID.

14448 **Issue 5**

14449 Aligned with ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, the **l** (ell) qualifier is  
14450 now defined for the **c**, **s**, and **[** conversion specifiers.

14451 The DESCRIPTION is updated to indicate that if infinity and NaN can be generated by the  
14452 *fprintf()* family of functions, then they are recognized by the *fscanf()* family.

14453 **Issue 6**

14454 The Open Group Corrigendum U021/7 and U028/10 are applied. These correct several  
14455 occurrences of “characters” in the text which have been replaced with the term “bytes”.

14456 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

14457 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

14458 • The prototypes for *fscanf()*, *scanf()*, and *sscanf()* are updated.

14459 • The DESCRIPTION is updated. |

14460 • The hh, ll, j, t, and z length modifiers are added. |

14461 • The a, A, and F conversion characters are added. |

14462 The DESCRIPTION is updated to use the terms “conversion specifier” and “conversion  
14463 specification” consistently.

14464 **NAME**

14465        fseek, fseeko — reposition a file-position indicator in a stream

14466 **SYNOPSIS**

14467        #include &lt;stdio.h&gt;

14468        int fseek(FILE \*stream, long offset, int whence);

14469 CX     int fseeko(FILE \*stream, off\_t offset, int whence);

14470

14471 **DESCRIPTION**

14472 CX     The functionality described on this reference page is aligned with the ISO C standard. Any  
 14473     conflict between the requirements described here and the ISO C standard is unintentional. This  
 14474     volume of IEEE Std 1003.1-200x defers to the ISO C standard.

14475     The *fseek()* function shall set the file-position indicator for the stream pointed to by *stream*. If a  
 14476     read or write error occurs, the error indicator for the stream shall be set and *fseek()* fails.

14477     The new position, measured in bytes from the beginning of the file, shall be obtained by adding  
 14478     *offset* to the position specified by *whence*. The specified point is the beginning of the file for  
 14479     SEEK\_SET, the current value of the file-position indicator for SEEK\_CUR, or end-of-file for  
 14480     SEEK\_END.

14481     If the stream is to be used with wide-character input/output functions, the application shall  
 14482     ensure that *offset* is either 0 or a value returned by an earlier call to *ftell()* on the same stream and  
 14483     *whence* is SEEK\_SET.

14484     A successful call to *fseek()* shall clear the end-of-file indicator for the stream and undo any effects  
 14485     of *ungetc()* and *ungetwc()* on the same stream. After an *fseek()* call, the next operation on an  
 14486     update stream may be either input or output.

14487 CX     If the most recent operation, other than *ftell()*, on a given stream is *flush()*, the file offset in the  
 14488     underlying open file description shall be adjusted to reflect the location specified by *fseek()*.

14489     The *fseek()* function shall allow the file-position indicator to be set beyond the end of existing  
 14490     data in the file. If data is later written at this point, subsequent reads of data in the gap shall  
 14491     return bytes with the value 0 until data is actually written into the gap.

14492     The behavior of *fseek()* on devices which are incapable of seeking is implementation-defined.  
 14493     The value of the file offset associated with such a device is undefined.

14494     If the stream is writable and buffered data had not been written to the underlying file, *fseek()*  
 14495     shall cause the unwritten data to be written to the file and shall mark the *st\_ctime* and *st\_mtime*  
 14496     fields of the file for update.

14497     In a locale with state-dependent encoding, whether *fseek()* restores the stream's shift state is  
 14498     implementation-defined.

14499     The *fseeko()* function shall be equivalent to the *fseek()* function except that the *offset* argument is  
 14500     of type **off\_t**.

14501 **RETURN VALUE**14502 CX     The *fseek()* and *fseeko()* functions shall return 0 if they succeed.14503 CX     Otherwise, they shall return -1 and set *errno* to indicate the error.14504 **ERRORS**

14505 CX     The *fseek()* and *fseeko()* functions shall fail if, either the *stream* is unbuffered or the *stream*'s  
 14506     buffer needed to be flushed, and the call to *fseek()* or *fseeko()* causes an underlying *lseek()* or  
 14507     *write()* to be invoked, and:

14508 CX 14509	[EAGAIN]	The O_NONBLOCK flag is set for the file descriptor and the process would be delayed in the write operation.
14510 CX 14511	[EBADF]	The file descriptor underlying the stream file is not open for writing or the stream's buffer needed to be flushed and the file is not open.
14512 CX	[EFBIG]	An attempt was made to write a file that exceeds the maximum file size.
14513 XSI	[EFBIG]	An attempt was made to write a file that exceeds the process' file size limit.
14514 CX 14515	[EFBIG]	The file is a regular file and an attempt was made to write at or beyond the offset maximum associated with the corresponding stream.
14516 CX 14517	[EINTR]	The write operation was terminated due to the receipt of a signal, and no data was transferred.
14518 CX 14519	[EINVAL]	The <i>whence</i> argument is invalid. The resulting file-position indicator would be set to a negative value.
14520 CX 14521 14522 14523 14524	[EIO]	A physical I/O error has occurred, or the process is a member of a background process group attempting to perform a <i>write()</i> to its controlling terminal, TOSTOP is set, the process is neither ignoring nor blocking SIGTTOU, and the process group of the process is orphaned. This error may also be returned under implementation-defined conditions.
14525 CX	[ENOSPC]	There was no free space remaining on the device containing the file.
14526 CX 14527	[ENXIO]	A request was made of a nonexistent device, or the request was outside the capabilities of the device.
14528 CX 14529	[EOVERFLOW]	For <i>fseek()</i> , the resulting file offset would be a value which cannot be represented correctly in an object of type <b>long</b> .
14530 CX 14531	[EOVERFLOW]	For <i>fseeko()</i> , the resulting file offset would be a value which cannot be represented correctly in an object of type <b>off_t</b> .
14532 CX 14533	[EPIPE]	An attempt was made to write to a pipe or FIFO that is not open for reading by any process; a SIGPIPE signal shall also be sent to the thread.
14534 CX	[ESPIPE]	The file descriptor underlying <i>stream</i> is associated with a pipe or FIFO.

14535 **EXAMPLES**

14536 None.

14537 **APPLICATION USAGE**

14538 None.

14539 **RATIONALE**

14540 None.

14541 **FUTURE DIRECTIONS**

14542 None.

14543 **SEE ALSO**

14544 *fopen()*, *fsetpos()*, *ftell()*, *getrlimit()*, *lseek()*, *rewind()*, *ulimit()*, *ungetc()*, *write()*, the Base  
 14545 Definitions volume of IEEE Std 1003.1-200x, <stdio.h>

14546 **CHANGE HISTORY**

14547 First released in Issue 1. Derived from Issue 1 of the SVID.

14548 **Issue 5**

14549 Normative text previously in the APPLICATION USAGE section is moved to the  
14550 DESCRIPTION.

14551 Large File Summit extensions are added.

14552 **Issue 6**

14553 Extensions beyond the ISO C standard are now marked.

14554 The following new requirements on POSIX implementations derive from alignment with the  
14555 Single UNIX Specification:

14556 • The *fseeko()* function is added.

14557 • The [EFBIG], [EOVERFLOW], and [ENXIO] mandatory error conditions are added.

14558 The following change is incorporated for alignment with the FIPS requirements:

14559 • The [EINTR] error is no longer an indication that the implementation does not report partial  
14560 transfers.

14561 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

14562 The DESCRIPTION is updated to explicitly state that *fseek()* sets the file-position indicator, and  
14563 then on error the error indicator is set and *fseek()* fails. This is for alignment with the  
14564 ISO/IEC 9899:1999 standard.

14565 **NAME**

14566 fsetpos — set current file position

14567 **SYNOPSIS**

14568 #include &lt;stdio.h&gt;

14569 int fsetpos(FILE \*stream, const fpos\_t \*pos);

14570 **DESCRIPTION**

14571 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 14572 conflict between the requirements described here and the ISO C standard is unintentional. This  
 14573 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

14574 The *fsetpos()* function shall set the file position and state indicators for the stream pointed to by  
 14575 *stream* according to the value of the object pointed to by *pos*, which the application shall ensure  
 14576 is a value obtained from an earlier call to *fgetpos()* on the same stream. If a read or write error  
 14577 occurs, the error indicator for the stream shall be set and *fsetpos()* fails.

14578 A successful call to the *fsetpos()* function shall clear the end-of-file indicator for the stream and  
 14579 undo any effects of *ungetc()* on the same stream. After an *fsetpos()* call, the next operation on an  
 14580 update stream may be either input or output.

14581 CX The behavior of *fsetpos()* on devices which are incapable of seeking is implementation-defined.  
 14582 The value of the file offset associated with such a device is undefined.

14583 **RETURN VALUE**

14584 The *fsetpos()* function shall return 0 if it succeeds; otherwise, it shall return a non-zero value and  
 14585 set *errno* to indicate the error.

14586 **ERRORS**

14587 CX The *fsetpos()* function shall fail if, either the *stream* is unbuffered or the *stream*'s buffer needed to  
 14588 be flushed, and the call to *fsetpos()* causes an underlying *lseek()* or *write()* to be invoked, and:

14589 CX [EAGAIN] The O\_NONBLOCK flag is set for the file descriptor and the process would be  
 14590 delayed in the write operation.

14591 CX [EBADF] The file descriptor underlying the stream file is not open for writing or the  
 14592 stream's buffer needed to be flushed and the file is not open.

14593 CX [EFBIG] An attempt was made to write a file that exceeds the maximum file size.

14594 XSI [EFBIG] An attempt was made to write a file that exceeds the process' file size limit.

14595 CX [EFBIG] The file is a regular file and an attempt was made to write at or beyond the  
 14596 offset maximum associated with the corresponding stream.

14597 CX [EINTR] The write operation was terminated due to the receipt of a signal, and no data  
 14598 was transferred.

14599 CX [EINVAL] The *whence* argument is invalid. The resulting file-position indicator would be  
 14600 set to a negative value.

14601 CX [EIO] A physical I/O error has occurred, or the process is a member of a  
 14602 background process group attempting to perform a *write()* to its controlling  
 14603 terminal, TOSTOP is set, the process is neither ignoring nor blocking  
 14604 SIGTTOU, and the process group of the process is orphaned. This error may  
 14605 also be returned under implementation-defined conditions.

14606 CX [ENOSPC] There was no free space remaining on the device containing the file.

14607 CX [ENXIO] A request was made of a nonexistent device, or the request was outside the  
14608 capabilities of the device.

14609 CX [EPIPE] The file descriptor underlying *stream* is associated with a pipe or FIFO.

14610 CX [EPIPE] An attempt was made to write to a pipe or FIFO that is not open for reading  
14611 by any process; a SIGPIPE signal shall also be sent to the thread.

#### 14612 EXAMPLES

14613 None.

#### 14614 APPLICATION USAGE

14615 None.

#### 14616 RATIONALE

14617 None.

#### 14618 FUTURE DIRECTIONS

14619 None.

#### 14620 SEE ALSO

14621 *fopen()*, *ftell()*, *lseek()*, *rewind()*, *ungetc()*, *write()*, the Base Definitions volume of  
14622 IEEE Std 1003.1-200x, <stdio.h>

#### 14623 CHANGE HISTORY

14624 First released in Issue 4. Derived from the ISO C standard.

#### 14625 Issue 6

14626 Extensions beyond the ISO C standard are now marked.

14627 An additional [EPIPE] error condition is added for sockets.

14628 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

14629 The DESCRIPTION is updated to clarify that the error indicator is set for the stream on a read or  
14630 write error. This is for alignment with the ISO/IEC 9899:1999 standard.

## 14631 NAME

14632 fstat — get file status

## 14633 SYNOPSIS

14634 #include &lt;sys/stat.h&gt;

14635 int fstat(int *fildev*, struct stat \**buf*);

## 14636 DESCRIPTION

14637 The *fstat()* function shall obtain information about an open file associated with the file |  
 14638 descriptor *fildev*, and shall write it to the area pointed to by *buf*. |

14639 SHM If *fildev* references a shared memory object, the implementation shall update in the **stat** structure |  
 14640 pointed to by the *buf* argument only the *st\_uid*, *st\_gid*, *st\_size*, and *st\_mode* fields, and only the |  
 14641 S\_IRUSR, S\_IWUSR, S\_IRGRP, S\_IWGRP, S\_IROTH, and S\_IWOTH file permission bits need be |  
 14642 valid. The implementation may update other fields and flags. |

14643 TYM If *fildev* references a typed memory object, the implementation shall update in the **stat** structure |  
 14644 pointed to by the *buf* argument only the *st\_uid*, *st\_gid*, *st\_size*, and *st\_mode* fields, and only the |  
 14645 S\_IRUSR, S\_IWUSR, S\_IRGRP, S\_IWGRP, S\_IROTH, and S\_IWOTH file permission bits need be |  
 14646 valid. The implementation may update other fields and flags. |

14647 The *buf* argument is a pointer to a **stat** structure, as defined in <sys/stat.h>, into which |  
 14648 information is placed concerning the file. |

14649 The structure members *st\_mode*, *st\_ino*, *st\_dev*, *st\_uid*, *st\_gid*, *st\_atime*, *st\_ctime*, and *st\_mtime* |  
 14650 shall have meaningful values for all other file types defined in this volume of |  
 14651 IEEE Std 1003.1-200x. The value of the member *st\_nlink* shall be set to the number of links to the |  
 14652 file. |

14653 An implementation that provides additional or alternative file access control mechanisms may, |  
 14654 under implementation-defined conditions, cause *fstat()* to fail. |

14655 The *fstat()* function shall update any time-related fields as described in the Base Definitions |  
 14656 volume of IEEE Std 1003.1-200x, Section 4.7, File Times Update, before writing into the **stat** |  
 14657 structure. |

## 14658 RETURN VALUE

14659 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to |  
 14660 indicate the error. |

## 14661 ERRORS

14662 The *fstat()* function shall fail if:

14663 [EBADF] The *fildev* argument is not a valid file descriptor.

14664 [EIO] An I/O error occurred while reading from the file system.

14665 [EOVERFLOW] The file size in bytes or the number of blocks allocated to the file or the file |  
 14666 serial number cannot be represented correctly in the structure pointed to by |  
 14667 *buf*. |

14668 The *fstat()* function may fail if:

14669 [EOVERFLOW] One of the values is too large to store into the structure pointed to by the *buf* |  
 14670 argument. |

14671 **EXAMPLES**14672 **Obtaining File Status Information**

14673 The following example shows how to obtain file status information for a file named  
 14674 `/home/cnd/mod1`. The structure variable `buffer` is defined for the `stat` structure. The  
 14675 `/home/cnd/mod1` file is opened with read/write privileges and is passed to the open file  
 14676 descriptor `fildev`.

```
14677 #include <sys/types.h>
14678 #include <sys/stat.h>
14679 #include <fcntl.h>

14680 struct stat buffer;
14681 int      status;
14682 ...
14683 fildev = open("/home/cnd/mod1", O_RDWR);
14684 status = fstat(fildev, &buffer);
```

14685 **APPLICATION USAGE**

14686 None.

14687 **RATIONALE**

14688 None.

14689 **FUTURE DIRECTIONS**

14690 None.

14691 **SEE ALSO**

14692 `lstat()`, `stat()`, the Base Definitions volume of IEEE Std 1003.1-200x, `<sys/stat.h>`, `<sys/types.h>`

14693 **CHANGE HISTORY**

14694 First released in Issue 1. Derived from Issue 1 of the SVID.

14695 **Issue 5**

14696 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension.

14697 Large File Summit extensions are added.

14698 **Issue 6**

14699 In the SYNOPSIS, the optional include of the `<sys/types.h>` header is removed.

14700 The following new requirements on POSIX implementations derive from alignment with the  
 14701 Single UNIX Specification:

- 14702 • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was  
 14703 required for conforming implementations of previous POSIX specifications, it was not  
 14704 required for UNIX applications.
- 14705 • The [EIO] mandatory error condition is added.
- 14706 • The [EOVERFLOW] mandatory error condition is added. This change is to support large  
 14707 files.
- 14708 • The [EOVERFLOW] optional error condition is added.

14709 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that  
 14710 shared memory object semantics apply to typed memory objects.

14711 **NAME**

14712 fstatvfs, statvfs — get file system information

14713 **SYNOPSIS**

14714 XSI #include &lt;sys/statvfs.h&gt;

14715 int fstatvfs(int *fildev*, struct statvfs \**buf*);14716 int statvfs(const char \*restrict *path*, struct statvfs \*restrict *buf*);

14717

14718 **DESCRIPTION**14719 The *fstatvfs()* function shall obtain information about the file system containing the file |  
14720 referenced by *fildev*. |14721 The *statvfs()* function shall obtain information about the file system containing the file named by |  
14722 *path*. |14723 For both functions, the *buf* argument is a pointer to a **statvfs** structure that shall be filled. Read, |  
14724 write, or execute permission of the named file is not required. |14725 The following flags can be returned in the *f\_flag* member: |

14726 ST\_RDONLY Read-only file system. |

14727 ST\_NOSUID Setuid/setgid bits ignored by *exec*. |14728 It is unspecified whether all members of the **statvfs** structure have meaningful values on all file |  
14729 systems. |14730 **RETURN VALUE**14731 Upon successful completion, *statvfs()* shall return 0. Otherwise, it shall return  $-1$  and set *errno* to |  
14732 indicate the error. |14733 **ERRORS**14734 The *fstatvfs()* and *statvfs()* functions shall fail if:

14735 [EIO] An I/O error occurred while reading the file system.

14736 [EINTR] A signal was caught during execution of the function.

14737 [EOVERFLOW] One of the values to be returned cannot be represented correctly in the |  
14738 structure pointed to by *buf*. |14739 The *fstatvfs()* function shall fail if:14740 [EBADF] The *fildev* argument is not an open file descriptor.14741 The *statvfs()* function shall fail if:

14742 [EACCES] Search permission is denied on a component of the path prefix.

14743 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path* |  
14744 argument. |

14745 [ENAMETOOLONG]

14746 The length of a pathname exceeds {PATH\_MAX} or a pathname component is |  
14747 longer than {NAME\_MAX}. |14748 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.14749 [ENOTDIR] A component of the path prefix of *path* is not a directory.14750 The *statvfs()* function may fail if:

14751 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
14752 resolution of the *path* argument.

14753 [ENAMETOOLONG]  
14754 Pathname resolution of a symbolic link produced an intermediate result |  
14755 whose length exceeds {PATH\_MAX}.

#### 14756 EXAMPLES

##### 14757 **Obtaining File System Information Using fstatvfs()**

14758 The following example shows how to obtain file system information for the file system upon  
14759 which the file named **/home/cnd/mod1** resides, using the *fstatvfs()* function. The  
14760 **/home/cnd/mod1** file is opened with read/write privileges and the open file descriptor is passed  
14761 to the *fstatvfs()* function.

```
14762 #include <statvfs.h>
14763 #include <fcntl.h>
14764 struct statvfs buffer;
14765 int status;
14766 ...
14767 fildes = open("/home/cnd/mod1", O_RDWR);
14768 status = fstatvfs(fildes, &buffer);
```

##### 14769 **Obtaining File System Information Using statvfs()**

14770 The following example shows how to obtain file system information for the file system upon  
14771 which the file named **/home/cnd/mod1** resides, using the *statvfs()* function.

```
14772 #include <statvfs.h>
14773 struct statvfs buffer;
14774 int status;
14775 ...
14776 status = statvfs("/home/cnd/mod1", &buffer);
```

#### 14777 APPLICATION USAGE

14778 None.

#### 14779 RATIONALE

14780 None.

#### 14781 FUTURE DIRECTIONS

14782 None.

#### 14783 SEE ALSO

14784 *chmod()*, *chown()*, *creat()*, *dup()*, *exec*, *fcntl()*, *link()*, *mknod()*, *open()*, *pipe()*, *read()*, *time()*,  
14785 *unlink()*, *utime()*, *write()*, the Base Definitions volume of IEEE Std 1003.1-200x, <**sys/statvfs.h**>

#### 14786 CHANGE HISTORY

14787 First released in Issue 4, Version 2.

#### 14788 Issue 5

14789 Moved from X/OPEN UNIX extension to BASE.

14790 Large File Summit extensions are added.

14791 **Issue 6**

14792 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

14793 The **restrict** keyword is added to the *statvfs()* prototype for alignment with the  
14794 ISO/IEC 9899:1999 standard.

14795 The wording of the mandatory [ELOOP] error condition is updated, and a second optional  
14796 [ELOOP] error condition is added.

14797 **NAME**

14798 fsync — synchronize changes to a file

14799 **SYNOPSIS**

14800 FSC #include &lt;unistd.h&gt;

14801 int fsync(int *fildev*);

14802

14803 **DESCRIPTION**

14804 The *fsync()* function shall request that all data for the open file descriptor named by *fildev* is to be  
 14805 transferred to the storage device associated with the file described by *fildev* in an  
 14806 implementation-defined manner. The *fsync()* function shall not return until the system has  
 14807 completed that action or until an error is detected.

14808 SIO If `_POSIX_SYNCHRONIZED_IO` is defined, the *fsync()* function shall force all currently queued  
 14809 I/O operations associated with the file indicated by file descriptor *fildev* to the synchronized I/O  
 14810 completion state. All I/O operations shall be completed as defined for synchronized I/O file  
 14811 integrity completion.

14812 **RETURN VALUE**

14813 Upon successful completion, *fsync()* shall return 0. Otherwise, `-1` shall be returned and *errno* set  
 14814 to indicate the error. If the *fsync()* function fails, outstanding I/O operations are not guaranteed  
 14815 to have been completed.

14816 **ERRORS**14817 The *fsync()* function shall fail if:14818 [EBADF] The *fildev* argument is not a valid descriptor.14819 [EINTR] The *fsync()* function was interrupted by a signal.14820 [EINVAL] The *fildev* argument does not refer to a file on which this operation is possible.

14821 [EIO] An I/O error occurred while reading from or writing to the file system.

14822 In the event that any of the queued I/O operations fail, *fsync()* shall return the error conditions  
 14823 defined for *read()* and *write()*.

14824 **EXAMPLES**

14825 None.

14826 **APPLICATION USAGE**

14827 The *fsync()* function should be used by programs which require modifications to a file to be  
 14828 completed before continuing; for example, a program which contains a simple transaction  
 14829 facility might use it to ensure that all modifications to a file or files caused by a transaction are  
 14830 recorded.

14831 **RATIONALE**

14832 The *fsync()* function is intended to force a physical write of data from the buffer cache, and to  
 14833 assure that after a system crash or other failure that all data up to the time of the *fsync()* call is  
 14834 recorded on the disk. Since the concepts of “buffer cache”, “system crash”, “physical write”, and  
 14835 “non-volatile storage” are not defined here, the wording has to be more abstract.

14836 If `_POSIX_SYNCHRONIZED_IO` is not defined, the wording relies heavily on the conformance  
 14837 document to tell the user what can be expected from the system. It is explicitly intended that a  
 14838 null implementation is permitted. This could be valid in the case where the system cannot assure  
 14839 non-volatile storage under any circumstances or when the system is highly fault-tolerant and the  
 14840 functionality is not required. In the middle ground between these extremes, *fsync()* might or  
 14841 might not actually cause data to be written where it is safe from a power failure. The

14842 conformance document should identify at least that one configuration exists (and how to obtain  
14843 that configuration) where this can be assured for at least some files that the user can select to use  
14844 for critical data. It is not intended that an exhaustive list is required, but rather sufficient  
14845 information is provided to let the user determine that if he or she has critical data he or she can  
14846 configure her system to allow it to be written to non-volatile storage.

14847 It is reasonable to assert that the key aspects of *fsync()* are unreasonable to test in a test suite.  
14848 That does not make the function any less valuable, just more difficult to test. A formal  
14849 conformance test should probably force a system crash (power shutdown) during the test for  
14850 this condition, but it needs to be done in such a way that automated testing does not require this  
14851 to be done except when a formal record of the results is being made. It would also not be  
14852 unreasonable to omit testing for *fsync()*, allowing it to be treated as a quality-of-implementation  
14853 issue.

#### 14854 **FUTURE DIRECTIONS**

14855 None.

#### 14856 **SEE ALSO**

14857 *sync()*, the Base Definitions volume of IEEE Std 1003.1-200x, <**unistd.h**>

#### 14858 **CHANGE HISTORY**

14859 First released in Issue 3.

#### 14860 **Issue 5**

14861 Aligned with *fsync()* in the POSIX Realtime Extension. Specifically, the DESCRIPTION and  
14862 RETURN VALUE sections are much expanded, and the ERRORS section is updated to indicate  
14863 that *fsync()* can return the error conditions defined for *read()* and *write()*.

#### 14864 **Issue 6**

14865 This function is marked as part of the File Synchronization option.

14866 The following new requirements on POSIX implementations derive from alignment with the  
14867 Single UNIX Specification:

- 14868 • The [EINVAL] and [EIO] mandatory error conditions are added.

14869 **NAME**

14870 ftell, ftello — return a file offset in a stream

14871 **SYNOPSIS**

14872 #include &lt;stdio.h&gt;

14873 long ftell(FILE \*stream);

14874 CX off\_t ftello(FILE \*stream);

14875

14876 **DESCRIPTION**

14877 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 14878 conflict between the requirements described here and the ISO C standard is unintentional. This  
 14879 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

14880 The *ftell()* function shall obtain the current value of the file-position indicator for the stream  
 14881 pointed to by *stream*.

14882 CX The *ftello()* function shall be equivalent to *ftell()*, except that the return value is of type **off\_t**.

14883 **RETURN VALUE**

14884 CX Upon successful completion, *ftell()* and *ftello()* shall return the current value of the file-position  
 14885 indicator for the stream measured in bytes from the beginning of the file.

14886 CX Otherwise, *ftell()* and *ftello()* shall return  $-1$ , cast to **long** and **off\_t** respectively, and set *errno* to  
 14887 indicate the error.

14888 **ERRORS**

14889 CX The *ftell()* and *ftello()* functions shall fail if:

14890 CX [EBADF] The file descriptor underlying *stream* is not an open file descriptor.

14891 CX [EOVERFLOW] For *ftell()*, the current file offset cannot be represented correctly in an object of  
 14892 type **long**.

14893 CX [EOVERFLOW] For *ftello()*, the current file offset cannot be represented correctly in an object  
 14894 of type **off\_t**.

14895 CX [ESPIPE] The file descriptor underlying *stream* is associated with a pipe or FIFO.

14896 The *ftell()* function may fail if:

14897 CX [ESPIPE] The file descriptor underlying *stream* is associated with a socket.

14898 **EXAMPLES**

14899 None.

14900 **APPLICATION USAGE**

14901 None.

14902 **RATIONALE**

14903 None.

14904 **FUTURE DIRECTIONS**

14905 None.

14906 **SEE ALSO**

14907 *fgetpos()*, *fopen()*, *fseek()*, *lseek()*, the Base Definitions volume of IEEE Std 1003.1-200x, <stdio.h>

14908 **CHANGE HISTORY**

14909 First released in Issue 1. Derived from Issue 1 of the SVID.

14910 **Issue 5**

14911 Large File Summit extensions are added.

14912 **Issue 6**

14913 Extensions beyond the ISO C standard are now marked.

14914 The following new requirements on POSIX implementations derive from alignment with the  
14915 Single UNIX Specification:

- 14916 • The *ftello()* function is added.
- 14917 • The [Eoverflow] error conditions are added.

14918 An additional [ESPIPE] error condition is added for sockets.

14919 **NAME**14920 ftime — get date and time (**LEGACY**)14921 **SYNOPSIS**

14922 xSI #include &lt;sys/timeb.h&gt;

14923 int ftime(struct timeb \*tp);

14924

14925 **DESCRIPTION**

14926 The *ftime()* function shall set the *time* and *millitm* members of the **timeb** structure pointed to by  
 14927 *tp* to contain the seconds and milliseconds portions, respectively, of the current time in seconds  
 14928 since the Epoch. The contents of the *timezone* and *dstflag* members of *tp* after a call to *ftime()* are  
 14929 unspecified.

14930 The system clock need not have millisecond granularity. Depending on any granularity  
 14931 (particularly a granularity of one) renders code non-portable.

14932 **RETURN VALUE**14933 Upon successful completion, the *ftime()* function shall return 0; otherwise, -1 shall be returned.14934 **ERRORS**

14935 No errors are defined.

14936 **EXAMPLES**14937 **Getting the Current Time and Date**

14938 The following example shows how to get the current system time values using the *ftime()*  
 14939 function. The **timeb** structure pointed to by *tp* is filled with the current system time values for  
 14940 *time* and *millitm*.

14941 #include &lt;sys/timeb.h&gt;

14942 struct timeb tp;

14943 int status;

14944 ...

14945 status = ftime(&amp;tp);

14946 **APPLICATION USAGE**

14947 For applications portability, the *time()* function should be used to determine the current time  
 14948 instead of *ftime()*. Realtime applications should use *clock\_gettime()* to determine the current  
 14949 time instead of *ftime()*.

14950 **RATIONALE**

14951 None.

14952 **FUTURE DIRECTIONS**

14953 This function may be withdrawn in a future version.

14954 **SEE ALSO**

14955 *clock\_getres()*, *ctime()*, *gettimeofday()*, *time()*, the Base Definitions volume of  
 14956 IEEE Std 1003.1-200x, <sys/timeb.h>

14957 **CHANGE HISTORY**

14958 First released in Issue 4, Version 2.

14959 **Issue 5**

14960 Moved from X/OPEN UNIX extension to BASE.

14961 Normative text previously in the APPLICATION USAGE section is moved to the  
14962 DESCRIPTION.

14963 **Issue 6**

14964 This function is marked LEGACY.

14965 The DESCRIPTION is updated to refer to “seconds since the Epoch” rather than “seconds since  
14966 00:00:00 UTC (Coordinated Universal Time), January 1 1970” for consistency with other *time*  
14967 functions.

14968 **NAME**

14969 ftok — generate an IPC key

14970 **SYNOPSIS**

14971 xSI #include &lt;sys/ipc.h&gt;

14972 key\_t ftok(const char \*path, int id);

14973

14974 **DESCRIPTION**

14975 The *ftok()* function shall return a key based on *path* and *id* that is usable in subsequent calls to  
 14976 *msgget()*, *semget()*, and *shmget()*. The application shall ensure that the *path* argument is the  
 14977 pathname of an existing file that the process is able to *stat()*.

14978 The *ftok()* function shall return the same key value for all paths that name the same file, when  
 14979 called with the same *id* value, and return different key values when called with different *id*  
 14980 values or with paths that name different files existing on the same file system at the same time. It  
 14981 is unspecified whether *ftok()* shall return the same key value when called again after the file  
 14982 named by *path* is removed and recreated with the same name.

14983 Only the low order 8-bits of *id* are significant. The behavior of *ftok()* is unspecified if these bits  
 14984 are 0.

14985 **RETURN VALUE**

14986 Upon successful completion, *ftok()* shall return a key. Otherwise, *ftok()* shall return (**key\_t**)−1  
 14987 and set *errno* to indicate the error.

14988 **ERRORS**14989 The *ftok()* function shall fail if:

14990 [EACCES] Search permission is denied for a component of the path prefix.

14991 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*  
 14992 argument.

14993 [ENAMETOOLONG]

14994 The length of the *path* argument exceeds {PATH\_MAX} or a pathname  
 14995 component is longer than {NAME\_MAX}.

14996 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

14997 [ENOTDIR] A component of the path prefix is not a directory.

14998 The *ftok()* function may fail if:

14999 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
 15000 resolution of the *path* argument.

15001 [ENAMETOOLONG]

15002 Pathname resolution of a symbolic link produced an intermediate result  
 15003 whose length exceeds {PATH\_MAX}.

15004 **EXAMPLES**15005 **Getting an IPC Key**

15006 The following example gets a unique key that can be used by the IPC functions *semget()*,  
 15007 *msgget()*, and *shmget()*. The key returned by *ftok()* for this example is based on the ID value *S* |  
 15008 and the pathname */tmp*. |

```
15009 #include <sys/ipc.h>
15010 ...
15011 key_t key;
15012 char *path = "/tmp";
15013 int id = 'S';
15014 key = ftok(path, id);
```

15015 **Saving an IPC Key**

15016 The following example gets a unique key based on the pathname */tmp* and the ID value *a*. It |  
 15017 also assigns the value of the resulting key to the *semkey* variable so that it will be available to a  
 15018 later call to *semget()*, *msgget()*, or *shmget()*.

```
15019 #include <sys/ipc.h>
15020 ...
15021 key_t semkey;
15022 if ((semkey = ftok("/tmp", 'a')) == (key_t) -1) {
15023     perror("IPC error: ftok"); exit(1);
15024 }
```

15025 **APPLICATION USAGE**

15026 For maximum portability, *id* should be a single-byte character.

15027 **RATIONALE**

15028 None.

15029 **FUTURE DIRECTIONS**

15030 None.

15031 **SEE ALSO**

15032 *msgget()*, *semget()*, *shmget()*, the Base Definitions volume of IEEE Std 1003.1-200x, *<sys/ipc.h>*

15033 **CHANGE HISTORY**

15034 First released in Issue 4, Version 2.

15035 **Issue 5**

15036 Moved from X/OPEN UNIX extension to BASE.

15037 **Issue 6**

15038 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

15039 The wording of the mandatory [ELOOP] error condition is updated, and a second optional  
 15040 [ELOOP] error condition is added.

15041 **NAME**

15042 ftruncate — truncate a file to a specified length

15043 **SYNOPSIS**

15044 #include &lt;unistd.h&gt;

15045 int ftruncate(int *fd*, off\_t *length*);15046 **DESCRIPTION**15047 If *fd* is not a valid file descriptor open for writing, the *ftruncate()* function shall fail.

15048 If *fd* refers to a regular file, the *ftruncate()* function shall cause the size of the file to be truncated to *length*. If the size of the file previously exceeded *length*, the extra data shall no longer be available to reads on the file. If the file previously was smaller than this size, *ftruncate()* shall either increase the size of the file or fail. XSI-conformant systems shall increase the size of the file. If the file size is increased, the extended area shall appear as if it were zero-filled. The value of the seek pointer shall not be modified by a call to *ftruncate()*.

15054 Upon successful completion, if *fd* refers to a regular file, the *ftruncate()* function shall mark for update the *st\_ctime* and *st\_mtime* fields of the file and the S\_ISUID and S\_ISGID bits of the file mode may be cleared. If the *ftruncate()* function is unsuccessful, the file is unaffected.

15057 XSI If the request would cause the file size to exceed the soft file size limit for the process, the request shall fail and the implementation shall generate the SIGXFSZ signal for the thread.

15059 If *fd* refers to a directory, *ftruncate()* shall fail.15060 If *fd* refers to any other file type, except a shared memory object, the result is unspecified.

15061 SHM If *fd* refers to a shared memory object, *ftruncate()* shall set the size of the shared memory object to *length*.

15063 MF|SHM If the effect of *ftruncate()* is to decrease the size of a shared memory object or memory mapped file and whole pages beyond the new end were previously mapped, then the whole pages beyond the new end shall be discarded.

15066 MPR If the Memory Protection option is supported, references to discarded pages shall result in the generation of a SIGBUS signal; otherwise, the result of such references is undefined.

15068 MF|SHM If the effect of *ftruncate()* is to increase the size of a shared memory object, it is unspecified if the contents of any mapped pages between the old end-of-file and the new are flushed to the underlying object.

15071 **RETURN VALUE**

15072 Upon successful completion, *ftruncate()* shall return 0; otherwise, -1 shall be returned and *errno* set to indicate the error.

15074 **ERRORS**15075 The *ftruncate()* function shall fail if:

15076 [EINTR] A signal was caught during execution.

15077 [EINVAL] The *length* argument was less than 0.

15078 [EFBIG] or [EINVAL]

15079 The *length* argument was greater than the maximum file size.

15080 XSI [EFBIG] The file is a regular file and *length* is greater than the offset maximum established in the open file description associated with *fd*.

15082 [EIO] An I/O error occurred while reading from or writing to a file system.

- 15083 [EBADF] or [EINVAL]  
15084 The *fildest* argument is not a file descriptor open for writing.
- 15085 [EINVAL]  
15086 The *fildest* argument references a file that was opened without write permission.
- 15087 [EROFS] The named file resides on a read-only file system.

**15088 EXAMPLES**

15089 None.

**15090 APPLICATION USAGE**

15091 None.

**15092 RATIONALE**

15093 The *ftruncate()* function is part of IEEE Std 1003.1-200x as it was deemed to be more useful than  
15094 *truncate()*. The *truncate()* function is provided as an XSI extension.

**15095 FUTURE DIRECTIONS**

15096 None.

**15097 SEE ALSO**

15098 *open()*, *truncate()*, the Base Definitions volume of IEEE Std 1003.1-200x, <**unistd.h**>

**15099 CHANGE HISTORY**

15100 First released in Issue 4, Version 2.

**15101 Issue 5**

15102 Moved from X/OPEN UNIX extension to BASE and aligned with *ftruncate()* in the POSIX  
15103 Realtime Extension. Specifically, the DESCRIPTION is extensively reworded and [EROFS] is  
15104 added to the list of mandatory errors that can be returned by *ftruncate()*.

15105 Large File Summit extensions are added.

**15106 Issue 6**

15107 The *truncate()* function has been split out into a separate reference page.

15108 The following new requirements on POSIX implementations derive from alignment with the  
15109 Single UNIX Specification:

- 15110 • The DESCRIPTION is change to indicate that if the file size is changed, and if the file is a  
15111 regular file, the S\_ISUID and S\_ISGID bits in the file mode may be cleared.

15112 The following changes were made to align with the IEEE P1003.1a draft standard:

- 15113 • The DESCRIPTION text is updated.

15114 XSI-conformant systems are required to increase the size of the file if the file was previously  
15115 smaller than the size requested.

15116 **NAME**

15117       ftrylockfile — stdio locking functions

15118 **SYNOPSIS**

15119 TSF     #include &lt;stdio.h&gt;

15120       int ftrylockfile(FILE \*file);

15121

15122 **DESCRIPTION**15123       Refer to *flockfile()*.

15124 **NAME**15125 `ftw` — traverse (walk) a file tree15126 **SYNOPSIS**15127 XSI 

```
#include <ftw.h>
```

```
15128 int ftw(const char *path, int (*fn)(const char *,
15129     const struct stat *ptr, int flag), int ndirs);
15130
```

15131 **DESCRIPTION**

15132 The `ftw()` function shall recursively descend the directory hierarchy rooted in *path*. For each  
 15133 object in the hierarchy, `ftw()` shall call the function pointed to by *fn*, passing it a pointer to a  
 15134 null-terminated character string containing the name of the object, a pointer to a **stat** structure  
 15135 containing information about the object, and an integer. Possible values of the integer, defined  
 15136 in the `<ftw.h>` header, are:

15137 `FTW_D` For a directory.15138 `FTW_DNR` For a directory that cannot be read.15139 `FTW_F` For a file.15140 `FTW_SL` For a symbolic link (but see also `FTW_NS` below).

15141 `FTW_NS` For an object other than a symbolic link on which `stat()` could not successfully be  
 15142 executed. If the object is a symbolic link and `stat()` failed, it is unspecified whether  
 15143 `ftw()` passes `FTW_SL` or `FTW_NS` to the user-supplied function.

15144 If the integer is `FTW_DNR`, descendants of that directory shall not be processed. If the integer is  
 15145 `FTW_NS`, the **stat** structure contains undefined values. An example of an object that would  
 15146 cause `FTW_NS` to be passed to the function pointed to by *fn* would be a file in a directory with  
 15147 read but without execute (search) permission.

15148 The `ftw()` function shall visit a directory before visiting any of its descendants.15149 The `ftw()` function shall use at most one file descriptor for each level in the tree.15150 The argument *ndirs* should be in the range of 1 to `{OPEN_MAX}`.

15151 The tree traversal shall continue until either the tree is exhausted, an invocation of *fn* returns a  
 15152 non-zero value, or some error, other than `[EACCES]`, is detected within `ftw()`.

15153 The *ndirs* argument shall specify the maximum number of directory streams or file descriptors  
 15154 or both available for use by `ftw()` while traversing the tree. When `ftw()` returns it shall close any  
 15155 directory streams and file descriptors it uses not counting any opened by the application-  
 15156 supplied *fn* function.

15157 The results are unspecified if the application-supplied *fn* function does not preserve the current  
 15158 working directory.

15159 The `ftw()` function need not be reentrant. A function that is not required to be reentrant is not  
 15160 required to be thread-safe.

15161 **RETURN VALUE**

15162 If the tree is exhausted, `ftw()` shall return 0. If the function pointed to by *fn* returns a non-zero  
 15163 value, `ftw()` shall stop its tree traversal and return whatever value was returned by the function  
 15164 pointed to by *fn*. If `ftw()` detects an error, it shall return `-1` and set *errno* to indicate the error.

15165 If `ftw()` encounters an error other than `[EACCES]` (see `FTW_DNR` and `FTW_NS` above), it shall  
 15166 return `-1` and set *errno* to indicate the error. The external variable *errno* may contain any error

15167 value that is possible when a directory is opened or when one of the *stat* functions is executed on  
15168 a directory or file.

### 15169 ERRORS

15170 The *ftw()* function shall fail if:

15171 [EACCES] Search permission is denied for any component of *path* or read permission is  
15172 denied for *path*.

15173 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*  
15174 argument.

15175 [ENAMETOOLONG]  
15176 The length of the *path* argument exceeds {PATH\_MAX} or a pathname |  
15177 component is longer than {NAME\_MAX}. |

15178 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

15179 [ENOTDIR] A component of *path* is not a directory. |

15180 [EOVERFLOW] A field in the *stat* structure cannot be represented correctly in the current |  
15181 programming environment for one or more files found in the file hierarchy. |

15182 The *ftw()* function may fail if:

15183 [EINVAL] The value of the *ndirs* argument is invalid.

15184 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
15185 resolution of the *path* argument.

15186 [ENAMETOOLONG]  
15187 Pathname resolution of a symbolic link produced an intermediate result |  
15188 whose length exceeds {PATH\_MAX}. |

15189 In addition, if the function pointed to by *fn* encounters system errors, *errno* may be set  
15190 accordingly.

### 15191 EXAMPLES

#### 15192 Walking a Directory Structure

15193 The following example walks the current directory structure, calling the *fn* function for every  
15194 directory entry, using at most 10 file descriptors:

```
15195 #include <ftw.h>
15196 ...
15197 if (ftw(".", fn, 10) != 0) {
15198     perror("ftw"); exit(2);
15199 }
```

### 15200 APPLICATION USAGE

15201 The *ftw()* function may allocate dynamic storage during its operation. If *ftw()* is forcibly  
15202 terminated, such as by *longjmp()* or *siglongjmp()* being executed by the function pointed to by *fn*  
15203 or an interrupt routine, *ftw()* does not have a chance to free that storage, so it remains  
15204 permanently allocated. A safe way to handle interrupts is to store the fact that an interrupt has  
15205 occurred, and arrange to have the function pointed to by *fn* return a non-zero value at its next  
15206 invocation.

15207 **RATIONALE**

15208           None.

15209 **FUTURE DIRECTIONS**

15210           None.

15211 **SEE ALSO**

15212           *longjmp()*, *lstat()*, *malloc()*, *nftw()*, *opendir()*, *siglongjmp()*, *stat()*, the Base Definitions volume of  
15213           IEEE Std 1003.1-200x, <ftw.h>, <sys/stat.h>

15214 **CHANGE HISTORY**

15215           First released in Issue 1. Derived from Issue 1 of the SVID.

15216 **Issue 5**

15217           UX codings in the DESCRIPTION, RETURN VALUE, and ERRORS sections have been changed  
15218           to EX.

15219 **Issue 6**

15220           The ERRORS section is updated as follows: |

15221           • The wording of the mandatory [ELOOP] error condition is updated. |

15222           • A second optional [ELOOP] error condition is added. |

15223           • The [EOVERFLOW] mandatory error condition is added. |

15224           Text is added to the DESCRIPTION to say that the *ftw()* function need not be reentrant and that |  
15225           the results are unspecified if the application-supplied *fn* function does not preserve the current |  
15226           working directory. |

15227 **NAME**

15228 funlockfile — stdio locking functions

15229 **SYNOPSIS**

15230 TSF #include &lt;stdio.h&gt;

15231 void funlockfile(FILE \*file);

15232

15233 **DESCRIPTION**15234 Refer to *flockfile()*.

15235 **NAME**

15236 fwide — set stream orientation

15237 **SYNOPSIS**

15238 #include &lt;stdio.h&gt;

15239 #include &lt;wchar.h&gt;

15240 int fwide(FILE \**stream*, int *mode*);15241 **DESCRIPTION**

15242 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
15243 conflict between the requirements described here and the ISO C standard is unintentional. This  
15244 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

15245 The *fwide()* function shall determine the orientation of the stream pointed to by *stream*. If *mode* is  
15246 greater than zero, the function first attempts to make the stream wide-oriented. If *mode* is less  
15247 than zero, the function first attempts to make the stream byte-oriented. Otherwise, *mode* is zero  
15248 and the function does not alter the orientation of the stream.

15249 If the orientation of the stream has already been determined, *fwide()* shall not change it.

15250 CX Since no return value is reserved to indicate an error, an application wishing to check for error  
15251 situations should set *errno* to 0, then call *fwide()*, then check *errno*, and if it is non-zero, assume  
15252 an error has occurred.

15253 **RETURN VALUE**

15254 The *fwide()* function shall return a value greater than zero if, after the call, the stream has wide-  
15255 orientation, a value less than zero if the stream has byte-orientation, or zero if the stream has no  
15256 orientation.

15257 **ERRORS**

15258 The *fwide()* function may fail if:

15259 CX [EBADF] The *stream* argument is not a valid stream.

15260 **EXAMPLES**

15261 None.

15262 **APPLICATION USAGE**

15263 A call to *fwide()* with *mode* set to zero can be used to determine the current orientation of a  
15264 stream.

15265 **RATIONALE**

15266 None.

15267 **FUTURE DIRECTIONS**

15268 None.

15269 **SEE ALSO**

15270 The Base Definitions volume of IEEE Std 1003.1-200x, <**wchar.h**>

15271 **CHANGE HISTORY**

15272 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995  
15273 (E).

15274 **Issue 6**

15275 Extensions beyond the ISO C standard are now marked.

15276 **NAME**

15277 fwprintf, swprintf, wprintf — print formatted wide-character output

15278 **SYNOPSIS**

15279 #include &lt;stdio.h&gt;

15280 #include &lt;wchar.h&gt;

15281 int fwprintf(FILE \*restrict *stream*, const wchar\_t \*restrict *format*, ...);15282 int swprintf(wchar\_t \*restrict *ws*, size\_t *n*,15283 const wchar\_t \*restrict *format*, ...);15284 int wprintf(const wchar\_t \*restrict *format*, ...);15285 **DESCRIPTION**

15286 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 15287 conflict between the requirements described here and the ISO C standard is unintentional. This  
 15288 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

15289 The *fwprintf()* function shall place output on the named output *stream*. The *wprintf()* function |  
 15290 shall place output on the standard output stream *stdout*. The *swprintf()* function shall place |  
 15291 output followed by the null wide character in consecutive wide characters starting at *\*ws*; no |  
 15292 more than *n* wide characters shall be written, including a terminating null wide character, which |  
 15293 is always added (unless *n* is zero).

15294 Each of these functions shall convert, format, and print its arguments under control of the *format* |  
 15295 wide-character string. The *format* is composed of zero or more directives: *ordinary wide-* |  
 15296 *characters*, which are simply copied to the output stream, and *conversion specifications*, each of |  
 15297 which results in the fetching of zero or more arguments. The results are undefined if there are |  
 15298 insufficient arguments for the *format*. If the *format* is exhausted while arguments remain, the |  
 15299 excess arguments are evaluated but are otherwise ignored.

15300 XSI Conversions can be applied to the *n*th argument after the *format* in the argument list, rather than |  
 15301 to the next unused argument. In this case, the conversion specifier wide character % (see below) |  
 15302 is replaced by the sequence "%n\$", where *n* is a decimal integer in the range [1,{NL\_ARGMAX}], |  
 15303 giving the position of the argument in the argument list. This feature provides for the definition |  
 15304 of *format* wide-character strings that select arguments in an order appropriate to specific |  
 15305 languages (see the EXAMPLES section).

15306 The *format* can contain either numbered argument specifications (that is, "%n\$" and "\*m\$"), or |  
 15307 unnumbered argument conversion specifications (that is, % and \*), but not both. The only |  
 15308 exception to this is that %% can be mixed with the "%n\$" form. The results of mixing numbered |  
 15309 and unnumbered argument specifications in a *format* wide-character string are undefined. When |  
 15310 numbered argument specifications are used, specifying the *N*th argument requires that all the |  
 15311 leading arguments, from the first to the (*N*-1)th, are specified in the format wide-character |  
 15312 string.

15313 In *format* wide-character strings containing the "%n\$" form of conversion specification, |  
 15314 numbered arguments in the argument list can be referenced from the *format* wide-character |  
 15315 string as many times as required.

15316 In *format* wide-character strings containing the % form of conversion specification, each |  
 15317 argument in the argument list shall be used exactly once.

15318 CX All forms of the *fwprintf()* function allow for the insertion of a locale-dependent radix character |  
 15319 in the output string, output as a wide-character value. The radix character is defined in the |  
 15320 program's locale (category *LC\_NUMERIC*). In the POSIX locale, or in a locale where the radix |  
 15321 character is not defined, the radix character shall default to a period ('.').

15322 XSI Each conversion specification is introduced by the '%' wide character or by the wide-character  
15323 sequence "%n\$", after which the following appear in sequence:

- 15324 • Zero or more *flags* (in any order), which modify the meaning of the conversion specification.
- 15325 • An optional minimum *field width*. If the converted value has fewer wide characters than the  
15326 field width, it shall be padded with spaces by default on the left; it shall be padded on the  
15327 right, if the left-adjustment flag ('-'), described below, is given to the field width. The field  
15328 width takes the form of an asterisk ('\*'), described below, or a decimal integer.
- 15329 • An optional *precision* that gives the minimum number of digits to appear for the d, i, o, u, x,  
15330 and X conversion specifiers; the number of digits to appear after the radix character for the a,  
15331 A, e, E, f, and F conversion specifiers; the maximum number of significant digits for the g  
15332 and G conversion specifiers; or the maximum number of wide characters to be printed from a  
15333 string in the s conversion specifiers. The precision takes the form of a period ('.') followed  
15334 either by an asterisk ('\*'), described below, or an optional decimal digit string, where a null  
15335 digit string is treated as 0. If a precision appears with any other conversion wide character,  
15336 the behavior is undefined.
- 15337 • An optional length modifier that specifies the size of the argument.
- 15338 • A *conversion specifier* wide character that indicates the type of conversion to be applied.

15339 A field width, or precision, or both, may be indicated by an asterisk ('\*'). In this case an  
15340 argument of type `int` supplies the field width or precision. Applications shall ensure that  
15341 arguments specifying field width, or precision, or both appear in that order before the argument,  
15342 if any, to be converted. A negative field width is taken as a '-' flag followed by a positive field  
15343 XSI width. A negative precision is taken as if the precision were omitted. In format wide-character  
15344 strings containing the "%n\$" form of a conversion specification, a field width or precision may  
15345 be indicated by the sequence "\*m\$", where *m* is a decimal integer in the range  
15346 [1,{NL\_ARGMAX}] giving the position in the argument list (after the format argument) of an  
15347 integer argument containing the field width or precision, for example:

```
15348 wprintf(L"%1$d:%2$.*3$d:%4$.*3$d\n", hour, min, precision, sec);
```

15349 The flag wide characters and their meanings are:

- 15350 XSI ' The integer portion of the result of a decimal conversion (%i, %d, %u, %f, %F, %g, or %G)  
15351 shall be formatted with thousands' grouping wide characters. For other conversions,  
15352 the behavior is undefined. The numeric grouping wide character is used.
- 15353 - The result of the conversion shall be left-justified within the field. The conversion shall  
15354 be right-justified if this flag is not specified.
- 15355 + The result of a signed conversion shall always begin with a sign ('+' or '-'). The  
15356 conversion shall begin with a sign only when a negative value is converted if this flag is  
15357 not specified.
- 15358 <space> If the first wide character of a signed conversion is not a sign, or if a signed conversion  
15359 results in no wide characters, a <space> shall be prefixed to the result. This means that  
15360 if the <space> and '+' flags both appear, the <space> flag shall be ignored.
- 15361 # Specifies that the value is to be converted to an alternative form. For o conversion, it  
15362 increases the precision (if necessary) to force the first digit of the result to be 0. For x or  
15363 X conversion specifiers, a non-zero result shall have 0x (or 0X) prefixed to it. For a, A, e,  
15364 E, f, F, g, and G conversion specifiers, the result shall always contain a radix character,  
15365 even if no digits follow it. Without this flag, a radix character appears in the result of  
15366 these conversions only if a digit follows it. For g and G conversion specifiers, trailing  
15367 zeros shall *not* be removed from the result as they normally are. For other conversion

15368                   specifiers, the behavior is undefined. |

15369           0       For `d`, `i`, `o`, `u`, `x`, `X`, `a`, `A`, `e`, `E`, `f`, `F`, `g`, and `G` conversion specifiers, leading zeros |  
15370                   (following any indication of sign or base) are used to pad to the field width; no space |  
15371                   padding is performed. If the `'0'` and `'-'` flags both appear, the `'0'` flag shall be |  
15372                   ignored. For `d`, `i`, `o`, `u`, `x`, and `X` conversion specifiers, if a precision is specified, the `'0'` |  
15373                   flag shall be ignored. If the `'0'` and `'\''` flags both appear, the grouping wide |  
15374                   characters are inserted before zero padding. For other conversions, the behavior is |  
15375                   undefined.

15376           The length modifiers and their meanings are:

15377           hh       Specifies that a following `d`, `i`, `o`, `u`, `x`, or `X` conversion specifier applies to a **signed char** |  
15378                   or **unsigned char** argument (the argument will have been promoted according to the |  
15379                   integer promotions, but its value shall be converted to **signed char** or **unsigned char** |  
15380                   before printing); or that a following `n` conversion specifier applies to a pointer to a |  
15381                   **signed char** argument.

15382           h       Specifies that a following `d`, `i`, `o`, `u`, `x`, or `X` conversion specifier applies to a **short** or |  
15383                   **unsigned short** argument (the argument will have been promoted according to the |  
15384                   integer promotions, but its value shall be converted to **short** or **unsigned short** before |  
15385                   printing); or that a following `n` conversion specifier applies to a pointer to a **short** |  
15386                   argument.

15387           l (ell)   Specifies that a following `d`, `i`, `o`, `u`, `x`, or `X` conversion specifier applies to a **long** or |  
15388                   **unsigned long** argument; that a following `n` conversion specifier applies to a pointer to |  
15389                   a **long** argument; that a following `c` conversion specifier applies to a **wint\_t** argument; |  
15390                   that a following `s` conversion specifier applies to a pointer to a **wchar\_t** argument; or |  
15391                   has no effect on a following `a`, `A`, `e`, `E`, `f`, `F`, `g`, or `G` conversion specifier.

15392           ll (ell-ell)   Specifies that a following `d`, `i`, `o`, `u`, `x`, or `X` conversion specifier applies to a **long long** or |  
15393                   **unsigned long long** argument; or that a following `n` conversion specifier applies to a |  
15394                   pointer to a **long long** argument. |  
15395                   

15396           j       Specifies that a following `d`, `i`, `o`, `u`, `x`, or `X` conversion specifier applies to an **intmax\_t** |  
15397                   or **uintmax\_t** argument; or that a following `n` conversion specifier applies to a pointer |  
15398                   to an **intmax\_t** argument.

15399           z       Specifies that a following `d`, `i`, `o`, `u`, `x`, or `X` conversion specifier applies to a **size\_t** or the |  
15400                   corresponding signed integer type argument; or that a following `n` conversion specifier |  
15401                   applies to a pointer to a signed integer type corresponding to **size\_t** argument.

15402           t       Specifies that a following `d`, `i`, `o`, `u`, `x`, or `X` conversion specifier applies to a **ptrdiff\_t** or |  
15403                   the corresponding **unsigned** type argument; or that a following `n` conversion specifier |  
15404                   applies to a pointer to a **ptrdiff\_t** argument.

15405           L       Specifies that a following `a`, `A`, `e`, `E`, `f`, `F`, `g`, or `G` conversion specifier applies to a **long** |  
15406                   **double** argument.

15407           If a length modifier appears with any conversion specifier other than as specified above, the |  
15408                   behavior is undefined.

15409           The conversion specifiers and their meanings are:

15410           d, i       The **int** argument shall be converted to a signed decimal in the style "`[-]ddd`". The |  
15411                   precision specifies the minimum number of digits to appear; if the value being |  
15412                   converted can be represented in fewer digits, it shall be expanded with leading zeros. |  
15413                   The default precision shall be 1. The result of converting zero with an explicit precision |

15414		of zero shall be no wide characters.	
15415	o	The <b>unsigned</b> argument shall be converted to unsigned octal format in the style	
15416		"ddd". The precision specifies the minimum number of digits to appear; if the value	
15417		being converted can be represented in fewer digits, it shall be expanded with leading	
15418		zeros. The default precision shall be 1. The result of converting zero with an explicit	
15419		precision of zero shall be no wide characters.	
15420	u	The <b>unsigned</b> argument shall be converted to unsigned decimal format in the style	
15421		"ddd". The precision specifies the minimum number of digits to appear; if the value	
15422		being converted can be represented in fewer digits, it shall be expanded with leading	
15423		zeros. The default precision shall be 1. The result of converting zero with an explicit	
15424		precision of zero shall be no wide characters.	
15425	x	The <b>unsigned</b> argument shall be converted to unsigned hexadecimal format in the style	
15426		"ddd"; the letters "abcdef" are used. The precision specifies the minimum number	
15427		of digits to appear; if the value being converted can be represented in fewer digits, it	
15428		shall be expanded with leading zeros. The default precision shall be 1. The result of	
15429		converting zero with an explicit precision of zero shall be no wide characters.	
15430	X	Equivalent to the x conversion specifier, except that letters "ABCDEF" are used instead	
15431		of "abcdef".	
15432	f, F	The <b>double</b> argument shall be converted to decimal notation in the style	
15433		"[-]ddd.ddd", where the number of digits after the radix character shall be equal to	
15434		the precision specification. If the precision is missing, it shall be taken as 6; if the	
15435		precision is explicitly zero and no '#' flag is present, no radix character shall appear. If	
15436		a radix character appears, at least one digit shall appear before it. The value shall be	
15437		rounded in an implementation-defined manner to the appropriate number of digits.	
15438		A <b>double</b> argument representing an infinity shall be converted in one of the styles	
15439		"[-]inf" or "[-]infinity"; which style is implementation-defined. A <b>double</b>	
15440		argument representing a NaN shall be converted in one of the styles "[-]nan" or	
15441		"[-]nan( <i>n-char-sequence</i> )"; which style, and the meaning of any <i>n-char-sequence</i> ,	
15442		is implementation-defined. The F conversion specifier produces "INF", "INFINITY",	
15443		or "NaN" instead of "inf", "infinity", or "nan", respectively.	
15444	e, E	The <b>double</b> argument shall be converted in the style "[-]d.dde±dd", where there	
15445		shall be one digit before the radix character (which is non-zero if the argument is non-	
15446		zero) and the number of digits after it shall be equal to the precision; if the precision is	
15447		missing, it shall be taken as 6; if the precision is zero and no '#' flag is present, no	
15448		radix character shall appear. The value shall be rounded in an implementation-defined	
15449		manner to the appropriate number of digits. The E conversion wide character shall	
15450		produce a number with 'E' instead of 'e' introducing the exponent. The exponent	
15451		always shall contain at least two digits. If the value is zero, the exponent shall be zero.	
15452		A <b>double</b> argument representing an infinity or NaN shall be converted in the style of	
15453		an f or F conversion specifier.	
15454	g, G	The <b>double</b> argument shall be converted in the style f or e (or in the style F or E in the	
15455		case of a G conversion specifier), with the precision specifying the number of significant	
15456		digits. If an explicit precision is zero, it shall be taken as 1. The style used depends on	
15457		the value converted; style e (or E) shall be used only if the exponent resulting from	
15458		such a conversion is less than -4 or greater than or equal to the precision. Trailing zeros	
15459		shall be removed from the fractional portion of the result; a radix character shall appear	
15460		only if it is followed by a digit.	

15461		A <b>double</b> argument representing an infinity or NaN shall be converted in the style of	
15462		an <code>f</code> or <code>F</code> conversion specifier.	
15463	a, A	A <b>double</b> argument representing a floating-point number shall be converted in the	
15464		style " <code>[-]0xh.hhhhp±d</code> ", where there shall be one hexadecimal digit (which is non-	
15465		zero if the argument is a normalized floating-point number and is otherwise	
15466		unspecified) before the decimal-point wide character and the number of hexadecimal	
15467		digits after it shall be equal to the precision; if the precision is missing and <code>FLT_RADIX</code>	
15468		is a power of 2, then the precision shall be sufficient for an exact representation of the	
15469		value; if the precision is missing and <code>FLT_RADIX</code> is not a power of 2, then the precision	
15470		shall be sufficient to distinguish values of type <b>double</b> , except that trailing zeros may	
15471		be omitted; if the precision is zero and the <code>'#'</code> flag is not specified, no decimal-point	
15472		wide character shall appear. The letters " <code>abcdef</code> " are used for a conversion and the	
15473		letters " <code>ABCDEF</code> " for A conversion. The A conversion specifier produces a number with	
15474		<code>'X'</code> and <code>'P'</code> instead of <code>'x'</code> and <code>'p'</code> . The exponent shall always contain at least one	
15475		digit, and only as many more digits as necessary to represent the decimal exponent of	
15476		2. If the value is zero, the exponent shall be zero.	
15477		A <b>double</b> argument representing an infinity or NaN shall be converted in the style of	
15478		an <code>f</code> or <code>F</code> conversion specifier.	
15479	c	If no <code>l</code> (ell) qualifier is present, the <b>int</b> argument shall be converted to a wide character	
15480		as if by calling the <code>btowc()</code> function and the resulting wide character shall be written.	
15481		Otherwise, the <b>wint_t</b> argument shall be converted to <b>wchar_t</b> , and written.	
15482	s	If no <code>l</code> (ell) qualifier is present, the application shall ensure that the argument is a	
15483		pointer to a character array containing a character sequence beginning in the initial	
15484		shift state. Characters from the array shall be converted as if by repeated calls to the	
15485		<code>mbrtowc()</code> function, with the conversion state described by an <b>mbstate_t</b> object	
15486		initialized to zero before the first character is converted, and written up to (but not	
15487		including) the terminating null wide character. If the precision is specified, no more	
15488		than that many wide characters shall be written. If the precision is not specified, or is	
15489		greater than the size of the array, the application shall ensure that the array contains a	
15490		null wide character.	
15491		If an <code>l</code> (ell) qualifier is present, the application shall ensure that the argument is a	
15492		pointer to an array of type <b>wchar_t</b> . Wide characters from the array shall be written up	
15493		to (but not including) a terminating null wide character. If no precision is specified, or	
15494		is greater than the size of the array, the application shall ensure that the array contains	
15495		a null wide character. If a precision is specified, no more than that many wide	
15496		characters shall be written.	
15497	p	The application shall ensure that the argument is a pointer to <b>void</b> . The value of the	
15498		pointer shall be converted to a sequence of printable wide characters in an	
15499		implementation-defined manner.	
15500	n	The application shall ensure that the argument is a pointer to an integer into which is	
15501		written the number of wide characters written to the output so far by this call to one of	
15502		the <code>fwprintf()</code> functions. No argument shall be converted, but one shall be consumed. If	
15503		the conversion specification includes any flags, a field width, or a precision, the	
15504		behavior is undefined.	
15505 XSI	C	Equivalent to <code>lc</code> .	
15506 XSI	S	Equivalent to <code>ls</code> .	

- 15507       %       Output a '%' wide character; no argument shall be converted. The entire conversion |  
15508                   specification shall be %%. |
- 15509       If a conversion specification does not match one of the above forms, the behavior is undefined.
- 15510       In no case does a nonexistent or small field width cause truncation of a field; if the result of a |  
15511       conversion is wider than the field width, the field shall be expanded to contain the conversion |  
15512       result. Characters generated by *fwprintf()* and *wprintf()* shall be printed as if *fputwc()* had been |  
15513       called.
- 15514       For a and A conversions, if FLT\_RADIX is not a power of 2 and the result is not exactly |  
15515       representable in the given precision, the result should be one of the two adjacent numbers in |  
15516       hexadecimal floating style with the given precision, with the extra stipulation that the error |  
15517       should have a correct sign for the current rounding direction. |
- 15518       For e, E, f, F, g, and G conversion specifiers, if the number of significant decimal digits is at most |  
15519       DECIMAL\_DIG, then the result should be correctly rounded. If the number of significant |  
15520       decimal digits is more than DECIMAL\_DIG but the source value is exactly representable with |  
15521       DECIMAL\_DIG digits, then the result should be an exact representation with trailing zeros. |  
15522       Otherwise, the source value is bounded by two adjacent decimal strings  $L < U$ , both having |  
15523       DECIMAL\_DIG significant digits; the value of the resultant decimal string  $D$  should satisfy  $L \leq$  |  
15524        $D \leq U$ , with the extra stipulation that the error should have a correct sign for the current |  
15525       rounding direction.
- 15526 CX       The *st\_ctime* and *st\_mtime* fields of the file shall be marked for update between the call to a |  
15527       successful execution of *fwprintf()* or *wprintf()* and the next successful completion of a call to |  
15528       *fflush()* or *fclose()* on the same stream, or a call to *exit()* or *abort()*.
- 15529 **RETURN VALUE**
- 15530       Upon successful completion, these functions shall return the number of wide characters |  
15531       transmitted, excluding the terminating null wide character in the case of *swprintf()*, or a negative |  
15532 CX       value if an output error was encountered, and set *errno* to indicate the error.
- 15533       If  $n$  or more wide characters were requested to be written, *swprintf()* shall return a negative |  
15534 CX       value, and set *errno* to indicate the error.
- 15535 **ERRORS**
- 15536       For the conditions under which *fwprintf()* and *wprintf()* fail and may fail, refer to *fputwc()*.
- 15537       In addition, all forms of *fwprintf()* may fail if:
- 15538 XSI       [EILSEQ]       A wide-character code that does not correspond to a valid character has been |  
15539                   detected.
- 15540 XSI       [EINVAL]       There are insufficient arguments.
- 15541       In addition, *wprintf()* and *fwprintf()* may fail if:
- 15542 XSI       [ENOMEM]       Insufficient storage space is available.

15543 **EXAMPLES**

15544 To print the language-independent date and time format, the following statement could be used:

```
15545 wprintf(format, weekday, month, day, hour, min);
```

15546 For American usage, *format* could be a pointer to the wide-character string:

```
15547 L"%s, %s %d, %d:%.2d\n"
```

15548 producing the message:

```
15549 Sunday, July 3, 10:02
```

15550 whereas for German usage, *format* could be a pointer to the wide-character string:

```
15551 L"%1$s, %3$d. %2$s, %4$d:%5$.2d\n"
```

15552 producing the message:

```
15553 Sonntag, 3. Juli, 10:02
```

15554 **APPLICATION USAGE**

15555 None.

15556 **RATIONALE**

15557 None.

15558 **FUTURE DIRECTIONS**

15559 None.

15560 **SEE ALSO**

15561 *btowc()*, *fputwc()*, *fwscanf()*, *mbrtowc()*, *setlocale()*, the Base Definitions volume of  
 15562 IEEE Std 1003.1-200x, `<stdio.h>`, `<wchar.h>`, the Base Definitions volume of  
 15563 IEEE Std 1003.1-200x, Chapter 7, Locale

15564 **CHANGE HISTORY**

15565 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995  
 15566 (E).

15567 **Issue 6**

15568 The Open Group Corrigendum U040/1 is applied to the RETURN VALUE section, describing  
 15569 the case if *n* or more wide characters are requested to be written using *swprintf()*.

15570 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

15571 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 15572 • The prototypes for *fwprintf()*, *swprintf()*, and *wprintf()* are updated.
- 15573 • The DESCRIPTION is updated. |
- 15574 • The hh, ll, j, t, and z length modifiers are added. |
- 15575 • The a, A, and F conversion characters are added. |
- 15576 • XSI shading is removed from the description of character string representations of infinity |
- 15577 and NaN floating-point values. |

15578 The DESCRIPTION is updated to use the terms “conversion specifier” and “conversion  
 15579 specification” consistently. |

15580 ISO/IEC 9899:1999 standard, Technical Corrigendum No. 1 is incorporated. |

15581 **NAME**

15582 fwrite — binary output

15583 **SYNOPSIS**

15584 #include &lt;stdio.h&gt;

15585 size\_t fwrite(const void \*restrict ptr, size\_t size, size\_t nitems,  
15586 FILE \*restrict stream);15587 **DESCRIPTION**15588 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
15589 conflict between the requirements described here and the ISO C standard is unintentional. This  
15590 volume of IEEE Std 1003.1-200x defers to the ISO C standard.15591 The *fwrite()* function shall write, from the array pointed to by *ptr*, up to *nitems* elements whose  
15592 size is specified by *size*, to the stream pointed to by *stream*. For each object, *size* calls shall be  
15593 made to the *fputc()* function, taking the values (in order) from an array of **unsigned char** exactly  
15594 overlaying the object. The file-position indicator for the stream (if defined) shall be advanced by  
15595 the number of bytes successfully written. If an error occurs, the resulting value of the file-  
15596 position indicator for the stream is unspecified.15597 CX The *st\_ctime* and *st\_mtime* fields of the file shall be marked for update between the successful  
15598 execution of *fwrite()* and the next successful completion of a call to *fflush()* or *fclose()* on the  
15599 same stream, or a call to *exit()* or *abort()*.15600 **RETURN VALUE**15601 The *fwrite()* function shall return the number of elements successfully written, which may be  
15602 less than *nitems* if a write error is encountered. If *size* or *nitems* is 0, *fwrite()* shall return 0 and the  
15603 state of the stream remains unchanged. Otherwise, if a write error occurs, the error indicator for  
15604 CX the stream shall be set, and *errno* shall be set to indicate the error.15605 **ERRORS**15606 Refer to *fputc()*.15607 **EXAMPLES**

15608 None.

15609 **APPLICATION USAGE**15610 Because of possible differences in element length and byte ordering, files written using *fwrite()*  
15611 are application-dependent, and possibly cannot be read using *fread()* by a different application  
15612 or by the same application on a different processor.15613 **RATIONALE**

15614 None.

15615 **FUTURE DIRECTIONS**

15616 None.

15617 **SEE ALSO**15618 *ferror()*, *fopen()*, *printf()*, *putc()*, *puts()*, *write()*, the Base Definitions volume of  
15619 IEEE Std 1003.1-200x, <stdio.h>15620 **CHANGE HISTORY**

15621 First released in Issue 1. Derived from Issue 1 of the SVID.

15622 **Issue 6**

15623 Extensions beyond the ISO C standard are now marked.

15624 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

15625

- The *fwrite()* prototype is updated.

15626

- The DESCRIPTION is updated to clarify how the data is written out using *fputc()*.

## 15627 NAME

15628 fwscanf, swscanf, wscanf — convert formatted wide-character input

## 15629 SYNOPSIS

15630 #include &lt;stdio.h&gt;

15631 #include &lt;wchar.h&gt;

15632 int fwscanf(FILE \*restrict stream, const wchar\_t \*restrict format, ... );

15633 int swscanf(const wchar\_t \*restrict ws,

15634 const wchar\_t \*restrict format, ... );

15635 int wscanf(const wchar\_t \*restrict format, ... );

## 15636 DESCRIPTION

15637 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 15638 conflict between the requirements described here and the ISO C standard is unintentional. This  
 15639 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

15640 The *fwscanf()* function shall read from the named input *stream*. The *wscanf()* function shall read  
 15641 from the standard input stream *stdin*. The *swscanf()* function shall read from the wide-character  
 15642 string *ws*. Each function reads wide characters, interprets them according to a format, and stores  
 15643 the results in its arguments. Each expects, as arguments, a control wide-character string *format*  
 15644 described below, and a set of *pointer* arguments indicating where the converted input should be  
 15645 stored. The result is undefined if there are insufficient arguments for the format. If the format is  
 15646 exhausted while arguments remain, the excess arguments are evaluated but are otherwise  
 15647 ignored.

15648 XSI Conversions can be applied to the *n*th argument after the *format* in the argument list, rather than  
 15649 to the next unused argument. In this case, the conversion specifier wide character % (see below)  
 15650 is replaced by the sequence "%n\$", where *n* is a decimal integer in the range [1,{NL\_ARGMAX}].  
 15651 This feature provides for the definition of format wide-character strings that select arguments in  
 15652 an order appropriate to specific languages. In format wide-character strings containing the  
 15653 "%n\$" form of conversion specifications, it is unspecified whether numbered arguments in the  
 15654 argument list can be referenced from the format wide-character string more than once.

15655 The *format* can contain either form of a conversion specification—that is, % or "%n\$"—but the  
 15656 two forms cannot normally be mixed within a single *format* wide-character string. The only  
 15657 exception to this is that %% or %\* can be mixed with the "%n\$" form. When numbered  
 15658 argument specifications are used, specifying the *N*th argument requires that all the leading  
 15659 arguments, from the first to the (*N*–1)th, are pointers.

15660 CX The *fwscanf()* function in all its forms allows for detection of a language-dependent radix  
 15661 character in the input string, encoded as a wide-character value. The radix character is defined in  
 15662 the program's locale (category *LC\_NUMERIC*). In the POSIX locale, or in a locale where the  
 15663 radix character is not defined, the radix character shall default to a period ( '.' ).

15664 The *format* is a wide-character string composed of zero or more directives. Each directive is  
 15665 composed of one of the following: one or more white-space wide characters (<space>s, <tab>s,  
 15666 <newline>s, <vertical-tab>s, or <form-feed>s); an ordinary wide character (neither '%' nor a  
 15667 white-space character); or a conversion specification. Each conversion specification is introduced  
 15668 XSI by a '%' or the sequence "%n\$" after which the following appear in sequence:

- 15669 • An optional assignment-suppressing character ' \* '.
- 15670 • An optional non-zero decimal integer that specifies the maximum field width.
- 15671 • An optional length modifier that specifies the size of the receiving object.

15672           • A conversion specifier wide character that specifies the type of conversion to be applied. The  
15673           valid conversion specifiers are described below.

15674           The *fwscanf()* functions shall execute each directive of the format in turn. If a directive fails, as  
15675           detailed below, the function shall return. Failures are described as input failures (due to the  
15676           unavailability of input bytes) or matching failures (due to inappropriate input).

15677           A directive composed of one or more white-space wide characters is executed by reading input  
15678           until no more valid input can be read, or up to the first wide character which is not a white-  
15679           space wide character, which remains unread.

15680           A directive that is an ordinary wide character shall be executed as follows. The next wide  
15681           character is read from the input and compared with the wide character that comprises the  
15682           directive; if the comparison shows that they are not equivalent, the directive shall fail, and the  
15683           differing and subsequent wide characters remain unread. Similarly, if end-of-file, an encoding  
15684           error, or a read error prevents a wide character from being read, the directive shall fail.

15685           A directive that is a conversion specification defines a set of matching input sequences, as  
15686           described below for each conversion wide character. A conversion specification is executed in  
15687           the following steps.

15688           Input white-space wide characters (as specified by *iswspace()*) shall be skipped, unless the  
15689           conversion specification includes a `[`, `c`, or `n` conversion specifier.

15690           An item shall be read from the input, unless the conversion specification includes an `n`  
15691           conversion specifier wide character. An input item is defined as the longest sequence of input  
15692           wide characters, not exceeding any specified field width, which is an initial subsequence of a  
15693           matching sequence. The first wide character, if any, after the input item shall remain unread. If  
15694           the length of the input item is zero, the execution of the conversion specification shall fail; this  
15695           condition is a matching failure, unless end-of-file, an encoding error, or a read error prevented  
15696           input from the stream, in which case it is an input failure.

15697           Except in the case of a `%` conversion specifier, the input item (or, in the case of a `%n` conversion  
15698           specification, the count of input wide characters) shall be converted to a type appropriate to the  
15699           conversion wide character. If the input item is not a matching sequence, the execution of the  
15700           conversion specification shall fail; this condition is a matching failure. Unless assignment  
15701           suppression was indicated by a `'*'`, the result of the conversion shall be placed in the object  
15702           pointed to by the first argument following the *format* argument that has not already received a  
15703 XSI           conversion result if the conversion specification is introduced by `%`, or in the *n*th argument if  
15704           introduced by the wide-character sequence `"%n$"`. If this object does not have an appropriate  
15705           type, or if the result of the conversion cannot be represented in the space provided, the behavior  
15706           is undefined.

15707           The length modifiers and their meanings are:

15708           hh       Specifies that a following `d`, `i`, `o`, `u`, `x`, `X`, or `n` conversion specifier applies to an  
15709           argument with type pointer to **signed char** or **unsigned char**.

15710           h        Specifies that a following `d`, `i`, `o`, `u`, `x`, `X`, or `n` conversion specifier applies to an  
15711           argument with type pointer to **short** or **unsigned short**.

15712           l (ell) Specifies that a following `d`, `i`, `o`, `u`, `x`, `X`, or `n` conversion specifier applies to an  
15713           argument with type pointer to **long** or **unsigned long**; that a following `a`, `A`, `e`, `E`, `f`, `F`, `g`,  
15714           or `G` conversion specifier applies to an argument with type pointer to **double**; or that a  
15715           following `c`, `s`, or `[` conversion specifier applies to an argument with type pointer to  
15716           **wchar\_t**.

15717	ll (ell-ell)	
15718		Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an
15719		argument with type pointer to <b>long long</b> or <b>unsigned long long</b> .
15720	j	Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an
15721		argument with type pointer to <b>intmax_t</b> or <b>uintmax_t</b> .
15722	z	Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an
15723		argument with type pointer to <b>size_t</b> or the corresponding signed integer type.
15724	t	Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an
15725		argument with type pointer to <b>ptrdiff_t</b> or the corresponding <b>unsigned</b> type.
15726	L	Specifies that a following a, A, e, E, f, F, g, or G conversion specifier applies to an
15727		argument with type pointer to <b>long double</b> .
15728		If a length modifier appears with any conversion specifier other than as specified above, the
15729		behavior is undefined.
15730		The following conversion specifier wide characters are valid:
15731	d	Matches an optionally signed decimal integer, whose format is the same as expected for
15732		the subject sequence of <i>wcstol()</i> with the value 10 for the <i>base</i> argument. In the absence
15733		of a size modifier, the application shall ensure that the corresponding argument is a
15734		pointer to <b>int</b> .
15735	i	Matches an optionally signed integer, whose format is the same as expected for the
15736		subject sequence of <i>wcstol()</i> with 0 for the <i>base</i> argument. In the absence of a size
15737		modifier, the application shall ensure that the corresponding argument is a pointer to
15738		<b>int</b> .
15739	o	Matches an optionally signed octal integer, whose format is the same as expected for
15740		the subject sequence of <i>wcstoul()</i> with the value 8 for the <i>base</i> argument. In the absence
15741		of a size modifier, the application shall ensure that the corresponding argument is a
15742		pointer to <b>unsigned</b> .
15743	u	Matches an optionally signed decimal integer, whose format is the same as expected for
15744		the subject sequence of <i>wcstoul()</i> with the value 10 for the <i>base</i> argument. In the absence
15745		of a size modifier, the application shall ensure that the corresponding argument is a
15746		pointer to <b>unsigned</b> .
15747	x	Matches an optionally signed hexadecimal integer, whose format is the same as
15748		expected for the subject sequence of <i>wcstoul()</i> with the value 16 for the <i>base</i> argument.
15749		In the absence of a size modifier, the application shall ensure that the corresponding
15750		argument is a pointer to <b>unsigned</b> .
15751	a, e, f, g	
15752		Matches an optionally signed floating-point number, infinity, or NaN whose format is
15753		the same as expected for the subject sequence of <i>wcstod()</i> . In the absence of a size
15754		modifier, the application shall ensure that the corresponding argument is a pointer to
15755		<b>float</b> .
15756		If the <i>fwprintf()</i> family of functions generates character string representations for
15757		infinity and NaN (a symbolic entity encoded in floating-point format) to support
15758		IEEE Std 754-1985, the <i>fwscanf()</i> family of functions shall recognize them as input.
15759	s	Matches a sequence of non white-space wide characters. If no l (ell) qualifier is present,
15760		characters from the input field shall be converted as if by repeated calls to the
15761		<i>wcrtomb()</i> function, with the conversion state described by an <b>mbstate_t</b> object

15762 initialized to zero before the first wide character is converted. The application shall  
 15763 ensure that the corresponding argument is a pointer to a character array large enough  
 15764 to accept the sequence and the terminating null character, which shall be added  
 15765 automatically.

15766 Otherwise, the application shall ensure that the corresponding argument is a pointer to  
 15767 an array of **wchar\_t** large enough to accept the sequence and the terminating null wide  
 15768 character, which shall be added automatically.

15769 [ Matches a non-empty sequence of wide characters from a set of expected wide  
 15770 characters (the *scanset*). If no **l** (ell) qualifier is present, wide characters from the input  
 15771 field shall be converted as if by repeated calls to the *wcrtomb()* function, with the  
 15772 conversion state described by an **mbstate\_t** object initialized to zero before the first  
 15773 wide character is converted. The application shall ensure that the corresponding  
 15774 argument is a pointer to a character array large enough to accept the sequence and the  
 15775 terminating null character, which shall be added automatically.

15776 If an **l** (ell) qualifier is present, the application shall ensure that the corresponding  
 15777 argument is a pointer to an array of **wchar\_t** large enough to accept the sequence and  
 15778 the terminating null wide character, which shall be added automatically.

15779 The conversion specification includes all subsequent wide characters in the *format*  
 15780 string up to and including the matching right square bracket (']'). The wide  
 15781 characters between the square brackets (the *scanlist*) comprise the scanset, unless the  
 15782 wide character after the left square bracket is a circumflex ('^'), in which case the  
 15783 scanset contains all wide characters that do not appear in the scanlist between the  
 15784 circumflex and the right square bracket. If the conversion specification begins with  
 15785 "[ ]" or "[ ^ ]", the right square bracket is included in the scanlist and the next right  
 15786 square bracket is the matching right square bracket that ends the conversion  
 15787 specification; otherwise, the first right square bracket is the one that ends the  
 15788 conversion specification. If a '-' is in the scanlist and is not the first wide character,  
 15789 nor the second where the first wide character is a '^', nor the last wide character, the  
 15790 behavior is implementation-defined.

15791 **c** Matches a sequence of wide characters of exactly the number specified by the field  
 15792 width (1 if no field width is present in the conversion specification).

15793 If no **l** (ell) length modifier is present, characters from the input field shall be converted  
 15794 as if by repeated calls to the *wcrtomb()* function, with the conversion state described by  
 15795 an **mbstate\_t** object initialized to zero before the first wide character is converted. The  
 15796 corresponding argument shall be a pointer to the initial element of a character array  
 15797 large enough to accept the sequence. No null character is added.

15798 If an **l** (ell) length modifier is present, the corresponding argument shall be a pointer to  
 15799 the initial element of an array of **wchar\_t** large enough to accept the sequence. No null  
 15800 wide character is added.

15801 Otherwise, the application shall ensure that the corresponding argument is a pointer to  
 15802 an array of **wchar\_t** large enough to accept the sequence. No null wide character is  
 15803 added.

15804 **p** Matches an implementation-defined set of sequences, which shall be the same as the set  
 15805 of sequences that is produced by the **%p** conversion specification of the corresponding  
 15806 *fwprintf()* functions. The application shall ensure that the corresponding argument is a  
 15807 pointer to a pointer to **void**. The interpretation of the input item is implementation-  
 15808 defined. If the input item is a value converted earlier during the same program  
 15809 execution, the pointer that results shall compare equal to that value; otherwise, the

15810		behavior of the %p conversion is undefined.
15811	n	No input is consumed. The application shall ensure that the corresponding argument is a pointer to the integer into which is to be written the number of wide characters read from the input so far by this call to the <i>fwscanf()</i> functions. Execution of a %n conversion specification shall not increment the assignment count returned at the completion of execution of the function. No argument shall be converted, but one shall be consumed. If the conversion specification includes an assignment-suppressing wide character or a field width, the behavior is undefined.
15812		
15813		
15814		
15815		
15816		
15817		
15818	XSI	C Equivalent to <code>lc</code> .
15819	XSI	S Equivalent to <code>ls</code> .
15820	%	Matches a single '%' wide character; no conversion or assignment shall occur. The complete conversion specification shall be %%.
15821		
15822		If a conversion specification is invalid, the behavior is undefined.
15823		The conversion specifiers A, E, F, G, and X are also valid and shall be equivalent to, respectively,
15824		a, e, f, g, and x.
15825		If end-of-file is encountered during input, conversion is terminated. If end-of-file occurs before any wide characters matching the current conversion specification (except for %n) have been read (other than leading white-space, where permitted), execution of the current conversion specification shall terminate with an input failure. Otherwise, unless execution of the current conversion specification is terminated with a matching failure, execution of the following conversion specification (if any) shall be terminated with an input failure.
15826		
15827		
15828		
15829		
15830		
15831		Reaching the end of the string in <i>swscanf()</i> shall be equivalent to encountering end-of-file for <i>fwscanf()</i> .
15832		
15833		If conversion terminates on a conflicting input, the offending input shall be left unread in the input. Any trailing white space (including <newline>) shall be left unread unless matched by a conversion specification. The success of literal matches and suppressed assignments is only directly determinable via the %n conversion specification.
15834		
15835		
15836		
15837	CX	The <i>fwscanf()</i> and <i>wscanf()</i> functions may mark the <i>st_atime</i> field of the file associated with <i>stream</i> for update. The <i>st_atime</i> field shall be marked for update by the first successful execution of <i>fgetc()</i> , <i>fgetwc()</i> , <i>fgets()</i> , <i>fgetws()</i> , <i>fread()</i> , <i>getc()</i> , <i>getwc()</i> , <i>getchar()</i> , <i>getwchar()</i> , <i>gets()</i> , <i>fscanf()</i> , or <i>fwscanf()</i> using <i>stream</i> that returns data not supplied by a prior call to <i>ungetc()</i> .
15838		
15839		
15840		
15841		<b>RETURN VALUE</b>
15842		Upon successful completion, these functions shall return the number of successfully matched and assigned input items; this number can be zero in the event of an early matching failure. If the input ends before the first matching failure or conversion, EOF shall be returned. If a read error occurs the error indicator for the stream is set, EOF shall be returned, and <i>errno</i> shall be set to indicate the error.
15843		
15844		
15845	CX	
15846		
15847		<b>ERRORS</b>
15848		For the conditions under which the <i>fwscanf()</i> functions shall fail and may fail, refer to <i>fgetwc()</i> .
15849		In addition, <i>fwscanf()</i> may fail if:
15850	XSI	[EILSEQ] Input byte sequence does not form a valid character.
15851	XSI	[EINVAL] There are insufficient arguments.

15852 **EXAMPLES**

15853 The call:

```
15854 int i, n; float x; char name[50];
15855 n = wscanf(L"%d%f%s", &i, &x, name);
```

15856 with the input line:

15857 25 54.32E-1 Hamster

15858 assigns to *n* the value 3, to *i* the value 25, to *x* the value 5.432, and *name* contains the string  
15859 "Hamster".

15860 The call:

```
15861 int i; float x; char name[50];
15862 (void) wscanf(L"%2d%f*d %[0123456789]", &i, &x, name);
```

15863 with input:

15864 56789 0123 56a72

15865 assigns 56 to *i*, 789.0 to *x*, skip 0123, and place the string "56\0" in *name*. The next call to  
15866 *getchar()* shall return the character 'a'.

15867 **APPLICATION USAGE**

15868 In format strings containing the '%' form of conversion specifications, each argument in the  
15869 argument list is used exactly once.

15870 **RATIONALE**

15871 None.

15872 **FUTURE DIRECTIONS**

15873 None.

15874 **SEE ALSO**

15875 *getwc()*, *fwprintf()*, *setlocale()*, *wcstod()*, *wcstol()*, *wcstoul()*, *wcrtomb()*, the Base Definitions  
15876 volume of IEEE Std 1003.1-200x, <**langinfo.h**>, <**stdio.h**>, <**wchar.h**>, the Base Definitions  
15877 volume of IEEE Std 1003.1-200x, Chapter 7, Locale

15878 **CHANGE HISTORY**

15879 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995  
15880 (E).

15881 **Issue 6**

15882 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

15883 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 15884 • The prototypes for *fwscanf()* and *swscanf()* are updated.
- 15885 • The DESCRIPTION is updated. |
- 15886 • The hh, ll, j, t, and z length modifiers are added. |
- 15887 • The a, A, and F conversion characters are added. |

15888 The DESCRIPTION is updated to use the terms “conversion specifier” and “conversion  
15889 specification” consistently.

15890 **NAME**

15891       gai\_strerror — address and name information error description

15892 **SYNOPSIS**

15893       #include &lt;netdb.h&gt;

15894       const char \*gai\_strerror(int *ecode*);15895 **DESCRIPTION**15896       The *gai\_strerror()* function shall return a text string describing an error value for the *getaddrinfo()*  
15897       and *getnameinfo()* functions listed in the <netdb.h> header.15898       When the *ecode* argument is one of the following values listed in the <netdb.h> header:

15899           [EAI\_AGAIN]

15900           [EAI\_BADFLAGS]

15901           [EAI\_FAIL]

15902           [EAI\_FAMILY]

15903           [EAI\_MEMORY]

15904           [EAI\_NONAME]

15905           [EAI\_SERVICE]

15906           [EAI\_SOCKTYPE]

15907           [EAI\_SYSTEM]

15908       the function return value shall point to a string describing the error. If the argument is not one  
15909       of those values, the function shall return a pointer to a string whose contents indicate an  
15910       unknown error.15911 **RETURN VALUE**15912       Upon successful completion, *gai\_strerror()* shall return a pointer to an implementation-defined  
15913       string.15914 **ERRORS**

15915       No errors are defined.

15916 **EXAMPLES**

15917       None.

15918 **APPLICATION USAGE**

15919       None.

15920 **RATIONALE**

15921       None.

15922 **FUTURE DIRECTIONS**

15923       None.

15924 **SEE ALSO**15925       *getaddrinfo()*, the Base Definitions volume of IEEE Std 1003.1-200x, <netdb.h>15926 **CHANGE HISTORY**

15927       First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

15928       The Open Group Base Resolution bwg2001-009 is applied, which changes the return type from  
15929       **char \*** to **const char \***. This is for coordination with the IPnG Working Group.

15930 **NAME**15931 gcvt — convert a floating-point number to a string (**LEGACY**)15932 **SYNOPSIS**

15933 XSI #include &lt;stdlib.h&gt;

15934 char \*gcvt(double value, int ndigit, char \*buf);

15935

15936 **DESCRIPTION**15937 Refer to *ecvt()*.

15938 **NAME**

15939       getaddrinfo — get address information

15940 **SYNOPSIS**

15941       #include <sys/socket.h>

15942       #include <netdb.h>

15943       int getaddrinfo(const char \*restrict *nodename*,

15944                    const char \*restrict *servname*,

15945                    const struct addrinfo \*restrict *hints*,

15946                    struct addrinfo \*\*restrict *res*);

15947 **DESCRIPTION**

15948       Refer to *freeaddrinfo()*.

15949 **NAME**

15950            getc — get a byte from a stream

15951 **SYNOPSIS**

15952            #include &lt;stdio.h&gt;

15953            int getc(FILE \*stream);

15954 **DESCRIPTION**

15955 cx        The functionality described on this reference page is aligned with the ISO C standard. Any  
15956        conflict between the requirements described here and the ISO C standard is unintentional. This  
15957        volume of IEEE Std 1003.1-200x defers to the ISO C standard.

15958        The *getc()* function shall be equivalent to *fgetc()*, except that if it is implemented as a macro it  
15959        may evaluate *stream* more than once, so the argument should never be an expression with side  
15960        effects.

15961 **RETURN VALUE**15962            Refer to *fgetc()*.15963 **ERRORS**15964            Refer to *fgetc()*.15965 **EXAMPLES**

15966            None.

15967 **APPLICATION USAGE**

15968        If the integer value returned by *getc()* is stored into a variable of type **char** and then compared  
15969        against the integer constant EOF, the comparison may never succeed, because sign-extension of  
15970        a variable of type **char** on widening to integer is implementation-defined.

15971        Since it may be implemented as a macro, *getc()* may treat incorrectly a *stream* argument with  
15972        side effects. In particular, *getc(\*f++)* does not necessarily work as expected. Therefore, use of this  
15973        function should be preceded by "#undef getc" in such situations; *fgetc()* could also be used.

15974 **RATIONALE**

15975            None.

15976 **FUTURE DIRECTIONS**

15977            None.

15978 **SEE ALSO**15979            *fgetc()*, the Base Definitions volume of IEEE Std 1003.1-200x, <stdio.h>15980 **CHANGE HISTORY**

15981            First released in Issue 1. Derived from Issue 1 of the SVID.

15982 **NAME**

15983        getc\_unlocked, getchar\_unlocked, putc\_unlocked, putchar\_unlocked — stdio with explicit client  
15984        locking

15985 **SYNOPSIS**

```
15986 TSF     #include <stdio.h>

15987        int getc_unlocked(FILE *stream);
15988        int getchar_unlocked(void);
15989        int putc_unlocked(int c, FILE *stream);
15990        int putchar_unlocked(int c);
15991
```

15992 **DESCRIPTION**

15993        Versions of the functions *getc()*, *getchar()*, *putc()*, and *putchar()* respectively named  
15994        *getc\_unlocked()*, *getchar\_unlocked()*, *putc\_unlocked()*, and *putchar\_unlocked()* shall be provided  
15995        which are functionally equivalent to the original versions, with the exception that they are not  
15996        required to be implemented in a thread-safe manner. They may only safely be used within a  
15997        scope protected by *flockfile()* (or *ftrylockfile()*) and *funlockfile()*. These functions may safely be  
15998        used in a multi-threaded program if and only if they are called while the invoking thread owns  
15999        the (**FILE\***) object, as is the case after a successful call of the *flockfile()* or *ftrylockfile()* functions.

16000 **RETURN VALUE**

16001        See *getc()*, *getchar()*, *putc()*, and *putchar()*.

16002 **ERRORS**

16003        See *getc()*, *getchar()*, *putc()*, and *putchar()*.

16004 **EXAMPLES**

16005        None.

16006 **APPLICATION USAGE**

16007        Since they may be implemented as macros, *getc\_unlocked()* and *putc\_unlocked()* may treat  
16008        incorrectly a stream argument with side effects. In particular, *getc\_unlocked(\*f++)* and  
16009        *putc\_unlocked(\*f++)* do not necessarily work as expected. Therefore, use of these functions in  
16010        such situations should be preceded by the following statement as appropriate:

```
16011        #undef getc_unlocked
16012        #undef putc_unlocked
```

16013 **RATIONALE**

16014        Some I/O functions are typically implemented as macros for performance reasons (for example,  
16015        *putc()* and *getc()*). For safety, they need to be synchronized, but it is often too expensive to  
16016        synchronize on every character. Nevertheless, it was felt that the safety concerns were more  
16017        important; consequently, the *getc()*, *getchar()*, *putc()*, and *putchar()* functions are required to be  
16018        thread-safe. However, unlocked versions are also provided with names that clearly indicate the  
16019        unsafe nature of their operation but can be used to exploit their higher performance. These  
16020        unlocked versions can be safely used only within explicitly locked program regions, using  
16021        exported locking primitives. In particular, a sequence such as:

```
16022        flockfile(fileptr);
16023        putc_unlocked('1', fileptr);
16024        putc_unlocked('\n', fileptr);
16025        fprintf(fileptr, "Line 2\n");
16026        funlockfile(fileptr);
```

16027        is permissible, and results in the text sequence:

16028 1  
16029 Line 2  
16030 being printed without being interspersed with output from other threads.

16031 It would be wrong to have the standard names such as *getc()*, *putc()*, and so on, map to the  
16032 “faster, but unsafe” rather than the “slower, but safe” versions. In either case, you would still  
16033 want to inspect all uses of *getc()*, *putc()*, and so on, by hand when converting existing code.  
16034 Choosing the safe bindings as the default, at least, results in correct code and maintains the  
16035 “atomicity at the function” invariant. To do otherwise would introduce gratuitous  
16036 synchronization errors into converted code. Other routines that modify the *stdio* (**FILE** \*)  
16037 structures or buffers are also safely synchronized.

16038 Note that there is no need for functions of the form *getc\_locked()*, *putc\_locked()*, and so on, since  
16039 this is the functionality of *getc()*, *putc()*, *et al.* It would be inappropriate to use a feature test  
16040 macro to switch a macro definition of *getc()* between *getc\_locked()* and *getc\_unlocked()*, since the  
16041 ISO C standard requires an actual function to exist, a function whose behavior could not be  
16042 changed by the feature test macro. Also, providing both the *xxx\_locked()* and *xxx\_unlocked()*  
16043 forms leads to the confusion of whether the suffix describes the behavior of the function or the  
16044 circumstances under which it should be used.

16045 Three additional routines, *flockfile()*, *ftrylockfile()*, and *funlockfile()* (which may be macros), are  
16046 provided to allow the user to delineate a sequence of I/O statements that are executed  
16047 synchronously.

16048 The *ungetc()* function is infrequently called relative to the other functions/macros so no  
16049 unlocked variation is needed.

16050 **FUTURE DIRECTIONS**  
16051 None.

16052 **SEE ALSO**  
16053 *getc()*, *getchar()*, *putc()*, *putchar()*, the Base Definitions volume of IEEE Std 1003.1-200x,  
16054 **<stdio.h>**

16055 **CHANGE HISTORY**  
16056 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

16057 **Issue 6**  
16058 These functions are marked as part of the Thread-Safe Functions option.

16059 The Open Group Corrigendum U030/2 is applied, adding APPLICATION USAGE describing  
16060 how applications should be written to avoid the case when the functions are implemented as  
16061 macros.

16062 **NAME**

16063            getchar — get a byte from a stdin stream

16064 **SYNOPSIS**

16065            #include &lt;stdio.h&gt;

16066            int getchar(void);

16067 **DESCRIPTION**

16068 **cx**        The functionality described on this reference page is aligned with the ISO C standard. Any  
16069            conflict between the requirements described here and the ISO C standard is unintentional. This  
16070            volume of IEEE Std 1003.1-200x defers to the ISO C standard.

16071            The *getchar()* function shall be equivalent to *getc(stdin)*.16072 **RETURN VALUE**16073            Refer to *fgetc()*.16074 **ERRORS**16075            Refer to *fgetc()*.16076 **EXAMPLES**

16077            None.

16078 **APPLICATION USAGE**

16079            If the integer value returned by *getchar()* is stored into a variable of type **char** and then  
16080            compared against the integer constant EOF, the comparison may never succeed, because sign-  
16081            extension of a variable of type **char** on widening to integer is implementation-defined.

16082 **RATIONALE**

16083            None.

16084 **FUTURE DIRECTIONS**

16085            None.

16086 **SEE ALSO**16087            *getc()*, the Base Definitions volume of IEEE Std 1003.1-200x, <**stdio.h**>16088 **CHANGE HISTORY**

16089            First released in Issue 1. Derived from Issue 1 of the SVID.

16090 **NAME**

16091        getchar\_unlocked — stdio with explicit client locking

16092 **SYNOPSIS**

16093 TSF     #include &lt;stdio.h&gt;

16094     int getchar\_unlocked(void);

16095

16096 **DESCRIPTION**16097        Refer to *getc\_unlocked()*.

## 16098 NAME

16099 getcontext, setcontext — get and set current user context

## 16100 SYNOPSIS

```
16101 xsi      #include <ucontext.h>
16102
16102      int getcontext(ucontext_t *ucp);
16103      int setcontext(const ucontext_t *ucp);
16104
```

## 16105 DESCRIPTION

16106 The *getcontext()* function shall initialize the structure pointed to by *ucp* to the current user  
 16107 context of the calling thread. The **ucontext\_t** type that *ucp* points to defines the user context and  
 16108 includes the contents of the calling thread's machine registers, the signal mask, and the current  
 16109 execution stack.

16110 The *setcontext()* function shall restore the user context pointed to by *ucp*. A successful call to  
 16111 *setcontext()* shall not return; program execution resumes at the point specified by the *ucp*  
 16112 argument passed to *setcontext()*. The *ucp* argument should be created either by a prior call to  
 16113 *getcontext()* or *makecontext()*, or by being passed as an argument to a signal handler. If the *ucp*  
 16114 argument was created with *getcontext()*, program execution continues as if the corresponding  
 16115 call of *getcontext()* had just returned. If the *ucp* argument was created with *makecontext()*,  
 16116 program execution continues with the function passed to *makecontext()*. When that function  
 16117 returns, the thread shall continue as if after a call to *setcontext()* with the *ucp* argument that was  
 16118 input to *makecontext()*. If the *uc\_link* member of the **ucontext\_t** structure pointed to by the *ucp*  
 16119 argument is equal to 0, then this context is the main context, and the thread shall exit when this  
 16120 context returns. The effects of passing a *ucp* argument obtained from any other source are  
 16121 unspecified.

## 16122 RETURN VALUE

16123 Upon successful completion, *setcontext()* shall not return and *getcontext()* shall return 0;  
 16124 otherwise, a value of -1 shall be returned.

## 16125 ERRORS

16126 No errors are defined.

## 16127 EXAMPLES

16128 Refer to *makecontext()*.

## 16129 APPLICATION USAGE

16130 When a signal handler is executed, the current user context is saved and a new context is  
 16131 created. If the thread leaves the signal handler via *longjmp()*, then it is unspecified whether the  
 16132 context at the time of the corresponding *setjmp()* call is restored and thus whether future calls to  
 16133 *getcontext()* provide an accurate representation of the current context, since the context restored  
 16134 by *longjmp()* does not necessarily contain all the information that *setcontext()* requires. Signal  
 16135 handlers should use *siglongjmp()* or *setcontext()* instead.

16136 Conforming applications should not modify or access the *uc\_mcontext* member of **ucontext\_t**. A  
 16137 conforming application cannot assume that context includes any process-wide static data,  
 16138 possibly including *errno*. Users manipulating contexts should take care to handle these  
 16139 explicitly when required.

16140 Use of contexts to create alternate stacks is not defined by this volume of IEEE Std 1003.1-200x.

16141 **RATIONALE**

16142           None.

16143 **FUTURE DIRECTIONS**

16144           None.

16145 **SEE ALSO**16146           *bsd\_signal()*, *makecontext()*, *setjmp()*, *sigaction()*, *sigaltstack()*, *sigprocmask()*, *sigsetjmp()*, the Base  
16147           Definitions volume of IEEE Std 1003.1-200x, <**ucontext.h**>16148 **CHANGE HISTORY**

16149           First released in Issue 4, Version 2.

16150 **Issue 5**

16151           Moved from X/OPEN UNIX extension to BASE.

16152           The following sentence was removed from the DESCRIPTION: “If the *ucp* argument was passed  
16153           to a signal handler, program execution continues with the program instruction following the  
16154           instruction interrupted by the signal.”

16155 **NAME**

16156       getcwd — get the pathname of the current working directory |

16157 **SYNOPSIS**

16158       #include &lt;unistd.h&gt;

16159       char \*getcwd(char \*buf, size\_t size);

16160 **DESCRIPTION**

16161       The *getcwd()* function shall place an absolute pathname of the current working directory in the |  
16162       array pointed to by *buf*, and return *buf*. The pathname copied to the array shall contain no |  
16163       components that are symbolic links. The *size* argument is the size in bytes of the character array |  
16164       pointed to by the *buf* argument. If *buf* is a null pointer, the behavior of *getcwd()* is unspecified.

16165 **RETURN VALUE**

16166       Upon successful completion, *getcwd()* shall return the *buf* argument. Otherwise, *getcwd()* shall |  
16167       return a null pointer and set *errno* to indicate the error. The contents of the array pointed to by |  
16168       *buf* are then undefined.

16169 **ERRORS**16170       The *getcwd()* function shall fail if:16171       [EINVAL]       The *size* argument is 0.

16172       [ERANGE]       The size argument is greater than 0, but is smaller than the length of the |  
16173       pathname +1. |

16174       The *getcwd()* function may fail if:

16175       [EACCES]       Read or search permission was denied for a component of the pathname. |

16176       [ENOMEM]       Insufficient storage space is available.

16177 **EXAMPLES**16178       **Determining the Absolute Pathname of the Current Working Directory** |

16179       The following example returns a pointer to an array that holds the absolute pathname of the |  
16180       current working directory. The pointer is returned in the *ptr* variable, which points to the *buf* |  
16181       array where the pathname is stored. |

16182       #include &lt;stdlib.h&gt;

16183       #include &lt;unistd.h&gt;

16184       ...

16185       long size;

16186       char \*buf;

16187       char \*ptr;

16188       size = pathconf(".", \_PC\_PATH\_MAX);

16189       if ((buf = (char \*)malloc((size\_t)size)) != NULL)

16190           ptr = getcwd(buf, (size\_t)size);

16191       ...

16192 **APPLICATION USAGE**

16193       None.

16194 **RATIONALE**

16195 Since the maximum pathname length is arbitrary unless {PATH\_MAX} is defined, an application  
16196 generally cannot supply a *buf* with *size* {{PATH\_MAX}+1}.

16197 Having *getcwd()* take no arguments and instead use the *malloc()* function to produce space for  
16198 the returned argument was considered. The advantage is that *getcwd()* knows how big the  
16199 working directory pathname is and can allocate an appropriate amount of space. But the  
16200 programmer would have to use the *free()* function to free the resulting object, or each use of  
16201 *getcwd()* would further reduce the available memory. Also, *malloc()* and *free()* are used nowhere  
16202 else in this volume of IEEE Std 1003.1-200x. Finally, *getcwd()* is taken from the SVID where it has  
16203 the two arguments used in this volume of IEEE Std 1003.1-200x.

16204 The older function *getwd()* was rejected for use in this context because it had only a buffer  
16205 argument and no *size* argument, and thus had no way to prevent overwriting the buffer, except  
16206 to depend on the programmer to provide a large enough buffer.

16207 On some implementations, if *buf* is a null pointer, *getcwd()* may obtain *size* bytes of memory  
16208 using *malloc()*. In this case, the pointer returned by *getcwd()* may be used as the argument in a  
16209 subsequent call to *free()*. Invoking *getcwd()* with *buf* as a null pointer is not recommended in  
16210 conforming applications.

16211 If a program is operating in a directory where some (grand)parent directory does not permit  
16212 reading, *getcwd()* may fail, as in most implementations it must read the directory to determine  
16213 the name of the file. This can occur if search, but not read, permission is granted in an  
16214 intermediate directory, or if the program is placed in that directory by some more privileged  
16215 process (for example, login). Including the [EACCES] error condition makes the reporting of the  
16216 error consistent and warns the application writer that *getcwd()* can fail for reasons beyond the  
16217 control of the application writer or user. Some implementations can avoid this occurrence (for  
16218 example, by implementing *getcwd()* using *pwd*, where *pwd* is a set-user-root process), thus the  
16219 error was made optional. Since this volume of IEEE Std 1003.1-200x permits the addition of other  
16220 errors, this would be a common addition and yet one that applications could not be expected to  
16221 deal with without this addition.

16222 **FUTURE DIRECTIONS**

16223 None.

16224 **SEE ALSO**

16225 *malloc()*, the Base Definitions volume of IEEE Std 1003.1-200x, <unistd.h>

16226 **CHANGE HISTORY**

16227 First released in Issue 1. Derived from Issue 1 of the SVID.

16228 **Issue 6**

16229 The following new requirements on POSIX implementations derive from alignment with the  
16230 Single UNIX Specification:

- 16231 • The [ENOMEM] optional error condition is added.

16232 **NAME**

16233 getdate — convert user format date and time

16234 **SYNOPSIS**

```
16235 xSI #include <time.h>
16236 struct tm *getdate(const char *string);
16237
```

16238 **DESCRIPTION**

16239 The *getdate()* function shall convert a string representation of a date or time into a broken-down  
 16240 time.

16241 The external variable or macro *getdate\_err* is used by *getdate()* to return error values.

16242 Templates are used to parse and interpret the input string. The templates are contained in a text  
 16243 file identified by the environment variable *DATEMSK*. The *DATEMSK* variable should be set to  
 16244 indicate the full pathname of the file that contains the templates. The first line in the template  
 16245 that matches the input specification is used for interpretation and conversion into the internal  
 16246 time format.

16247 The following conversion specifications shall be supported:

16248	%%	Equivalent to %.	
16249	%a	Abbreviated weekday name.	
16250	%A	Full weekday name.	
16251	%b	Abbreviated month name.	
16252	%B	Full month name.	
16253	%c	Locale's appropriate date and time representation.	
16254	%C	Century number [00,99]; leading zeros are permitted but not required.	
16255	%d	Day of month [01,31]; the leading 0 is optional.	
16256	%D	Date as %m/%d/%y.	
16257	%e	Equivalent to %d.	
16258	%h	Abbreviated month name.	
16259	%H	Hour [00,23].	
16260	%I	Hour [01,12].	
16261	%m	Month number [01,12].	
16262	%M	Minute [00,59].	
16263	%n	Equivalent to <newline>.	
16264	%p	Locale's equivalent of either AM or PM.	
16265	%r	The locale's appropriate representation of time in AM and PM notation. In the POSIX	
16266		locale, this shall be equivalent to %I:%M:%S %p.	
16267	%R	Time as %H:%M.	
16268	%S	Seconds [00,60]. The range goes to 60 (rather than stopping at 59) to allow positive leap	
16269		seconds to be expressed. Since leap seconds cannot be predicted by any algorithm, leap	
16270		second data must come from some external source.	

16271	%t	Equivalent to <tab>.	
16272	%T	Time as %H:%M:%S.	
16273	%w	Weekday number (Sunday = [0,6]).	
16274	%x	Locale's appropriate date representation.	
16275	%X	Locale's appropriate time representation.	
16276	%y	Year within century. When a century is not otherwise specified, values in the range	
16277		[69,99] shall refer to years 1969 to 1999 inclusive, and values in the range [00,68] shall	
16278		refer to years 2000 to 2068 inclusive.	
16279		<b>Note:</b> It is expected that in a future version of IEEE Std 1003.1-200x the default century	
16280		inferred from a 2-digit year will change. (This would apply to all commands	
16281		accepting a 2-digit year as input.)	
16282	%Y	Year as "ccyy" (for example, 2001).	
16283	%Z	Timezone name or no characters if no timezone exists. If the timezone supplied by %Z is	
16284		not the timezone that <i>getdate()</i> expects, an invalid input specification error shall result.	
16285		The <i>getdate()</i> function calculates an expected timezone based on information supplied	
16286		to the function (such as the hour, day, and month).	
16287		The match between the template and input specification performed by <i>getdate()</i> shall be case-	
16288		insensitive.	
16289		The month and weekday names can consist of any combination of upper and lowercase letters.	
16290		The process can request that the input date or time specification be in a specific language by	
16291		setting the <i>LC_TIME</i> category (see <i>setlocale()</i> ).	
16292		Leading 0s are not necessary for the descriptors that allow leading 0s. However, at most two	
16293		digits are allowed for those descriptors, including leading 0s. Extra whitespace in either the	
16294		template file or in <i>string</i> shall be ignored.	
16295		The results are undefined if the conversion specifications %c, %x, and %X include unsupported	
16296		conversion specifications.	
16297		The following rules apply for converting the input specification into the internal format:	
16298		• If %Z is being scanned, then <i>getdate()</i> shall initialize the broken-down time to be the current	
16299		time in the scanned timezone. Otherwise, it shall initialize the broken-down time based on	
16300		the current local time as if <i>localtime()</i> had been called.	
16301		• If only the weekday is given, the day chosen shall be the day, starting with today and moving	
16302		into the future, which first matches the named day.	
16303		• If only the month (and no year) is given, the month chosen shall be the month, starting with	
16304		the current month and moving into the future, which first matches the named month. The	
16305		first day of the month shall be assumed if no day is given.	
16306		• If no hour, minute, and second are given the current hour, minute, and second shall be	
16307		assumed.	
16308		• If no date is given, the hour chosen shall be the hour, starting with the current hour and	
16309		moving into the future, which first matches the named hour.	
16310		If a conversion specification in the <i>DATMSK</i> file does not correspond to one of the conversion	
16311		specifications above, the behavior is unspecified.	
16312		The <i>getdate()</i> function need not be reentrant. A function that is not required to be reentrant is not	
16313		required to be thread-safe.	

## 16314 RETURN VALUE

16315 Upon successful completion, `getdate()` shall return a pointer to a **struct tm**. Otherwise, it shall  
 16316 return a null pointer and set `getdate_err` to indicate the error.

## 16317 ERRORS

16318 The `getdate()` function shall fail in the following cases, setting `getdate_err` to the value shown in  
 16319 the list below. Any changes to `errno` are unspecified.

- 16320 1. The `DATEMSK` environment variable is null or undefined.
- 16321 2. The template file cannot be opened for reading.
- 16322 3. Failed to get file status information.
- 16323 4. The template file is not a regular file.
- 16324 5. An I/O error is encountered while reading the template file.
- 16325 6. Memory allocation failed (not enough memory available).
- 16326 7. There is no line in the template that matches the input.
- 16327 8. Invalid input specification. For example, February 31; or a time is specified that cannot be  
 16328 represented in a `time_t` (representing the time in seconds since the Epoch).

## 16329 EXAMPLES

- 16330 1. The following example shows the possible contents of a template:

```
16331 %m
16332 %A %B %d, %Y, %H:%M:%S
16333 %A
16334 %B
16335 %m/%d/%y %I %p
16336 %d,%m,%Y %H:%M
16337 at %A the %dst of %B in %Y
16338 run job at %I %p,%B %dnd
16339 %A den %d. %B %Y %H.%M Uhr
```

- 16340 2. The following are examples of valid input specifications for the template in Example 1:

```
16341 getdate("10/1/87 4 PM");
16342 getdate("Friday");
16343 getdate("Friday September 18, 1987, 10:30:30");
16344 getdate("24,9,1986 10:30");
16345 getdate("at monday the 1st of december in 1986");
16346 getdate("run job at 3 PM, december 2nd");
```

16347 If the `LC_TIME` category is set to a German locale that includes *freitag* as a weekday name  
 16348 and *oktober* as a month name, the following would be valid:

```
16349 getdate("freitag den 10. oktober 1986 10.30 Uhr");
```

- 16350 3. The following example shows how local date and time specification can be defined in the  
 16351 template:

16352  
16353  
16354  
16355  
16356  
16357

Invocation	Line in Template
getdate("11/27/86")	%m/%d/%y
getdate("27.11.86")	%d.%m.%y
getdate("86-11-27")	%y-%m-%d
getdate("Friday 12:00:00")	%A %H:%M:%S

16358  
16359  
16360  
16361

4. The following examples help to illustrate the above rules assuming that the current date is Mon Sep 22 12:19:47 EDT 1986 and the *LC\_TIME* category is set to the default C locale:

16362  
16363  
16364  
16365  
16366  
16367  
16368  
16369  
16370  
16371  
16372  
16373  
16374  
16375

Input	Line in Template	Date
Mon	%a	Mon Sep 22 12:19:47 EDT 1986
Sun	%a	Sun Sep 28 12:19:47 EDT 1986
Fri	%a	Fri Sep 26 12:19:47 EDT 1986
September	%B	Mon Sep 1 12:19:47 EDT 1986
January	%B	Thu Jan 1 12:19:47 EST 1987
December	%B	Mon Dec 1 12:19:47 EST 1986
Sep Mon	%b %a	Mon Sep 1 12:19:47 EDT 1986
Jan Fri	%b %a	Fri Jan 2 12:19:47 EST 1987
Dec Mon	%b %a	Mon Dec 1 12:19:47 EST 1986
Jan Wed 1989	%b %a %Y	Wed Jan 4 12:19:47 EST 1989
Fri 9	%a %H	Fri Sep 26 09:00:00 EDT 1986
Feb 10:30	%b %H:%S	Sun Feb 1 10:00:30 EST 1987
10:30	%H:%M	Tue Sep 23 10:30:00 EDT 1986
13:30	%H:%M	Mon Sep 22 13:30:00 EDT 1986

#### 16376 APPLICATION USAGE

16377 Although historical versions of *getdate()* did not require that **<time.h>** declare the external  
16378 variable *getdate\_err*, this volume of IEEE Std 1003.1-200x does require it. The standard  
16379 developers encourage applications to remove declarations of *getdate\_err* and instead incorporate  
16380 the declaration by including **<time.h>**.

16381 Applications should use %Y (4-digit years) in preference to %y (2-digit years).

#### 16382 RATIONALE

16383 In standard locales, the conversion specifications %c, %x, and %X do not include unsupported  
16384 conversion specifiers and so the text regarding results being undefined is not a problem in that  
16385 case.

#### 16386 FUTURE DIRECTIONS

16387 None.

#### 16388 SEE ALSO

16389 *ctime()*, *localtime()*, *setlocale()*, *strftime()*, *times()*, the Base Definitions volume of  
16390 IEEE Std 1003.1-200x, **<time.h>**

#### 16391 CHANGE HISTORY

16392 First released in Issue 4, Version 2.

#### 16393 Issue 5

16394 Moved from X/OPEN UNIX extension to BASE.

16395 The last paragraph of the DESCRIPTION is added.

16396 The %C conversion specification is added, and the exact meaning of the %y conversion  
16397 specification is clarified in the DESCRIPTION.



16407 **NAME**

16408           getegid — get the effective group ID

16409 **SYNOPSIS**

16410           #include <unistd.h>

16411           gid\_t getegid(void);

16412 **DESCRIPTION**

16413           The *getegid()* function shall return the effective group ID of the calling process.

16414 **RETURN VALUE**

16415           The *getegid()* function shall always be successful and no return value is reserved to indicate an error.  
16416

16417 **ERRORS**

16418           No errors are defined.

16419 **EXAMPLES**

16420           None.

16421 **APPLICATION USAGE**

16422           None.

16423 **RATIONALE**

16424           None.

16425 **FUTURE DIRECTIONS**

16426           None.

16427 **SEE ALSO**

16428           *geteuid()*, *getgid()*, *getuid()*, *setegid()*, *seteuid()*, *setgid()*, *setregid()*, *setreuid()*, *setuid()*, the Base  
16429           Definitions volume of IEEE Std 1003.1-200x, <sys/types.h>, <unistd.h>

16430 **CHANGE HISTORY**

16431           First released in Issue 1. Derived from Issue 1 of the SVID.

16432 **Issue 6**

16433           In the SYNOPSIS, the optional include of the <sys/types.h> header is removed.

16434           The following new requirements on POSIX implementations derive from alignment with the  
16435           Single UNIX Specification:

- 16436
  - The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was  
16437           required for conforming implementations of previous POSIX specifications, it was not  
16438           required for UNIX applications.

16439 **NAME**

16440            getenv — get value of an environment variable

16441 **SYNOPSIS**

16442            #include <stdlib.h>

16443            char \*getenv(const char \*name);

16444 **DESCRIPTION**

16445 **CX**        The functionality described on this reference page is aligned with the ISO C standard. Any  
16446 conflict between the requirements described here and the ISO C standard is unintentional. This  
16447 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

16448        The *getenv()* function shall search the environment of the calling process (see the Base  
16449 Definitions volume of IEEE Std 1003.1-200x, Chapter 8, Environment Variables) for the  
16450 environment variable *name* if it exists and return a pointer to the value of the environment  
16451 variable. If the specified environment variable cannot be found, a null pointer shall be returned.  
16452 The application shall ensure that it does not modify the string pointed to by the *getenv()*  
16453 function.

16454 **CX**        The string pointed to may be overwritten by a subsequent call to *getenv()*, *setenv()*, or *unsetenv()*,  
16455 but shall not be overwritten by a call to any other function in this volume of  
16456 IEEE Std 1003.1-200x.

16457 **CX**        If the application modifies *environ* or the pointers to which it points, the behavior of *getenv()* is  
16458 undefined.

16459        The *getenv()* function need not be reentrant. A function that is not required to be reentrant is not  
16460 required to be thread-safe.

16461 **RETURN VALUE**

16462        Upon successful completion, *getenv()* shall return a pointer to a string containing the *value* for  
16463 the specified *name*. If the specified name cannot be found in the environment of the calling  
16464 process, a null pointer shall be returned.

16465        The return value from *getenv()* may point to static data which may be overwritten by  
16466 **CX**        subsequent calls to *getenv()*, *setenv()*, or *unsetenv()*.

16467 **XSI**        On XSI-conformant systems, the return value from *getenv()* may point to static data which may  
16468 also be overwritten by subsequent calls to *putenv()*.

16469 **ERRORS**

16470        No errors are defined.

16471 **EXAMPLES**16472        **Getting the Value of an Environment Variable**

16473        The following example gets the value of the *HOME* environment variable.

```
16474        #include <stdlib.h>
16475        ...
16476        const char *name = "HOME";
16477        char *value;
16478        value = getenv(name);
```

16479 **APPLICATION USAGE**

16480 None.

16481 **RATIONALE**

16482 The *clearenv()* function was considered but rejected. The *putenv()* function has now been  
 16483 included for alignment with the Single UNIX Specification.

16484 The *getenv()* function is inherently not reentrant because it returns a value pointing to static  
 16485 data.

16486 Conforming applications are required not to modify *environ* directly, but to use only the  
 16487 functions described here to manipulate the process environment as an abstract object. Thus, the  
 16488 implementation of the environment access functions has complete control over the data  
 16489 structure used to represent the environment (subject to the requirement that *environ* be  
 16490 maintained as a list of strings with embedded equal signs for applications that wish to scan the  
 16491 environment). This constraint allows the implementation to properly manage the memory it  
 16492 allocates, either by using allocated storage for all variables (copying them on the first invocation  
 16493 of *setenv()* or *unsetenv()*), or keeping track of which strings are currently in allocated space and  
 16494 which are not, via a separate table or some other means. This enables the implementation to free  
 16495 any allocated space used by strings (and perhaps the pointers to them) stored in *environ* when  
 16496 *unsetenv()* is called. A C runtime start-up procedure (that which invokes *main()* and perhaps  
 16497 initializes *environ*) can also initialize a flag indicating that none of the environment has yet been  
 16498 copied to allocated storage, or that the separate table has not yet been initialized.

16499 In fact, for higher performance of *getenv()*, the implementation could also maintain a separate  
 16500 copy of the environment in a data structure that could be searched much more quickly (such as  
 16501 an indexed hash table, or a binary tree), and update both it and the linear list at *environ* when  
 16502 *setenv()* or *unsetenv()* is invoked.

16503 Performance of *getenv()* can be important for applications which have large numbers of  
 16504 environment variables. Typically, applications like this use the environment as a resource  
 16505 database of user-configurable parameters. The fact that these variables are in the user's shell  
 16506 environment usually means that any other program that uses environment variables (such as *ls*,  
 16507 which attempts to use *COLUMNS*, or really almost any utility (*LANG*, *LC\_ALL*, and so on) is  
 16508 similarly slowed down by the linear search through the variables.

16509 An implementation that maintains separate data structures, or even one that manages the  
 16510 memory it consumes, is not currently required as it was thought it would reduce consensus  
 16511 among implementors who do not want to change their historical implementations.

16512 The POSIX Threads Extension states that multi-threaded applications must not modify *environ*  
 16513 directly, and that IEEE Std 1003.1-200x is providing functions which such applications can use in  
 16514 the future to manipulate the environment in a thread-safe manner. Thus, moving away from  
 16515 application use of *environ* is desirable from that standpoint as well.

16516 **FUTURE DIRECTIONS**

16517 None.

16518 **SEE ALSO**

16519 *exec*, *putenv()*, *setenv()*, *unsetenv()*, the Base Definitions volume of IEEE Std 1003.1-200x,  
 16520 `<stdlib.h>`, the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 8, Environment  
 16521 Variables

16522 **CHANGE HISTORY**

16523 First released in Issue 1. Derived from Issue 1 of the SVID.

16524 **Issue 5**

16525 Normative text previously in the APPLICATION USAGE section is moved to the RETURN  
16526 VALUE section.

16527 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

16528 **Issue 6**

16529 The following changes were made to align with the IEEE P1003.1a draft standard:

16530 • References added to the new *setenv()* and *unsetenv()* functions.

16531 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

16532 **NAME**

16533           geteuid — get the effective user ID

16534 **SYNOPSIS**

16535           #include <unistd.h>

16536           uid\_t geteuid(void);

16537 **DESCRIPTION**

16538           The *geteuid()* function shall return the effective user ID of the calling process.

16539 **RETURN VALUE**

16540           The *geteuid()* function shall always be successful and no return value is reserved to indicate an error.

16542 **ERRORS**

16543           No errors are defined.

16544 **EXAMPLES**

16545           None.

16546 **APPLICATION USAGE**

16547           None.

16548 **RATIONALE**

16549           None.

16550 **FUTURE DIRECTIONS**

16551           None.

16552 **SEE ALSO**

16553           *getegid()*, *getgid()*, *getuid()*, *setegid()*, *seteuid()*, *setgid()*, *setregid()*, *setreuid()*, *setuid()*, the Base  
16554           Definitions volume of IEEE Std 1003.1-200x, <sys/types.h>, <unistd.h>

16555 **CHANGE HISTORY**

16556           First released in Issue 1. Derived from Issue 1 of the SVID.

16557 **Issue 6**

16558           In the SYNOPSIS, the optional include of the <sys/types.h> header is removed.

16559           The following new requirements on POSIX implementations derive from alignment with the  
16560           Single UNIX Specification:

- 16561           • The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was  
16562           required for conforming implementations of previous POSIX specifications, it was not  
16563           required for UNIX applications.

16564 **NAME**

16565           getgid — get the real group ID

16566 **SYNOPSIS**

16567           #include <unistd.h>

16568           gid\_t getgid(void);

16569 **DESCRIPTION**

16570           The *getgid()* function shall return the real group ID of the calling process.

16571 **RETURN VALUE**

16572           The *getgid()* function shall always be successful and no return value is reserved to indicate an error.

16574 **ERRORS**

16575           No errors are defined.

16576 **EXAMPLES**

16577           None.

16578 **APPLICATION USAGE**

16579           None.

16580 **RATIONALE**

16581           None.

16582 **FUTURE DIRECTIONS**

16583           None.

16584 **SEE ALSO**

16585           *getegid()*, *geteuid()*, *getuid()*, *setegid()*, *seteuid()*, *setgid()*, *setregid()*, *setreuid()*, *setuid()*, the Base Definitions volume of IEEE Std 1003.1-200x, <sys/types.h>, <unistd.h>

16587 **CHANGE HISTORY**

16588           First released in Issue 1. Derived from Issue 1 of the SVID.

16589 **Issue 6**

16590           In the SYNOPSIS, the optional include of the <sys/types.h> header is removed.

16591           The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- 16593           • The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.
- 16594
- 16595

16596 **NAME**

16597           getgrent — get the group database entry

16598 **SYNOPSIS**

16599 xSI       #include &lt;grp.h&gt;

16600           struct group \*getgrent(void);

16601

16602 **DESCRIPTION**16603           Refer to *endgrent()*.

## 16604 NAME

16605 getgrgid, getgrgid\_r — get group database entry for a group ID

## 16606 SYNOPSIS

16607 #include &lt;grp.h&gt;

16608 struct group \*getgrgid(gid\_t gid);

16609 TSF int getgrgid\_r(gid\_t gid, struct group \*grp, char \*buffer,

16610 size\_t bufsize, struct group \*\*result);

16611

## 16612 DESCRIPTION

16613 The *getgrgid()* function shall search the group database for an entry with a matching *gid*.16614 The *getgrgid()* function need not be reentrant. A function that is not required to be reentrant is  
16615 not required to be thread-safe.16616 TSF The *getgrgid\_r()* function shall update the **group** structure pointed to by *grp* and store a pointer |  
16617 to that structure at the location pointed to by *result*. The structure shall contain an entry from |  
16618 the group database with a matching *gid*. Storage referenced by the group structure is allocated |  
16619 from the memory provided with the *buffer* parameter, which is *bufsize* bytes in size. The |  
16620 maximum size needed for this buffer can be determined with the `{_SC_GETGR_R_SIZE_MAX}` |  
16621 *sysconf()* parameter. A NULL pointer shall be returned at the location pointed to by *result* on |  
16622 error or if the requested entry is not found.

## 16623 RETURN VALUE

16624 Upon successful completion, *getgrgid()* shall return a pointer to a **struct group** with the structure  
16625 defined in <grp.h> with a matching entry if one is found. The *getgrgid()* function shall return a  
16626 null pointer if either the requested entry was not found, or an error occurred. On error, *errno*  
16627 shall be set to indicate the error.16628 The return value may point to a static area which is overwritten by a subsequent call to  
16629 *getgrent()*, *getgrgid()*, or *getgrnam()*.16630 TSF If successful, the *getgrgid\_r()* function shall return zero; otherwise, an error number shall be  
16631 returned to indicate the error.

## 16632 ERRORS

16633 The *getgrgid()* and *getgrgid\_r()* functions may fail if:

16634 [EIO] An I/O error has occurred.

16635 [EINTR] A signal was caught during *getgrgid()*.

16636 [EMFILE] {OPEN\_MAX} file descriptors are currently open in the calling process.

16637 [ENFILE] The maximum allowable number of files is currently open in the system.

16638 TSF The *getgrgid\_r()* function may fail if:16639 TSF [ERANGE] Insufficient storage was supplied via *buffer* and *bufsize* to contain the data to  
16640 be referenced by the resulting **group** structure.

16641 **EXAMPLES**16642 **Finding an Entry in the Group Database**

16643 The following example uses *getgrgid()* to search the group database for a group ID that was  
 16644 previously stored in a **stat** structure, then prints out the group name if it is found. If the group is  
 16645 not found, the program prints the numeric value of the group for the entry.

```
16646 #include <sys/types.h>
16647 #include <grp.h>
16648 #include <stdio.h>
16649 ...
16650 struct stat statbuf;
16651 struct group *grp;
16652 ...
16653 if ((grp = getgrgid(statbuf.st_gid)) != NULL)
16654     printf(" %-8.8s", grp->gr_name);
16655 else
16656     printf(" %-8d", statbuf.st_gid);
16657 ...
```

16658 **APPLICATION USAGE**

16659 Applications wishing to check for error situations should set *errno* to 0 before calling *getgrgid()*.  
 16660 If *errno* is set on return, an error occurred.

16661 The *getgrgid\_r()* function is thread-safe and shall return values in a user-supplied buffer instead  
 16662 of possibly using a static data area that may be overwritten by each call.

16663 **RATIONALE**

16664 None.

16665 **FUTURE DIRECTIONS**

16666 None.

16667 **SEE ALSO**

16668 *endgrent()*, *getgrnam()*, the Base Definitions volume of IEEE Std 1003.1-200x, **<grp.h>**,  
 16669 **<limits.h>**, **<sys/types.h>**

16670 **CHANGE HISTORY**

16671 First released in Issue 1. Derived from System V Release 2.0.

16672 **Issue 5**

16673 Normative text previously in the APPLICATION USAGE section is moved to the RETURN  
 16674 VALUE section.

16675 The *getgrgid\_r()* function is included for alignment with the POSIX Threads Extension.

16676 A note indicating that the *getgrgid()* function need not be reentrant is added to the  
 16677 DESCRIPTION.

16678 **Issue 6**

16679 The *getgrgid\_r()* function is marked as part of the Thread-Safe Functions option.

16680 The Open Group Corrigendum U028/3 is applied, correcting text in the DESCRIPTION  
 16681 describing matching the *gid*.

16682 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

16683 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

- 16684 The following new requirements on POSIX implementations derive from alignment with the  
16685 Single UNIX Specification:
- 16686 • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was  
16687 required for conforming implementations of previous POSIX specifications, it was not  
16688 required for UNIX applications.
  - 16689 • In the RETURN VALUE section, the requirement to set *errno* on error is added.
  - 16690 • The [EIO], [EINTR], [EMFILE], and [ENFILE] optional error conditions are added.
- 16691 The APPLICATION USAGE section is updated to include a note on the thread-safe function and  
16692 its avoidance of possibly using a static data area.
- 16693 IEEE PASC Interpretation 1003.1 #116 is applied, changing the description of the size of the  
16694 buffer from *bufsize* characters to bytes.

16695 **NAME**

16696 getgrnam, getgrnam\_r — search group database for a name

16697 **SYNOPSIS**

16698 #include &lt;grp.h&gt;

16699 struct group \*getgrnam(const char \*name);

16700 TSF int getgrnam\_r(const char \*name, struct group \*grp, char \*buffer,

16701 size\_t bufsize, struct group \*\*result);

16702

16703 **DESCRIPTION**16704 The *getgrnam()* function shall search the group database for an entry with a matching *name*.16705 The *getgrnam()* function need not be reentrant. A function that is not required to be reentrant is  
16706 not required to be thread-safe.16707 TSF The *getgrnam\_r()* function shall update the **group** structure pointed to by *grp* and store a pointer |  
16708 to that structure at the location pointed to by *result*. The structure shall contain an entry from |  
16709 the group database with a matching *gid* or *name*. Storage referenced by the group structure is |  
16710 allocated from the memory provided with the *buffer* parameter, which is *bufsize* bytes in size. The  
16711 maximum size needed for this buffer can be determined with the `{_SC_GETGR_R_SIZE_MAX}`  
16712 *sysconf()* parameter. A NULL pointer is returned at the location pointed to by *result* on error or if  
16713 the requested entry is not found.16714 **RETURN VALUE**16715 The *getgrnam()* function shall return a pointer to a **struct group** with the structure defined in  
16716 <grp.h> with a matching entry if one is found. The *getgrnam()* function shall return a null  
16717 pointer if either the requested entry was not found, or an error occurred. On error, *errno* shall be  
16718 set to indicate the error.16719 The return value may point to a static area which is overwritten by a subsequent call to  
16720 *getgrent()*, *getgrgid()*, or *getgrnam()*.16721 TSF If successful, the *getgrnam\_r()* function shall return zero; otherwise, an error number shall be  
16722 returned to indicate the error.16723 **ERRORS**16724 The *getgrnam()* and *getgrnam\_r()* functions may fail if:

16725 [EIO] An I/O error has occurred.

16726 [EINTR] A signal was caught during *getgrnam()*.

16727 [EMFILE] {OPEN\_MAX} file descriptors are currently open in the calling process.

16728 [ENFILE] The maximum allowable number of files is currently open in the system.

16729 The *getgrnam\_r()* function may fail if:16730 TSF [ERANGE] Insufficient storage was supplied via *buffer* and *bufsize* to contain the data to  
16731 be referenced by the resulting **group** structure.

16732 **EXAMPLES**

16733 None.

16734 **APPLICATION USAGE**

16735 Applications wishing to check for error situations should set *errno* to 0 before calling *getgrnam()*.  
 16736 If *errno* is set on return, an error occurred.

16737 The *getgrnam\_r()* function is thread-safe and shall return values in a user-supplied buffer instead  
 16738 of possibly using a static data area that may be overwritten by each call.

16739 **RATIONALE**

16740 None.

16741 **FUTURE DIRECTIONS**

16742 None.

16743 **SEE ALSO**

16744 *endgrent()*, *getgrgid()*, the Base Definitions volume of IEEE Std 1003.1-200x, **<grp.h>**, **<limits.h>**,  
 16745 **<sys/types.h>**

16746 **CHANGE HISTORY**

16747 First released in Issue 1. Derived from System V Release 2.0.

16748 **Issue 5**

16749 Normative text previously in the APPLICATION USAGE section is moved to the RETURN  
 16750 VALUE section.

16751 The *getgrnam\_r()* function is included for alignment with the POSIX Threads Extension.

16752 A note indicating that the *getgrnam()* function need not be reentrant is added to the  
 16753 DESCRIPTION.

16754 **Issue 6**16755 The *getgrnam\_r()* function is marked as part of the Thread-Safe Functions option.

16756 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

16757 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

16758 The following new requirements on POSIX implementations derive from alignment with the  
 16759 Single UNIX Specification:

16760 • The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was  
 16761 required for conforming implementations of previous POSIX specifications, it was not  
 16762 required for UNIX applications.

16763 • In the RETURN VALUE section, the requirement to set *errno* on error is added.

16764 • The [EIO], [EINTR], [EMFILE], and [ENFILE] optional error conditions are added.

16765 The APPLICATION USAGE section is updated to include a note on the thread-safe function and  
 16766 its avoidance of possibly using a static data area.

16767 IEEE PASC Interpretation 1003.1 #116 is applied, changing the description of the size of the  
 16768 buffer from *bufsize* characters to bytes.

16769 **NAME**16770 `getgroups` — get supplementary group IDs16771 **SYNOPSIS**16772 `#include <unistd.h>`16773 `int getgroups(int gidsetsize, gid_t grouplist[]);`16774 **DESCRIPTION**

16775 The `getgroups()` function shall fill in the array `grouplist` with the current supplementary group |  
 16776 IDs of the calling process. It is implementation-defined whether `getgroups()` also returns the |  
 16777 effective group ID in the `grouplist` array.

16778 The `gidsetsize` argument specifies the number of elements in the array `grouplist`. The actual |  
 16779 number of group IDs stored in the array shall be returned. The values of array entries with |  
 16780 indices greater than or equal to the value returned are undefined.

16781 If `gidsetsize` is 0, `getgroups()` shall return the number of group IDs that it would otherwise return |  
 16782 without modifying the array pointed to by `grouplist`.

16783 If the effective group ID of the process is returned with the supplementary group IDs, the value |  
 16784 returned shall always be greater than or equal to one and less than or equal to the value of |  
 16785 `{NGROUPS_MAX}+1`.

16786 **RETURN VALUE**

16787 Upon successful completion, the number of supplementary group IDs shall be returned. A |  
 16788 return value of `-1` indicates failure and `errno` shall be set to indicate the error.

16789 **ERRORS**16790 The `getgroups()` function shall fail if:

16791 `[EINVAL]` The `gidsetsize` argument is non-zero and less than the number of group IDs |  
 16792 that would have been returned.

16793 **EXAMPLES**16794 **Getting the Supplementary Group IDs of the Calling Process**

16795 The following example places the current supplementary group IDs of the calling process into |  
 16796 the `group` array.

```
16797 #include <sys/types.h>
16798 #include <unistd.h>
16799 ...
16800 gid_t *group;
16801 int nogroups;
16802 long ngroups_max;

16803 ngroups_max = sysconf(_SC_NGROUPS_MAX) + 1;
16804 group = (gid_t *)malloc(ngroups_max * sizeof(gid_t));
16805 ngroups = getgroups(ngroups_max, group);
```

16806 **APPLICATION USAGE**

16807 None.

16808 **RATIONALE**

16809 The related function `setgroups()` is a privileged operation and therefore is not covered by this |  
 16810 volume of IEEE Std 1003.1-200x.

- 16811 As implied by the definition of supplementary groups, the effective group ID may appear in the  
16812 array returned by *getgroups()* or it may be returned only by *getegid()*. Duplication may exist, but  
16813 the application needs to call *getegid()* to be sure of getting all of the information. Various  
16814 implementation variations and administrative sequences cause the set of groups appearing in  
16815 the result of *getgroups()* to vary in order and as to whether the effective group ID is included,  
16816 even when the set of groups is the same (in the mathematical sense of “set”). (The history of a  
16817 process and its parents could affect the details of result.)
- 16818 Applications writers should note that {NGROUPS\_MAX} is not necessarily a constant on all  
16819 implementations.
- 16820 **FUTURE DIRECTIONS**
- 16821 None.
- 16822 **SEE ALSO**
- 16823 *getegid()*, *setgid()*, the Base Definitions volume of IEEE Std 1003.1-200x, <sys/types.h>,  
16824 <unistd.h>
- 16825 **CHANGE HISTORY**
- 16826 First released in Issue 3.
- 16827 Entry included for alignment with the POSIX.1-1988 standard.
- 16828 **Issue 5**
- 16829 Normative text previously in the APPLICATION USAGE section is moved to the  
16830 DESCRIPTION.
- 16831 **Issue 6**
- 16832 In the SYNOPSIS, the optional include of the <sys/types.h> header is removed.
- 16833 The following new requirements on POSIX implementations derive from alignment with the  
16834 Single UNIX Specification:
- 16835 • The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was  
16836 required for conforming implementations of previous POSIX specifications, it was not  
16837 required for UNIX applications.
  - 16838 • A return value of 0 is not permitted, because {NGROUPS\_MAX} cannot be 0. This is a FIPS  
16839 requirement.
- 16840 The following changes were made to align with the IEEE P1003.1a draft standard:
- 16841 • Explanation added that the effective group ID may be included in the supplementary group  
16842 list.

16843 **NAME**

16844 gethostbyaddr, gethostbyname — network host database functions

16845 **SYNOPSIS**

16846 #include &lt;netdb.h&gt;

16847 OB struct hostent \*gethostbyaddr(const void \*addr, socklen\_t len,  
16848 int type);

16849 struct hostent \*gethostbyname(const char \*name);

16850

16851 **DESCRIPTION**

16852 These functions shall retrieve information about hosts. This information is considered to be |  
 16853 stored in a database that can be accessed sequentially or randomly. Implementation of this |  
 16854 database is unspecified. |

16855 **Note:** In many cases it is implemented by the Domain Name System, as documented in RFC 1034,  
 16856 RFC 1035, and RFC 1886.

16857 Entries shall be returned in **hostent** structures. |

16858 The *gethostbyaddr()* function shall return an entry containing addresses of address family *type* for  
 16859 the host with address *addr*. The *len* argument contains the length of the address pointed to by |  
 16860 *addr*. The *gethostbyaddr()* function need not be reentrant. A function that is not required to be  
 16861 reentrant is not required to be thread-safe.

16862 The *gethostbyname()* function shall return an entry containing addresses of address family  
 16863 AF\_INET for the host with name *name*. The *gethostbyname()* function need not be reentrant. A  
 16864 function that is not required to be reentrant is not required to be thread-safe.

16865 The *addr* argument of *gethostbyaddr()* shall be an **in\_addr** structure when *type* is AF\_INET. It |  
 16866 contains a binary format (that is, not null-terminated) address in network byte order. The |  
 16867 *gethostbyaddr()* function is not guaranteed to return addresses of address families other than  
 16868 AF\_INET, even when such addresses exist in the database.

16869 If *gethostbyaddr()* returns successfully, then the *h\_addrtype* field in the result shall be the same as |  
 16870 the *type* argument that was passed to the function, and the *h\_addr\_list* field shall list a single  
 16871 address that is a copy of the *addr* argument that was passed to the function. |

16872 The *name* argument of *gethostbyname()* shall be a node name; the behavior of *gethostbyname()* |  
 16873 when passed a numeric address string is unspecified. For IPv4, a numeric address string shall be  
 16874 in the dotted-decimal notation described in *inet\_addr()*. |

16875 If *name* is not a numeric address string and is an alias for a valid host name, then *gethostbyname()* |  
 16876 shall return information about the host name to which the alias refers, and *name* shall be  
 16877 included in the list of aliases returned.

16878 **RETURN VALUE**

16879 Upon successful completion, these functions shall return a pointer to a **hostent** structure if the  
 16880 requested entry was found, and a null pointer if the end of the database was reached or the  
 16881 requested entry was not found.

16882 Upon unsuccessful completion, *gethostbyaddr()* and *gethostbyname()* shall set *h\_errno* to indicate  
 16883 the error.

16884 **ERRORS**

16885 These functions shall fail in the following cases. The *gethostbyaddr()* and *gethostbyname()*  
 16886 functions shall set *h\_errno* to the value shown in the list below. Any changes to *errno* are  
 16887 unspecified.

- 16888 [HOST\_NOT\_FOUND]  
16889 No such host is known.
- 16890 [NO\_DATA] The server recognized the request and the name, but no address is available.  
16891 Another type of request to the name server for the domain might return an  
16892 answer.
- 16893 [NO\_RECOVERY]  
16894 An unexpected server failure occurred which cannot be recovered.
- 16895 [TRY\_AGAIN] A temporary and possibly transient error occurred, such as a failure of a  
16896 server to respond.
- 16897 **EXAMPLES**  
16898 None.
- 16899 **APPLICATION USAGE**  
16900 The *gethostbyaddr()* and *gethostbyname()* functions may return pointers to static data, which may  
16901 be overwritten by subsequent calls to any of these functions.
- 16902 The *getaddrinfo()* and *getnameinfo()* functions are preferred over the *gethostbyaddr()* and  
16903 *gethostbyname()* functions.
- 16904 **RATIONALE**  
16905 None.
- 16906 **FUTURE DIRECTIONS**  
16907 The *gethostbyaddr()* and *gethostbyname()* functions may be withdrawn in a future version.
- 16908 **SEE ALSO**  
16909 *endhostent()*, *endservent()*, *gai\_strerror()*, *getaddrinfo()*, *h\_errno*, *inet\_addr()*, the Base Definitions  
16910 volume of IEEE Std 1003.1-200x, <**netdb.h**>
- 16911 **CHANGE HISTORY**  
16912 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

16913 **NAME**

16914           gethostbyname — network host database functions

16915 **SYNOPSIS**

16916           #include &lt;netdb.h&gt;

16917 OB        struct hostent \*gethostbyname(const char \*name);

16918

16919 **DESCRIPTION**16920           Refer to *gethostbyaddr()*.

16921 **NAME**

16922       gethostent — network host database functions

16923 **SYNOPSIS**

16924       #include <netdb.h>

16925       struct hostent \*gethostent(void);

16926 **DESCRIPTION**

16927       Refer to *endhostent()*.

16928 **NAME**

16929           gethostid — get an identifier for the current host

16930 **SYNOPSIS**

16931 XSI        #include <unistd.h>

16932            long gethostid(void);

16933

16934 **DESCRIPTION**

16935            The *gethostid()* function shall retrieve a 32-bit identifier for the current host. |

16936 **RETURN VALUE**

16937            Upon successful completion, *gethostid()* shall return an identifier for the current host.

16938 **ERRORS**

16939            No errors are defined.

16940 **EXAMPLES**

16941            None.

16942 **APPLICATION USAGE**

16943            This volume of IEEE Std 1003.1-200x does not define the domain in which the return value is  
16944            unique.

16945 **RATIONALE**

16946            None.

16947 **FUTURE DIRECTIONS**

16948            None.

16949 **SEE ALSO**

16950            *random()*, the Base Definitions volume of IEEE Std 1003.1-200x, <unistd.h>

16951 **CHANGE HISTORY**

16952            First released in Issue 4, Version 2.

16953 **Issue 5**

16954            Moved from X/OPEN UNIX extension to BASE.

16955 **NAME**

16956        gethostname — get name of current host

16957 **SYNOPSIS**

16958        #include &lt;unistd.h&gt;

16959        int gethostname(char \*name, size\_t namelen);

16960 **DESCRIPTION**

16961        The *gethostname()* function shall return the standard host name for the current machine. The  
16962        *namelen* argument shall specify the size of the array pointed to by the *name* argument. The  
16963        returned name shall be null-terminated, except that if *namelen* is an insufficient length to hold  
16964        the host name, then the returned name shall be truncated and it is unspecified whether the  
16965        returned name is null-terminated.

16966        Host names are limited to {HOST\_NAME\_MAX} bytes.

16967 **RETURN VALUE**

16968        Upon successful completion, 0 shall be returned; otherwise, -1 shall be returned.

16969 **ERRORS**

16970        No errors are defined.

16971 **EXAMPLES**

16972        None.

16973 **APPLICATION USAGE**

16974        None.

16975 **RATIONALE**

16976        None.

16977 **FUTURE DIRECTIONS**

16978        None.

16979 **SEE ALSO**16980        *gethostid()*, *uname()*, the Base Definitions volume of IEEE Std 1003.1-200x, <unistd.h>16981 **CHANGE HISTORY**

16982        First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

16983        The Open Group Base Resolution bwg2001-008 is applied, changing the *namelen* parameter from16984        **socklen\_t** to **size\_t**.

16985 **NAME**

16986 getitimer, setitimer — get and set value of interval timer

16987 **SYNOPSIS**

```
16988 XSI #include <sys/time.h>
16989
16989 int getitimer(int which, struct itimerval *value);
16990 int setitimer(int which, const struct itimerval *restrict value,
16991              struct itimerval *restrict ovalue);
16992
```

16993 **DESCRIPTION**

16994 The *getitimer()* function shall store the current value of the timer specified by *which* into the  
 16995 structure pointed to by *value*. The *setitimer()* function shall set the timer specified by *which* to  
 16996 the value specified in the structure pointed to by *value*, and if *ovalue* is not a null pointer, stores  
 16997 the previous value of the timer in the structure pointed to by *ovalue*.

16998 A timer value is defined by the **itimerval** structure, specified in `<sys/time.h>`. If *it\_value* is non-  
 16999 zero, it shall indicate the time to the next timer expiration. If *it\_interval* is non-zero, it shall  
 17000 specify a value to be used in reloading *it\_value* when the timer expires. Setting *it\_value* to 0 shall  
 17001 disable a timer, regardless of the value of *it\_interval*. Setting *it\_interval* to 0 shall disable a timer  
 17002 after its next expiration (assuming *it\_value* is non-zero).

17003 Implementations may place limitations on the granularity of timer values. For each interval  
 17004 timer, if the requested timer value requires a finer granularity than the implementation supports,  
 17005 the actual timer value shall be rounded up to the next supported value.

17006 An XSI-conforming implementation provides each process with at least three interval timers,  
 17007 which are indicated by the *which* argument:

17008	ITIMER_REAL	Decrements in real time. A SIGALRM signal is delivered when this timer expires.
17009		
17010	ITIMER_VIRTUAL	Decrements in process virtual time. It runs only when the process is executing. A SIGVTALRM signal is delivered when it expires.
17011		
17012	ITIMER_PROF	Decrements both in process virtual time and when the system is running on behalf of the process. It is designed to be used by interpreters in statistically profiling the execution of interpreted programs. Each time the ITIMER_PROF timer expires, the SIGPROF signal is delivered.
17013		
17014		
17015		

17016 The interaction between *setitimer()* and any of *alarm()*, *sleep()*, or *usleep()* is unspecified.

17017 **RETURN VALUE**

17018 Upon successful completion, *getitimer()* or *setitimer()* shall return 0; otherwise, `-1` shall be  
 17019 returned and *errno* set to indicate the error.

17020 **ERRORS**

17021 The *setitimer()* function shall fail if:

17022	[EINVAL]	The <i>value</i> argument is not in canonical form. (In canonical form, the number of microseconds is a non-negative integer less than 1,000,000 and the number of seconds is a non-negative integer.)
17023		
17024		

17025 The *getitimer()* and *setitimer()* functions may fail if:

17026	[EINVAL]	The <i>which</i> argument is not recognized.
-------	----------	--

17027 **EXAMPLES**

17028           None.

17029 **APPLICATION USAGE**

17030           None.

17031 **RATIONALE**

17032           None.

17033 **FUTURE DIRECTIONS**

17034           None.

17035 **SEE ALSO**

17036           *alarm()*, *sleep()*, *timer\_getoverrun()*, *ualarm()*, *usleep()*, the Base Definitions volume of  
17037           IEEE Std 1003.1-200x, <**signal.h**>, <**sys/time.h**>

17038 **CHANGE HISTORY**

17039           First released in Issue 4, Version 2.

17040 **Issue 5**

17041           Moved from X/OPEN UNIX extension to BASE.

17042 **Issue 6**

17043           The **restrict** keyword is added to the *setitimer()* prototype for alignment with the  
17044           ISO/IEC 9899:1999 standard.

17045 **NAME**

17046 getlogin, getlogin\_r — get login name

17047 **SYNOPSIS**

17048 #include &lt;unistd.h&gt;

17049 char \*getlogin(void);

17050 TSF int getlogin\_r(char \*name, size\_t namesize);

17051

17052 **DESCRIPTION**

17053 The *getlogin()* function shall return a pointer to a string containing the user name associated by  
 17054 the login activity with the controlling terminal of the current process. If *getlogin()* returns a non-  
 17055 null pointer, then that pointer points to the name that the user logged in under, even if there are  
 17056 several login names with the same user ID.

17057 The *getlogin()* function need not be reentrant. A function that is not required to be reentrant is  
 17058 not required to be thread-safe.

17059 TSF The *getlogin\_r()* function shall put the name associated by the login activity with the controlling  
 17060 terminal of the current process in the character array pointed to by *name*. The array is *namesize*  
 17061 characters long and should have space for the name and the terminating null character. The  
 17062 maximum size of the login name is {LOGIN\_NAME\_MAX}.

17063 If *getlogin\_r()* is successful, *name* points to the name the user used at login, even if there are  
 17064 several login names with the same user ID.

17065 **RETURN VALUE**

17066 Upon successful completion, *getlogin()* shall return a pointer to the login name or a null pointer  
 17067 if the user's login name cannot be found. Otherwise, it shall return a null pointer and set *errno* to  
 17068 indicate the error.

17069 The return value from *getlogin()* may point to static data whose content is overwritten by each  
 17070 call.

17071 TSF If successful, the *getlogin\_r()* function shall return zero; otherwise, an error number shall be  
 17072 returned to indicate the error.

17073 **ERRORS**17074 The *getlogin()* and *getlogin\_r()* functions may fail if:

17075 [EMFILE] {OPEN\_MAX} file descriptors are currently open in the calling process.

17076 [ENFILE] The maximum allowable number of files is currently open in the system.

17077 [ENXIO] The calling process has no controlling terminal.

17078 The *getlogin\_r()* function may fail if:

17079 TSF [ERANGE] The value of *namesize* is smaller than the length of the string to be returned  
 17080 including the terminating null character.

## 17081 EXAMPLES

17082 **Getting the User Login Name**

17083 The following example calls the *getlogin()* function to obtain the name of the user associated  
 17084 with the calling process, and passes this information to the *getpwnam()* function to get the  
 17085 associated user database information.

```

17086 #include <unistd.h>
17087 #include <sys/types.h>
17088 #include <pwd.h>
17089 #include <stdio.h>
17090 ...
17091 char *lgn;
17092 struct passwd *pw;
17093 ...
17094 if ((lgn = getlogin()) == NULL || (pw = getpwnam(lgn)) == NULL) {
17095     fprintf(stderr, "Get of user information failed.\n"); exit(1);
17096 }
```

17097 **APPLICATION USAGE**

17098 Three names associated with the current process can be determined: *getpwuid(geteuid())* shall  
 17099 return the name associated with the effective user ID of the process; *getlogin()* shall return the  
 17100 name associated with the current login activity; and *getpwuid(getuid())* shall return the name  
 17101 associated with the real user ID of the process.

17102 The *getlogin\_r()* function is thread-safe and returns values in a user-supplied buffer instead of  
 17103 possibly using a static data area that may be overwritten by each call.

17104 **RATIONALE**

17105 The *getlogin()* function returns a pointer to the user's login name. The same user ID may be  
 17106 shared by several login names. If it is desired to get the user database entry that is used during  
 17107 login, the result of *getlogin()* should be used to provide the argument to the *getpwnam()*  
 17108 function. (This might be used to determine the user's login shell, particularly where a single user  
 17109 has multiple login shells with distinct login names, but the same user ID.)

17110 The information provided by the *cuserid()* function, which was originally defined in the  
 17111 POSIX.1-1988 standard and subsequently removed, can be obtained by the following:

```
17112 getpwuid(geteuid())
```

17113 while the information provided by historical implementations of *cuserid()* can be obtained by:

```
17114 getpwuid(getuid())
```

17115 The thread-safe version of this function places the user name in a user-supplied buffer and  
 17116 returns a non-zero value if it fails. The non-thread-safe version may return the name in a static  
 17117 data area that may be overwritten by each call.

17118 **FUTURE DIRECTIONS**

17119 None.

17120 **SEE ALSO**

17121 *getpwnam()*, *getpwuid()*, *geteuid()*, *getuid()*, the Base Definitions volume of IEEE Std 1003.1-200x,  
 17122 <limits.h>, <unistd.h>

17123 **CHANGE HISTORY**

17124 First released in Issue 1. Derived from System V Release 2.0.

17125 **Issue 5**

17126 Normative text previously in the APPLICATION USAGE section is moved to the RETURN  
17127 VALUE section.

17128 The *getlogin\_r()* function is included for alignment with the POSIX Threads Extension.

17129 A note indicating that the *getlogin()* function need not be reentrant is added to the  
17130 DESCRIPTION.

17131 **Issue 6**

17132 The *getlogin\_r()* function is marked as part of the Thread-Safe Functions option.

17133 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

17134 The following new requirements on POSIX implementations derive from alignment with the  
17135 Single UNIX Specification:

17136 • In the RETURN VALUE section, the requirement to set *errno* on error is added.

17137 • The [EMFILE], [ENFILE], and [ENXIO] optional error conditions are added.

17138 The APPLICATION USAGE section is updated to include a note on the thread-safe function and  
17139 its avoidance of possibly using a static data area.

## 17140 NAME

17141 getmsg, getpmsg — receive next message from a STREAMS file (STREAMS)

## 17142 SYNOPSIS

17143 XSR #include &lt;stropts.h&gt;

```

17144 int getmsg(int fildes, struct strbuf *restrict ctlptr,
17145           struct strbuf *restrict dataptr, int *restrict flagsp);
17146 int getpmsg(int fildes, struct strbuf *restrict ctlptr,
17147            struct strbuf *restrict dataptr, int *restrict bandp,
17148            int *restrict flagsp);
17149

```

## 17150 DESCRIPTION

17151 The *getmsg()* function shall retrieve the contents of a message located at the head of the  
 17152 STREAM head read queue associated with a STREAMS file and place the contents into one or  
 17153 more buffers. The message contains either a data part, a control part, or both. The data and  
 17154 control parts of the message shall be placed into separate buffers, as described below. The  
 17155 semantics of each part are defined by the originator of the message.

17156 The *getpmsg()* function shall be equivalent to *getmsg()*, except that it provides finer control over  
 17157 the priority of the messages received. Except where noted, all requirements on *getmsg()* also  
 17158 pertain to *getpmsg()*.

17159 The *fildes* argument specifies a file descriptor referencing a STREAMS-based file.

17160 The *ctlptr* and *dataptr* arguments each point to a **strbuf** structure, in which the *buf* member points  
 17161 to a buffer in which the data or control information is to be placed, and the *maxlen* member  
 17162 indicates the maximum number of bytes this buffer can hold. On return, the *len* member shall  
 17163 contain the number of bytes of data or control information actually received. The *len* member  
 17164 shall be set to 0 if there is a zero-length control or data part and *len* shall be set to -1 if no data or  
 17165 control information is present in the message.

17166 When *getmsg()* is called, *flagsp* should point to an integer that indicates the type of message the  
 17167 process is able to receive. This is described further below.

17168 The *ctlptr* argument is used to hold the control part of the message, and *dataptr* is used to hold  
 17169 the data part of the message. If *ctlptr* (or *dataptr*) is a null pointer or the *maxlen* member is -1, the  
 17170 control (or data) part of the message shall not be processed and shall be left on the STREAM  
 17171 head read queue, and if the *ctlptr* (or *dataptr*) is not a null pointer, *len* shall be set to -1. If the  
 17172 *maxlen* member is set to 0 and there is a zero-length control (or data) part, that zero-length part  
 17173 shall be removed from the read queue and *len* shall be set to 0. If the *maxlen* member is set to 0  
 17174 and there are more than 0 bytes of control (or data) information, that information shall be left on  
 17175 the read queue and *len* shall be set to 0. If the *maxlen* member in *ctlptr* (or *dataptr*) is less than the  
 17176 control (or data) part of the message, *maxlen* bytes shall be retrieved. In this case, the remainder  
 17177 of the message shall be left on the STREAM head read queue and a non-zero return value shall  
 17178 be provided.

17179 By default, *getmsg()* shall process the first available message on the STREAM head read queue.  
 17180 However, a process may choose to retrieve only high-priority messages by setting the integer  
 17181 pointed to by *flagsp* to RS\_HIPRI. In this case, *getmsg()* shall only process the next message if it is  
 17182 a high-priority message. When the integer pointed to by *flagsp* is 0, any available message shall  
 17183 be retrieved. In this case, on return, the integer pointed to by *flagsp* shall be set to RS\_HIPRI if a  
 17184 high-priority message was retrieved, or 0 otherwise.

17185 For *getpmsg()*, the flags are different. The *flagsp* argument points to a bitmask with the following  
 17186 mutually-exclusive flags defined: MSG\_HIPRI, MSG\_BAND, and MSG\_ANY. Like *getmsg()*,

17187 *getpmsg()* shall process the first available message on the STREAM head read queue. A process |  
 17188 may choose to retrieve only high-priority messages by setting the integer pointed to by *flagsp* to |  
 17189 MSG\_HIPRI and the integer pointed to by *bandp* to 0. In this case, *getpmsg()* shall only process |  
 17190 the next message if it is a high-priority message. In a similar manner, a process may choose to |  
 17191 retrieve a message from a particular priority band by setting the integer pointed to by *flagsp* to |  
 17192 MSG\_BAND and the integer pointed to by *bandp* to the priority band of interest. In this case, |  
 17193 *getpmsg()* shall only process the next message if it is in a priority band equal to, or greater than, |  
 17194 the integer pointed to by *bandp*, or if it is a high-priority message. If a process wants to get the |  
 17195 first message off the queue, the integer pointed to by *flagsp* should be set to MSG\_ANY and the |  
 17196 integer pointed to by *bandp* should be set to 0. On return, if the message retrieved was a high- |  
 17197 priority message, the integer pointed to by *flagsp* shall be set to MSG\_HIPRI and the integer |  
 17198 pointed to by *bandp* shall be set to 0. Otherwise, the integer pointed to by *flagsp* shall be set to |  
 17199 MSG\_BAND and the integer pointed to by *bandp* shall be set to the priority band of the message.

17200 If O\_NONBLOCK is not set, *getmsg()* and *getpmsg()* shall block until a message of the type |  
 17201 specified by *flagsp* is available at the front of the STREAM head read queue. If O\_NONBLOCK is |  
 17202 set and a message of the specified type is not present at the front of the read queue, *getmsg()* and |  
 17203 *getpmsg()* shall fail and set *errno* to [EAGAIN].

17204 If a hangup occurs on the STREAM from which messages are retrieved, *getmsg()* and *getpmsg()* |  
 17205 shall continue to operate normally, as described above, until the STREAM head read queue is |  
 17206 empty. Thereafter, they shall return 0 in the *len* members of *ctlptr* and *dataptr*.

#### 17207 RETURN VALUE

17208 Upon successful completion, *getmsg()* and *getpmsg()* shall return a non-negative value. A value |  
 17209 of 0 indicates that a full message was read successfully. A return value of MORECTL indicates |  
 17210 that more control information is waiting for retrieval. A return value of MOREDATA indicates |  
 17211 that more data is waiting for retrieval. A return value of the bitwise-logical OR of MORECTL |  
 17212 and MOREDATA indicates that both types of information remain. Subsequent *getmsg()* and |  
 17213 *getpmsg()* calls shall retrieve the remainder of the message. However, if a message of higher |  
 17214 priority has come in on the STREAM head read queue, the next call to *getmsg()* or *getpmsg()* |  
 17215 shall retrieve that higher-priority message before retrieving the remainder of the previous |  
 17216 message.

17217 If the high priority control part of the message is consumed, the message shall be placed back on |  
 17218 the queue as a normal message of band 0. Subsequent *getmsg()* and *getpmsg()* calls shall retrieve |  
 17219 the remainder of the message. If, however, a priority message arrives or already exists on the |  
 17220 STREAM head, the subsequent call to *getmsg()* or *getpmsg()* shall retrieve the higher-priority |  
 17221 message before retrieving the remainder of the message that was put back.

17222 Upon failure, *getmsg()* and *getpmsg()* shall return -1 and set *errno* to indicate the error.

#### 17223 ERRORS

17224 The *getmsg()* and *getpmsg()* functions shall fail if:

- |       |           |  |
|-------|-----------|--|
| 17225 | [EAGAIN]  | The O_NONBLOCK flag is set and no messages are available.  |
| 17226 | [EBADF]   | The <i>fildes</i> argument is not a valid file descriptor open for reading.  |
| 17227 | [EBADMSG] | The queued message to be read is not valid for <i>getmsg()</i> or <i>getpmsg()</i> or a pending file descriptor is at the STREAM head.                                       |
| 17228 |           |  |
| 17229 | [EINTR]   | A signal was caught during <i>getmsg()</i> or <i>getpmsg()</i> .   |
| 17230 | [EINVAL]  | An illegal value was specified by <i>flagsp</i> , or the STREAM or multiplexer referenced by <i>fildes</i> is linked (directly or indirectly) downstream from a multiplexer. |
| 17231 |           |  |
| 17232 |           |  |

17233 [ENOSTR] A STREAM is not associated with *fildev*.

17234 In addition, *getmsg()* and *getpmsg()* shall fail if the STREAM head had processed an  
17235 asynchronous error before the call. In this case, the value of *errno* does not reflect the result of  
17236 *getmsg()* or *getpmsg()* but reflects the prior error.

#### 17237 EXAMPLES

##### 17238 Getting Any Message

17239 In the following example, the value of *fd* is assumed to refer to an open STREAMS file. The call  
17240 to *getmsg()* retrieves any available message on the associated STREAM-head read queue,  
17241 returning control and data information to the buffers pointed to by *ctrlbuf* and *databuf*,  
17242 respectively.

```
17243 #include <stropts.h>
17244 ...
17245 int fd;
17246 char ctrlbuf[128];
17247 char databuf[512];
17248 struct strbuf ctrl;
17249 struct strbuf data;
17250 int flags = 0;
17251 int ret;

17252 ctrl.buf = ctrlbuf;
17253 ctrl.maxlen = sizeof(ctrlbuf);

17254 data.buf = databuf;
17255 data.maxlen = sizeof(databuf);

17256 ret = getmsg (fd, &ctrl, &data, &flags);
```

##### 17257 Getting the First Message off the Queue

17258 In the following example, the call to *getpmsg()* retrieves the first available message on the  
17259 associated STREAM-head read queue.

```
17260 #include <stropts.h>
17261 ...
17262 int fd;
17263 char ctrlbuf[128];
17264 char databuf[512];
17265 struct strbuf ctrl;
17266 struct strbuf data;
17267 int band = 0;
17268 int flags = MSG_ANY;
17269 int ret;

17270 ctrl.buf = ctrlbuf;
17271 ctrl.maxlen = sizeof(ctrlbuf);

17272 data.buf = databuf;
17273 data.maxlen = sizeof(databuf);

17274 ret = getpmsg (fd, &ctrl, &data, &band, &flags);
```

17275 **APPLICATION USAGE**

17276 None.

17277 **RATIONALE**

17278 None.

17279 **FUTURE DIRECTIONS**

17280 None.

17281 **SEE ALSO**

17282 *poll()*, *putmsg()*, *read()*, *write()*, the Base Definitions volume of IEEE Std 1003.1-200x,  
17283 **<stropts.h>**, Section 2.6 (on page 488)

17284 **CHANGE HISTORY**

17285 First released in Issue 4, Version 2.

17286 **Issue 5**

17287 Moved from X/OPEN UNIX extension to BASE.

17288 A paragraph regarding “high-priority control parts of messages” is added to the RETURN  
17289 VALUE section.

17290 **Issue 6**

17291 This function is marked as part of the XSI STREAMS Option Group.

17292 The **restrict** keyword is added to the *getmsg()* and *getpmsg()* prototypes for alignment with the  
17293 ISO/IEC 9899:1999 standard.

## 17294 NAME

17295 getnameinfo — get name information |

## 17296 SYNOPSIS

17297 #include &lt;sys/socket.h&gt;

17298 #include &lt;netdb.h&gt;

```
17299 int getnameinfo(const struct sockaddr *restrict sa, socklen_t salen,
17300 char *restrict node, socklen_t nodelen, char *restrict service,
17301 socklen_t servicelen, unsigned flags);
```

## 17302 DESCRIPTION

17303 The *getnameinfo()* function shall translate a socket address to a node name and service location, |  
 17304 all of which are defined as in *getaddrinfo()*.

17305 The *sa* argument points to a socket address structure to be translated.

17306 If the *node* argument is non-NULL and the *nodelen* argument is non-zero, then the *node* argument  
 17307 points to a buffer able to contain up to *nodelen* characters that receives the node name as a null-  
 17308 terminated string. If the *node* argument is NULL or the *nodelen* argument is zero, the node name  
 17309 shall not be returned. If the node's name cannot be located, the numeric form of the node's  
 17310 address is returned instead of its name.

17311 If the *service* argument is non-NULL and the *servicelen* argument is non-zero, then the *service*  
 17312 argument points to a buffer able to contain up to *servicelen* bytes that receives the service name |  
 17313 as a null-terminated string. If the *service* argument is NULL or the *servicelen* argument is zero, |  
 17314 the service name shall not be returned. If the service's name cannot be located, the numeric form |  
 17315 of the service address (for example, its port number) shall be returned instead of its name. |

17316 The *flags* argument is a flag that changes the default actions of the function. By default the fully- |  
 17317 qualified domain name (FQDN) for the host shall be returned, but: |

- 17318 • If the flag bit NI\_NOFQDN is set, only the node name portion of the FQDN shall be returned  
 17319 for local hosts.
- 17320 • If the flag bit NI\_NUMERICHOST is set, the numeric form of the host's address shall be  
 17321 returned instead of its name, under all circumstances.
- 17322 • If the flag bit NI\_NAMEREQD is set, an error shall be returned if the host's name cannot be  
 17323 located.
- 17324 • If the flag bit NI\_NUMERICSERV is set, the numeric form of the service address shall be  
 17325 returned (for example, its port number) instead of its name, under all circumstances.
- 17326 • If the flag bit NI\_DGRAM is set, this indicates that the service is a datagram service  
 17327 (SOCK\_DGRAM). The default behavior shall assume that the service is a stream service  
 17328 (SOCK\_STREAM).

## 17329 Notes:

- 17330 1. The two NI\_NUMERICxxx flags are required to support the *-n* flag that many  
 17331 commands provide.
- 17332 2. The NI\_DGRAM flag is required for the few AF\_INET and AF\_INET6 port numbers (for |  
 17333 example, [512,514]) that represent different services for UDP and TCP. |

17334 The *getnameinfo()* function shall be thread-safe.

17335 **RETURN VALUE**

17336 A zero return value for *getnameinfo()* indicates successful completion; a non-zero return value  
 17337 indicates failure. The possible values for the failures are listed in the ERRORS section.

17338 Upon successful completion, *getnameinfo()* shall return the *node* and *service* names, if requested,  
 17339 in the buffers provided. The returned names are always null-terminated strings. |

17340 **ERRORS**

17341 The *getnameinfo()* function shall fail and return the corresponding value if:

17342 [EAI\_AGAIN] The name could not be resolved at this time. Future attempts may succeed.

17343 [EAI\_BADFLAGS]

17344 The *flags* had an invalid value.

17345 [EAI\_FAIL] A non-recoverable error occurred.

17346 [EAI\_FAMILY] The address family was not recognized or the address length was invalid for  
 17347 the specified family.

17348 [EAI\_MEMORY] There was a memory allocation failure.

17349 [EAI\_NONAME] The name does not resolve for the supplied parameters.

17350 NI\_NAMEREQD is set and the host's name cannot be located, or both  
 17351 *nodename* and *servname* were null.

17352 [EAI\_SYSTEM] A system error occurred. The error code can be found in *errno*.

17353 **EXAMPLES**

17354 None.

17355 **APPLICATION USAGE**

17356 If the returned values are to be used as part of any further name resolution (for example, passed  
 17357 to *getaddrinfo()*), applications should provide buffers large enough to store any result possible  
 17358 on the system. |

17359 **RATIONALE**

17360 None.

17361 **FUTURE DIRECTIONS**

17362 None.

17363 **SEE ALSO**

17364 *gai\_strerror()*, *getaddrinfo()*, *getservbyname()*, *getservbyport()*, *inet\_ntop()*, *socket()*, the Base  
 17365 Definitions volume of IEEE Std 1003.1-200x, <netdb.h>, <sys/socket.h>

17366 **CHANGE HISTORY**

17367 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

17368 The **restrict** keyword is added to the *getnameinfo()* prototype for alignment with the  
 17369 ISO/IEC 9899:1999 standard.

17370 **NAME**

17371       getnetbyaddr — network database functions

17372 **SYNOPSIS**

17373       #include <netdb.h>

17374       struct netent \*getnetbyaddr(uint32\_t net, int type);

17375 **DESCRIPTION**

17376       Refer to *endnetent()*.

17377 **NAME**

17378           getnetbyname — network database functions

17379 **SYNOPSIS**

17380           #include <netdb.h>

17381           struct netent \*getnetbyname(const char \*name);

17382 **DESCRIPTION**

17383           Refer to *endnetent()*.

17384 **NAME**

17385       getnetent — network database functions

17386 **SYNOPSIS**

17387       #include <netdb.h>

17388       struct netent \*getnetent(void);

17389 **DESCRIPTION**

17390       Refer to *endnetent()*.

17391 **NAME**

17392            getopt, optarg, opterr, optind, optopt — command option parsing

17393 **SYNOPSIS**

17394            #include &lt;unistd.h&gt;

17395            int getopt(int argc, char \* const argv[], const char \*optstring);

17396            extern char \*optarg;

17397            extern int optind, opterr, optopt;

17398 **DESCRIPTION**

17399            The *getopt()* function is a command-line parser that shall follow Utility Syntax Guidelines 3, 4, 5, |  
 17400            6, 7, 9, and 10 in the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2, Utility Syntax |  
 17401            Guidelines. |

17402            The parameters *argc* and *argv* are the argument count and argument array as passed to *main()* |  
 17403            (see *exec*). The argument *optstring* is a string of recognized option characters; if a character is |  
 17404            followed by a colon, the option takes an argument. All option characters allowed by Utility |  
 17405            Syntax Guideline 3 are allowed in *optstring*. The implementation may accept other characters as |  
 17406            an extension.

17407            The variable *optind* is the index of the next element of the *argv[]* vector to be processed. It shall |  
 17408            be initialized to 1 by the system, and *getopt()* shall update it when it finishes with each element |  
 17409            of *argv[]*. When an element of *argv[]* contains multiple option characters, it is unspecified how |  
 17410            *getopt()* determines which options have already been processed.

17411            The *getopt()* function shall return the next option character (if one is found) from *argv* that |  
 17412            matches a character in *optstring*, if there is one that matches. If the option takes an argument, |  
 17413            *getopt()* shall set the variable *optarg* to point to the option-argument as follows:

17414            1. If the option was the last character in the string pointed to by an element of *argv*, then |  
 17415            *optarg* shall contain the next element of *argv*, and *optind* shall be incremented by 2. If the |  
 17416            resulting value of *optind* is greater than *argc*, this indicates a missing option-argument, and |  
 17417            *getopt()* shall return an error indication.

17418            2. Otherwise, *optarg* shall point to the string following the option character in that element of |  
 17419            *argv*, and *optind* shall be incremented by 1. |

17420            If, when *getopt()* is called:

17421            *argv[optind]* is a null pointer

17422            \**argv[optind]* is not the character -

17423            *argv[optind]* points to the string "--"

17424            *getopt()* shall return -1 without changing *optind*. If:

17425            *argv[optind]* points to the string "--"

17426            *getopt()* shall return -1 after incrementing *optind*.

17427            If *getopt()* encounters an option character that is not contained in *optstring*, it shall return the |  
 17428            question-mark ('?') character. If it detects a missing option-argument, it shall return the colon |  
 17429            character (':') if the first character of *optstring* was a colon, or a question-mark character ('?') |  
 17430            otherwise. In either case, *getopt()* shall set the variable *optopt* to the option character that caused |  
 17431            the error. If the application has not set the variable *opterr* to 0 and the first character of *optstring* |  
 17432            is not a colon, *getopt()* shall also print a diagnostic message to *stderr* in the format specified for |  
 17433            the *getopts* utility.

17434            The *getopt()* function need not be reentrant. A function that is not required to be reentrant is not |  
 17435            required to be thread-safe.

17436 **RETURN VALUE**

17437 The *getopt()* function shall return the next option character specified on the command line.

17438 A colon (':') shall be returned if *getopt()* detects a missing argument and the first character of  
17439 *optstring* was a colon (':').

17440 A question mark ('?') shall be returned if *getopt()* encounters an option character not in  
17441 *optstring* or detects a missing argument and the first character of *optstring* was not a colon (':').

17442 Otherwise, *getopt()* shall return -1 when all command line options are parsed.

17443 **ERRORS**

17444 No errors are defined.

17445 **EXAMPLES**17446 **Parsing Command Line Options**

17447 The following code fragment shows how you might process the arguments for a utility that can  
17448 take the mutually-exclusive options *a* and *b* and the options *f* and *o*, both of which require  
17449 arguments:

```
17450 #include <unistd.h>
17451 int
17452 main(int argc, char *argv[ ])
17453 {
17454     int c;
17455     int bflg, aflag, errflag;
17456     char *ifile;
17457     char *ofile;
17458     extern char *optarg;
17459     extern int optind, optopt;
17460     . . .
17461     while ((c = getopt(argc, argv, ":abf:o:")) != -1) {
17462         switch(c) {
17463             case 'a':
17464                 if (bflg)
17465                     errflag++;
17466                 else
17467                     aflag++;
17468                 break;
17469             case 'b':
17470                 if (aflag)
17471                     errflag++;
17472                 else {
17473                     bflg++;
17474                     bproc();
17475                 }
17476                 break;
17477             case 'f':
17478                 ifile = optarg;
17479                 break;
17480             case 'o':
17481                 ofile = optarg;
17482                 break;
```

```

17483         case ':':          /* -f or -o without operand */
17484             fprintf(stderr,
17485                 "Option -%c requires an operand\n", optopt);
17486             errflg++;
17487             break;
17488         case '?':
17489             fprintf(stderr,
17490                 "Unrecognized option: -%c\n", optopt);
17491             errflg++;
17492     }
17493 }
17494 if (errflg) {
17495     fprintf(stderr, "usage: . . . ");
17496     exit(2);
17497 }
17498 for ( ; optind < argc; optind++) {
17499     if (access(argv[optind], R_OK)) {
17500         . . .
17501     }

```

17502 **This code accepts any of the following as equivalent:**

```

17503 cmd -ao arg path path
17504 cmd -a -o arg path path
17505 cmd -o arg -a path path
17506 cmd -a -o arg -- path path
17507 cmd -a -oarg path path
17508 cmd -aoarg path path

```

### 17509 **Checking Options and Arguments**

17510 The following example parses a set of command line options and prints messages to standard  
17511 output for each option and argument that it encounters.

```

17512 #include <unistd.h>
17513 #include <stdio.h>
17514 ...
17515 int c;
17516 char *filename;
17517 extern char *optarg;
17518 extern int optind, optopt, opterr;
17519 ...
17520 while ((c = getopt(argc, argv, ":abf:")) != -1) {
17521     switch(c) {
17522     case 'a':
17523         printf("a is set\n");
17524         break;
17525     case 'b':
17526         printf("b is set\n");
17527         break;
17528     case 'f':
17529         filename = optarg;
17530         printf("filename is %s\n", filename);
17531         break;

```

```

17532         case ':' :
17533             printf("--%c without filename\n", optopt);
17534             break;
17535         case '?':
17536             printf("unknown arg %c\n", optopt);
17537             break;
17538     }
17539 }

```

#### 17540 **Selecting Options from the Command Line**

17541 The following example selects the type of database routines the user wants to use based on the  
 17542 *Options* argument.

```

17543 #include <unistd.h>
17544 #include <string.h>
17545 ...
17546 char *Options = "hdbt1";
17547 ...
17548 int dbtype, i;
17549 char c;
17550 char *st;
17551 ...
17552 dbtype = 0;
17553 while ((c = getopt(argc, argv, Options)) != -1) {
17554     if ((st = strchr(Options, c)) != NULL) {
17555         dbtype = st - Options;
17556         break;
17557     }
17558 }

```

#### 17559 **APPLICATION USAGE**

17560 The *getopt()* function is only required to support option characters included in Utility Syntax  
 17561 Guideline 3. Many historical implementations of *getopt()* support other characters as options.  
 17562 This is an allowed extension, but applications that use extensions are not maximally portable.  
 17563 Note that support for multi-byte option characters is only possible when such characters can be  
 17564 represented as type **int**.

#### 17565 **RATIONALE**

17566 The *optopt* variable represents historical practice and allows the application to obtain the identity  
 17567 of the invalid option.

17568 The description has been written to make it clear that *getopt()*, like the *getopts* utility, deals with  
 17569 option-arguments whether separated from the option by <blank>s or not. Note that the  
 17570 requirements on *getopt()* and *getopts* are more stringent than the Utility Syntax Guidelines.

17571 The *getopt()* function shall return  $-1$ , rather than EOF, so that <**stdio.h**> is not required.

17572 The special significance of a colon as the first character of *optstring* makes *getopt()* consistent  
 17573 with the *getopts* utility. It allows an application to make a distinction between a missing  
 17574 argument and an incorrect option letter without having to examine the option letter. It is true  
 17575 that a missing argument can only be detected in one case, but that is a case that has to be  
 17576 considered.

17577 **FUTURE DIRECTIONS**

17578 None.

17579 **SEE ALSO**17580 *exec*, the Base Definitions volume of IEEE Std 1003.1-200x, <**unistd.h**>, the Shell and Utilities  
17581 volume of IEEE Std 1003.1-200x17582 **CHANGE HISTORY**

17583 First released in Issue 1. Derived from Issue 1 of the SVID.

17584 **Issue 5**17585 A note indicating that the *getopt()* function need not be reentrant is added to the DESCRIPTION.17586 **Issue 6**

17587 IEEE PASC Interpretation 1003.2 #150 is applied.

17588 **NAME**

17589 getpeername — get the name of the peer socket

17590 **SYNOPSIS**

17591 #include <sys/socket.h>

17592 int getpeername(int *socket*, struct sockaddr \*restrict *address*,  
17593 socklen\_t \*restrict *address\_len*);

17594 **DESCRIPTION**

17595 The *getpeername()* function shall retrieve the peer address of the specified socket, store this  
17596 address in the **sockaddr** structure pointed to by the *address* argument, and store the length of this  
17597 address in the object pointed to by the *address\_len* argument.

17598 If the actual length of the address is greater than the length of the supplied **sockaddr** structure,  
17599 the stored address shall be truncated.

17600 If the protocol permits connections by unbound clients, and the peer is not bound, then the value  
17601 stored in the object pointed to by *address* is unspecified.

17602 **RETURN VALUE**

17603 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to  
17604 indicate the error.

17605 **ERRORS**

17606 The *getpeername()* function shall fail if:

17607 [EBADF] The *socket* argument is not a valid file descriptor.

17608 [EINVAL] The socket has been shut down.

17609 [ENOTCONN] The socket is not connected or otherwise has not had the peer prespecified.

17610 [ENOTSOCK] The *socket* argument does not refer to a socket.

17611 [EOPNOTSUPP] The operation is not supported for the socket protocol.

17612 The *getpeername()* function may fail if:

17613 [ENOBUFS] Insufficient resources were available in the system to complete the call.

17614 **EXAMPLES**

17615 None.

17616 **APPLICATION USAGE**

17617 None.

17618 **RATIONALE**

17619 None.

17620 **FUTURE DIRECTIONS**

17621 None.

17622 **SEE ALSO**

17623 *accept()*, *bind()*, *getsockname()*, *socket()*, the Base Definitions volume of IEEE Std 1003.1-200x,  
17624 <sys/socket.h>

17625 **CHANGE HISTORY**

17626 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

17627 The **restrict** keyword is added to the *getpeername()* prototype for alignment with the  
17628 ISO/IEC 9899:1999 standard.

17629 **NAME**

17630 `getpgid` — get the process group ID for a process

17631 **SYNOPSIS**

17632 XSI `#include <unistd.h>`

17633 `pid_t getpgid(pid_t pid);`

17634

17635 **DESCRIPTION**

17636 The `getpgid()` function shall return the process group ID of the process whose process ID is equal  
17637 to `pid`. If `pid` is equal to 0, `getpgid()` shall return the process group ID of the calling process.

17638 **RETURN VALUE**

17639 Upon successful completion, `getpgid()` shall return a process group ID. Otherwise, it shall return  
17640 `(pid_t)-1` and set `errno` to indicate the error.

17641 **ERRORS**

17642 The `getpgid()` function shall fail if:

17643 [EPERM] The process whose process ID is equal to `pid` is not in the same session as the  
17644 calling process, and the implementation does not allow access to the process  
17645 group ID of that process from the calling process.

17646 [ESRCH] There is no process with a process ID equal to `pid`.

17647 The `getpgid()` function may fail if:

17648 [EINVAL] The value of the `pid` argument is invalid.

17649 **EXAMPLES**

17650 None.

17651 **APPLICATION USAGE**

17652 None.

17653 **RATIONALE**

17654 None.

17655 **FUTURE DIRECTIONS**

17656 None.

17657 **SEE ALSO**

17658 `exec`, `fork()`, `getpgrp()`, `getpid()`, `getsid()`, `setpgid()`, `setsid()`, the Base Definitions volume of  
17659 IEEE Std 1003.1-200x, `<unistd.h>`

17660 **CHANGE HISTORY**

17661 First released in Issue 4, Version 2.

17662 **Issue 5**

17663 Moved from X/OPEN UNIX extension to BASE.

17664 **NAME**

17665           getpgrp — get the process group ID of the calling process

17666 **SYNOPSIS**

17667           #include <unistd.h>  
17668           pid\_t getpgrp(void);

17669 **DESCRIPTION**

17670           The *getpgrp()* function shall return the process group ID of the calling process.

17671 **RETURN VALUE**

17672           The *getpgrp()* function shall always be successful and no return value is reserved to indicate an  
17673           error. |

17674 **ERRORS**

17675           No errors are defined.

17676 **EXAMPLES**

17677           None.

17678 **APPLICATION USAGE**

17679           None.

17680 **RATIONALE**

17681           4.3 BSD provides a *getpgrp()* function that returns the process group ID for a specified process. |  
17682           Although this function supports job control, all known job control shells always specify the |  
17683           calling process with this function. Thus, the simpler System V *getpgrp()* suffices, and the added  
17684           complexity of the 4.3 BSD *getpgrp()* is provided by the XSI extension *getpgid()*.

17685 **FUTURE DIRECTIONS**

17686           None.

17687 **SEE ALSO**

17688           *exec*, *fork()*, *getpgid()*, *getpid()*, *getppid()*, *kill()*, *setpgid()*, *setsid()*, the Base Definitions volume of  
17689           IEEE Std 1003.1-200x, <sys/types.h>, <unistd.h>

17690 **CHANGE HISTORY**

17691           First released in Issue 1. Derived from Issue 1 of the SVID.

17692 **Issue 6**

17693           In the SYNOPSIS, the optional include of the <sys/types.h> header is removed.

17694           The following new requirements on POSIX implementations derive from alignment with the  
17695           Single UNIX Specification:

- 17696           • The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was  
17697           required for conforming implementations of previous POSIX specifications, it was not  
17698           required for UNIX applications.

17699 **NAME**

17700            *getpid* — get the process ID

17701 **SYNOPSIS**

17702            #include <unistd.h>

17703            pid\_t *getpid*(void);

17704 **DESCRIPTION**

17705            The *getpid*() function shall return the process ID of the calling process.

17706 **RETURN VALUE**

17707            The *getpid*() function shall always be successful and no return value is reserved to indicate an error.  
17708

17709 **ERRORS**

17710            No errors are defined.

17711 **EXAMPLES**

17712            None.

17713 **APPLICATION USAGE**

17714            None.

17715 **RATIONALE**

17716            None.

17717 **FUTURE DIRECTIONS**

17718            None.

17719 **SEE ALSO**

17720            *exec*, *fork*(), *getpgrp*(), *getppid*(), *kill*(), *setpgid*(), *setsid*(), the Base Definitions volume of  
17721 IEEE Std 1003.1-200x, <sys/types.h>, <unistd.h>

17722 **CHANGE HISTORY**

17723            First released in Issue 1. Derived from Issue 1 of the SVID.

17724 **Issue 6**

17725            In the SYNOPSIS, the optional include of the <sys/types.h> header is removed.

17726            The following new requirements on POSIX implementations derive from alignment with the  
17727 Single UNIX Specification:

- 17728
  - The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was  
17729 required for conforming implementations of previous POSIX specifications, it was not  
17730 required for UNIX applications.

17731 **NAME**

17732       getpmsg — receive next message from a STREAMS file

17733 **SYNOPSIS**

```
17734 xSI     #include <stropts.h>
```

```
17735       int getpmsg(int fildev, struct strbuf *restrict ctlptr,  
17736                 struct strbuf *restrict dataptr, int *restrict bandp,  
17737                 int *restrict flagsp);
```

17738

17739 **DESCRIPTION**

17740       Refer to *getmsg()*.

17741 **NAME**

17742           getppid — get the parent process ID

17743 **SYNOPSIS**

17744           #include <unistd.h>

17745           pid\_t getppid(void);

17746 **DESCRIPTION**

17747           The *getppid()* function shall return the parent process ID of the calling process.

17748 **RETURN VALUE**

17749           The *getppid()* function shall always be successful and no return value is reserved to indicate an error.

17751 **ERRORS**

17752           No errors are defined.

17753 **EXAMPLES**

17754           None.

17755 **APPLICATION USAGE**

17756           None.

17757 **RATIONALE**

17758           None.

17759 **FUTURE DIRECTIONS**

17760           None.

17761 **SEE ALSO**

17762           *exec*, *fork()*, *getpgid()*, *getpgrp()*, *getpid()*, *kill()*, *setpgid()*, *setsid()*, the Base Definitions volume of IEEE Std 1003.1-200x, <sys/types.h>, <unistd.h>

17764 **CHANGE HISTORY**

17765           First released in Issue 1. Derived from Issue 1 of the SVID.

17766 **Issue 6**

17767           In the SYNOPSIS, the optional include of the <sys/types.h> header is removed.

17768           The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- 17770
  - The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.
- 17771
- 17772

## 17773 NAME

17774 getpriority, setpriority — get and set the nice value

## 17775 SYNOPSIS

17776 xSI #include &lt;sys/resource.h&gt;

17777 int getpriority(int which, id\_t who);

17778 int setpriority(int which, id\_t who, int value);

17779

## 17780 DESCRIPTION

17781 The *getpriority()* function shall obtain the nice value of a process, process group, or user. The  
 17782 *setpriority()* function shall set the nice value of a process, process group, or user to  
 17783 *value*+{NZERO}.

17784 Target processes are specified by the values of the *which* and *who* arguments. The *which*  
 17785 argument may be one of the following values: PRIO\_PROCESS, PRIO\_PGRP, or PRIO\_USER,  
 17786 indicating that the *who* argument is to be interpreted as a process ID, a process group ID, or an  
 17787 effective user ID, respectively. A 0 value for the *who* argument specifies the current process,  
 17788 process group, or user.

17789 The nice value set with *setpriority()* shall be applied to the process. If the process is multi-  
 17790 threaded, the nice value shall affect all system scope threads in the process.

17791 If more than one process is specified, *getpriority()* shall return value {NZERO} less than the  
 17792 lowest nice value pertaining to any of the specified processes, and *setpriority()* shall set the nice  
 17793 values of all of the specified processes to *value*+{NZERO}.

17794 The default nice value is {NZERO}; lower nice values shall cause more favorable scheduling.  
 17795 While the range of valid nice values is [0,{NZERO}\*2-1], implementations may enforce more  
 17796 restrictive limits. If *value*+{NZERO} is less than the system's lowest supported nice value,  
 17797 *setpriority()* shall set the nice value to the lowest supported value; if *value*+{NZERO} is greater  
 17798 than the system's highest supported nice value, *setpriority()* shall set the nice value to the highest  
 17799 supported value.

17800 Only a process with appropriate privileges can lower its nice value.

17801 PS|TPS Any processes or threads using SCHED\_FIFO or SCHED\_RR shall be unaffected by a call to  
 17802 *setpriority()*. This is not considered an error. A process which subsequently reverts to  
 17803 SCHED\_OTHER need not have its priority affected by such a *setpriority()* call.

17804 The effect of changing the nice value may vary depending on the process-scheduling algorithm  
 17805 in effect.

17806 Since *getpriority()* can return the value -1 on successful completion, it is necessary to set *errno* to  
 17807 0 prior to a call to *getpriority()*. If *getpriority()* returns the value -1, then *errno* can be checked to  
 17808 see if an error occurred or if the value is a legitimate nice value.

## 17809 RETURN VALUE

17810 Upon successful completion, *getpriority()* shall return an integer in the range from -{NZERO} to  
 17811 {NZERO}-1. Otherwise, -1 shall be returned and *errno* set to indicate the error.

17812 Upon successful completion, *setpriority()* shall return 0; otherwise, -1 shall be returned and *errno*  
 17813 set to indicate the error.

## 17814 ERRORS

17815 The *getpriority()* and *setpriority()* functions shall fail if:

17816 [ESRCH] No process could be located using the *which* and *who* argument values  
 17817 specified.

17818 [EINVAL] The value of the *which* argument was not recognized, or the value of the *who*  
 17819 argument is not a valid process ID, process group ID, or user ID.

17820 In addition, *setpriority()* may fail if:

17821 [EPERM] A process was located, but neither the real nor effective user ID of the  
 17822 executing process match the effective user ID of the process whose nice value  
 17823 is being changed.

17824 [EACCES] A request was made to change the nice value to a lower numeric value and  
 17825 the current process does not have appropriate privileges.

#### 17826 EXAMPLES

##### 17827 Using *getpriority()*

17828 The following example returns the current scheduling priority for the process ID returned by the  
 17829 call to *getpid()*.

```
17830 #include <sys/resource.h>
17831 ...
17832 int which = PRIO_PROCESS;
17833 id_t pid;
17834 int ret;

17835 pid = getpid();
17836 ret = getpriority(which, pid);
```

##### 17837 Using *setpriority()*

17838 The following example sets the priority for the current process ID to  $-20$ .

```
17839 #include <sys/resource.h>
17840 ...
17841 int which = PRIO_PROCESS;
17842 id_t pid;
17843 int priority = -20;
17844 int ret;

17845 pid = getpid();
17846 ret = setpriority(which, pid, priority);
```

#### 17847 APPLICATION USAGE

17848 The *getpriority()* and *setpriority()* functions work with an offset nice value (nice value  
 17849  $-\{\text{NZERO}\}$ ). The nice value is in the range  $[0, 2 * \{\text{NZERO}\} - 1]$ , while the return value for  
 17850 *getpriority()* and the third parameter for *setpriority()* are in the range  $[-\{\text{NZERO}\}, \{\text{NZERO}\} - 1]$ .

#### 17851 RATIONALE

17852 None.

#### 17853 FUTURE DIRECTIONS

17854 None.

#### 17855 SEE ALSO

17856 *nice()*, *sched\_get\_priority\_max()*, *sched\_setscheduler()*, the Base Definitions volume of  
 17857 IEEE Std 1003.1-200x, <sys/resource.h>

17858 **CHANGE HISTORY**

17859           First released in Issue 4, Version 2.

17860 **Issue 5**

17861           Moved from X/OPEN UNIX extension to BASE.

17862           The DESCRIPTION is reworded in terms of the nice value rather than *priority* to avoid confusion  
17863           with functionality in the POSIX Realtime Extension.

17864 **NAME**

17865       getprotobyname — network protocol database functions

17866 **SYNOPSIS**

17867       #include <netdb.h>

17868       struct protoent \*getprotobyname(const char \*name);

17869 **DESCRIPTION**

17870       Refer to *endprotoent()*.

17871 **NAME**

17872       getprotobynumber — network protocol database functions

17873 **SYNOPSIS**

17874       #include <netdb.h>

17875       struct protoent \*getprotobynumber(int *proto*);

17876 **DESCRIPTION**

17877       Refer to *endprotoent()*.

17878 **NAME**

17879           getprotoent — network protocol database functions

17880 **SYNOPSIS**

17881           #include <netdb.h>

17882           struct protoent \*getprotoent(void);

17883 **DESCRIPTION**

17884           Refer to *endprotoent()*.

17885 **NAME**

17886           getpwent — get user database entry

17887 **SYNOPSIS**

17888 xSI       #include <pwd.h>

17889           struct passwd \*getpwent(void);

17890

17891 **DESCRIPTION**

17892           Refer to *endpwent()*.

## 17893 NAME

17894 getpwnam, getpwnam\_r — search user database for a name

## 17895 SYNOPSIS

17896 #include &lt;pwd.h&gt;

17897 struct passwd \*getpwnam(const char \*name);

17898 TSF int getpwnam\_r(const char \*name, struct passwd \*pwd, char \*buffer,  
17899 size\_t bufsize, struct passwd \*\*result);

17900

## 17901 DESCRIPTION

17902 The *getpwnam()* function shall search the user database for an entry with a matching *name*.17903 The *getpwnam()* function need not be reentrant. A function that is not required to be reentrant is  
17904 not required to be thread-safe.17905 Applications wishing to check for error situations should set *errno* to 0 before calling  
17906 *getpwnam()*. If *getpwnam()* returns a null pointer and *errno* is non-zero, an error occurred.17907 TSF The *getpwnam\_r()* function shall update the **passwd** structure pointed to by *pwd* and store a |  
17908 pointer to that structure at the location pointed to by *result*. The structure shall contain an entry |  
17909 from the user database with a matching *name*. Storage referenced by the structure is allocated  
17910 from the memory provided with the *buffer* parameter, which is *bufsize* bytes in size. The  
17911 maximum size needed for this buffer can be determined with the `{_SC_GETPW_R_SIZE_MAX}`  
17912 *sysconf()* parameter. A NULL pointer shall be returned at the location pointed to by *result* on  
17913 error or if the requested entry is not found.

## 17914 RETURN VALUE

17915 The *getpwnam()* function shall return a pointer to a **struct passwd** with the structure as defined  
17916 in `<pwd.h>` with a matching entry if found. A null pointer shall be returned if the requested  
17917 entry is not found, or an error occurs. On error, *errno* shall be set to indicate the error.17918 The return value may point to a static area which is overwritten by a subsequent call to  
17919 *getpwent()*, *getpwnam()*, or *getpwuid()*.17920 TSF If successful, the *getpwnam\_r()* function shall return zero; otherwise, an error number shall be  
17921 returned to indicate the error.

## 17922 ERRORS

17923 The *getpwnam()* and *getpwnam\_r()* functions may fail if:

17924 [EIO] An I/O error has occurred.

17925 [EINTR] A signal was caught during *getpwnam()*.

17926 [EMFILE] {OPEN\_MAX} file descriptors are currently open in the calling process.

17927 [ENFILE] The maximum allowable number of files is currently open in the system.

17928 The *getpwnam\_r()* function may fail if:17929 TSF [ERANGE] Insufficient storage was supplied via *buffer* and *bufsize* to contain the data to  
17930 be referenced by the resulting **passwd** structure.

17931 **EXAMPLES**17932 **Getting an Entry for the Login Name**

17933 The following example uses the *getlogin()* function to return the name of the user who logged in;  
17934 this information is passed to the *getpwnam()* function to get the user database entry for that user.

```
17935 #include <sys/types.h>
17936 #include <pwd.h>
17937 #include <unistd.h>
17938 #include <stdio.h>
17939 #include <stdlib.h>
17940 ...
17941 char *lgn;
17942 struct passwd *pw;
17943 ...
17944 if ((lgn = getlogin()) == NULL || (pw = getpwnam(lgn)) == NULL) {
17945     fprintf(stderr, "Get of user information failed.\n"); exit(1);
17946 }
17947 ...
```

17948 **APPLICATION USAGE**

17949 Three names associated with the current process can be determined: *getpwuid(geteuid())* returns  
17950 the name associated with the effective user ID of the process; *getlogin()* returns the name  
17951 associated with the current login activity; and *getpwuid(getuid())* returns the name associated  
17952 with the real user ID of the process.

17953 The *getpwnam\_r()* function is thread-safe and returns values in a user-supplied buffer instead of  
17954 possibly using a static data area that may be overwritten by each call.

17955 **RATIONALE**

17956 None.

17957 **FUTURE DIRECTIONS**

17958 None.

17959 **SEE ALSO**

17960 *getpwuid()*, the Base Definitions volume of IEEE Std 1003.1-200x, <limits.h>, <pwd.h>,  
17961 <sys/types.h>

17962 **CHANGE HISTORY**

17963 First released in Issue 1. Derived from System V Release 2.0.

17964 **Issue 5**

17965 Normative text previously in the APPLICATION USAGE section is moved to the RETURN  
17966 VALUE section.

17967 The *getpwnam\_r()* function is included for alignment with the POSIX Threads Extension.

17968 A note indicating that the *getpwnam()* function need not be reentrant is added to the  
17969 DESCRIPTION.

17970 **Issue 6**

17971 The *getpwnam\_r()* function is marked as part of the Thread-Safe Functions option.

17972 The Open Group Corrigendum U028/3 is applied, correcting text in the DESCRIPTION  
17973 describing matching the *name*.

- 17974 In the SYNOPSIS, the optional include of the `<sys/types.h>` header is removed.
- 17975 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.
- 17976 The following new requirements on POSIX implementations derive from alignment with the  
17977 Single UNIX Specification:
- 17978 • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was  
17979 required for conforming implementations of previous POSIX specifications, it was not  
17980 required for UNIX applications.
  - 17981 • In the RETURN VALUE section, the requirement to set *errno* on error is added.
  - 17982 • The [EMFILE], [ENFILE], and [ENXIO] optional error conditions are added.
- 17983 The APPLICATION USAGE section is updated to include a note on the thread-safe function and  
17984 its avoidance of possibly using a static data area.
- 17985 IEEE PASC Interpretation 1003.1 #116 is applied, changing the description of the size of the  
17986 buffer from *bufsize* characters to bytes.

## 17987 NAME

17988 getpwuid, getpwuid\_r — search user database for a user ID

## 17989 SYNOPSIS

17990 #include &lt;pwd.h&gt;

17991 struct passwd \*getpwuid(uid\_t uid);

17992 TSF int getpwuid\_r(uid\_t uid, struct passwd \*pwd, char \*buffer,

17993 size\_t bufsize, struct passwd \*\*result);

17994

## 17995 DESCRIPTION

17996 The *getpwuid()* function shall search the user database for an entry with a matching *uid*.17997 The *getpwuid()* function need not be reentrant. A function that is not required to be reentrant is  
17998 not required to be thread-safe.17999 Applications wishing to check for error situations should set *errno* to 0 before calling *getpwuid()*.  
18000 If *getpwuid()* returns a null pointer and *errno* is set to non-zero, an error occurred.18001 TSF The *getpwuid\_r()* function shall update the **passwd** structure pointed to by *pwd* and store a |  
18002 pointer to that structure at the location pointed to by *result*. The structure shall contain an entry |  
18003 from the user database with a matching *uid*. Storage referenced by the structure is allocated  
18004 from the memory provided with the *buffer* parameter, which is *bufsize* bytes in size. The  
18005 maximum size needed for this buffer can be determined with the `{_SC_GETPW_R_SIZE_MAX}`  
18006 *sysconf()* parameter. A NULL pointer shall be returned at the location pointed to by *result* on  
18007 error or if the requested entry is not found.

## 18008 RETURN VALUE

18009 The *getpwuid()* function shall return a pointer to a **struct passwd** with the structure as defined in  
18010 <**pwd.h**> with a matching entry if found. A null pointer shall be returned if the requested entry  
18011 is not found, or an error occurs. On error, *errno* shall be set to indicate the error.18012 The return value may point to a static area which is overwritten by a subsequent call to  
18013 *getpwent()*, *getpwnam()*, or *getpwuid()*.18014 TSF If successful, the *getpwuid\_r()* function shall return zero; otherwise, an error number shall be  
18015 returned to indicate the error.

## 18016 ERRORS

18017 The *getpwuid()* and *getpwuid\_r()* functions may fail if:

18018 [EIO] An I/O error has occurred.

18019 [EINTR] A signal was caught during *getpwuid()*.

18020 [EMFILE] {OPEN\_MAX} file descriptors are currently open in the calling process.

18021 [ENFILE] The maximum allowable number of files is currently open in the system.

18022 The *getpwuid\_r()* function may fail if:18023 TSF [ERANGE] Insufficient storage was supplied via *buffer* and *bufsize* to contain the data to  
18024 be referenced by the resulting **passwd** structure.

18025 **EXAMPLES**18026 **Getting an Entry for the Root User**

18027 The following example gets the user database entry for the user with user ID 0 (root).

```
18028 #include <sys/types.h>
18029 #include <pwd.h>
18030 ...
18031 uid_t id = 0;
18032 struct passwd *pwd;
18033 pwd = getpwuid(id);
```

18034 **Finding the Name for the Effective User ID**

18035 The following example defines *pws* as a pointer to a structure of type **passwd**, which is used to  
 18036 store the structure pointer returned by the call to the *getpwuid()* function. The *geteuid()* function  
 18037 shall return the effective user ID of the calling process; this is used as the search criteria for the  
 18038 *getpwuid()* function. The call to *getpwuid()* shall return a pointer to the structure containing that  
 18039 user ID value.

```
18040 #include <unistd.h>
18041 #include <sys/types.h>
18042 #include <pwd.h>
18043 ...
18044 struct passwd *pws;
18045 pws = getpwuid(geteuid());
```

18046 **Finding an Entry in the User Database**

18047 The following example uses *getpwuid()* to search the user database for a user ID that was  
 18048 previously stored in a **stat** structure, then prints out the user name if it is found. If the user is not  
 18049 found, the program prints the numeric value of the user ID for the entry.

```
18050 #include <sys/types.h>
18051 #include <pwd.h>
18052 #include <stdio.h>
18053 ...
18054 struct stat statbuf;
18055 struct passwd *pwd;
18056 ...
18057 if ((pwd = getpwuid(statbuf.st_uid)) != NULL)
18058     printf(" %-8.8s", pwd->pw_name);
18059 else
18060     printf(" %-8d", statbuf.st_uid);
```

18061 **APPLICATION USAGE**

18062 Three names associated with the current process can be determined: *getpwuid(geteuid())* returns  
 18063 the name associated with the effective user ID of the process; *getlogin()* returns the name  
 18064 associated with the current login activity; and *getpwuid(getuid())* returns the name associated  
 18065 with the real user ID of the process.

18066 The *getpwuid\_r()* function is thread-safe and returns values in a user-supplied buffer instead of  
 18067 possibly using a static data area that may be overwritten by each call.

18068 **RATIONALE**

18069 None.

18070 **FUTURE DIRECTIONS**

18071 None.

18072 **SEE ALSO**

18073 *getpwnam()*, *geteuid()*, *getuid()*, *getlogin()*, the Base Definitions volume of IEEE Std 1003.1-200x,  
18074 `<limits.h>`, `<pwd.h>`, `<sys/types.h>`

18075 **CHANGE HISTORY**

18076 First released in Issue 1. Derived from System V Release 2.0.

18077 **Issue 5**

18078 Normative text previously in the APPLICATION USAGE section is moved to the RETURN  
18079 VALUE section.

18080 The *getpwuid\_r()* function is included for alignment with the POSIX Threads Extension.

18081 A note indicating that the *getpwuid()* function need not be reentrant is added to the  
18082 DESCRIPTION.

18083 **Issue 6**18084 The *getpwuid\_r()* function is marked as part of the Thread-Safe Functions option.

18085 The Open Group Corrigendum U028/3 is applied, correcting text in the DESCRIPTION  
18086 describing matching the *uid*.

18087 In the SYNOPSIS, the optional include of the `<sys/types.h>` header is removed.

18088 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

18089 The following new requirements on POSIX implementations derive from alignment with the  
18090 Single UNIX Specification:

18091 • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was  
18092 required for conforming implementations of previous POSIX specifications, it was not  
18093 required for UNIX applications.

18094 • In the RETURN VALUE section, the requirement to set *errno* on error is added.

18095 • The [EIO], [EINTR], [EMFILE], and [ENFILE] optional error conditions are added.

18096 The APPLICATION USAGE section is updated to include a note on the thread-safe function and  
18097 its avoidance of possibly using a static data area.

18098 IEEE PASC Interpretation 1003.1 #116 is applied, changing the description of the size of the  
18099 buffer from *bufsize* characters to bytes.

## 18100 NAME

18101 getrlimit, setrlimit — control maximum resource consumption

## 18102 SYNOPSIS

18103 xSI #include &lt;sys/resource.h&gt;

18104 int getrlimit(int resource, struct rlimit \*rlp);

18105 int setrlimit(int resource, const struct rlimit \*rlp);

18106

## 18107 DESCRIPTION

18108 The *getrlimit()* function shall get, and the *setrlimit()* function shall set, limits on the consumption |  
18109 of a variety of resources. |18110 Each call to either *getrlimit()* or *setrlimit()* identifies a specific resource to be operated upon as  
18111 well as a resource limit. A resource limit is represented by an **rlimit** structure. The *rlim\_cur*  
18112 member specifies the current or soft limit and the *rlim\_max* member specifies the maximum or  
18113 hard limit. Soft limits may be changed by a process to any value that is less than or equal to the  
18114 hard limit. A process may (irreversibly) lower its hard limit to any value that is greater than or  
18115 equal to the soft limit. Only a process with appropriate privileges can raise a hard limit. Both  
18116 hard and soft limits can be changed in a single call to *setrlimit()* subject to the constraints  
18117 described above.18118 The value RLIM\_INFINITY, defined in <sys/resource.h>, shall be considered to be larger than  
18119 any other limit value. If a call to *getrlimit()* returns RLIM\_INFINITY for a resource, it means the  
18120 implementation shall not enforce limits on that resource. Specifying RLIM\_INFINITY as any  
18121 resource limit value on a successful call to *setrlimit()* shall inhibit enforcement of that resource  
18122 limit.

18123 The following resources are defined:

18124 RLIMIT\_CORE This is the maximum size of a core file, in bytes, that may be created by a |  
18125 process. A limit of 0 shall prevent the creation of a core file. If this limit is |  
18126 exceeded, the writing of a core file shall terminate at this size.18127 RLIMIT\_CPU This is the maximum amount of CPU time, in seconds, used by a process. |  
18128 If this limit is exceeded, SIGXCPU shall be generated for the process. If |  
18129 the process is catching or ignoring SIGXCPU, or all threads belonging to |  
18130 that process are blocking SIGXCPU, the behavior is unspecified.18131 RLIMIT\_DATA This is the maximum size of a process' data segment, in bytes. If this limit |  
18132 is exceeded, the *malloc()* function shall fail with *errno* set to [ENOMEM].18133 RLIMIT\_FSIZE This is the maximum size of a file, in bytes, that may be created by a |  
18134 process. If a write or truncate operation would cause this limit to be |  
18135 exceeded, SIGXFSZ shall be generated for the thread. If the thread is |  
18136 blocking, or the process is catching or ignoring SIGXFSZ, continued |  
18137 attempts to increase the size of a file from end-of-file to beyond the limit |  
18138 shall fail with *errno* set to [EFBIG].18139 RLIMIT\_NOFILE This is a number one greater than the maximum value that the system |  
18140 may assign to a newly-created descriptor. If this limit is exceeded, |  
18141 functions that allocate new file descriptors may fail with *errno* set to |  
18142 [EMFILE]. This limit constrains the number of file descriptors that a |  
18143 process may allocate.18144 RLIMIT\_STACK This is the maximum size of a process' stack, in bytes. The |  
18145 implementation does not automatically grow the stack beyond this limit.

18146 If this limit is exceeded, SIGSEGV shall be generated for the thread. If the  
 18147 thread is blocking SIGSEGV, or the process is ignoring or catching  
 18148 SIGSEGV and has not made arrangements to use an alternate stack, the  
 18149 disposition of SIGSEGV shall be set to SIG\_DFL before it is generated.

18150 **RLIMIT\_AS** This is the maximum size of a process' total available memory, in bytes. If  
 18151 this limit is exceeded, the *malloc()* and *mmap()* functions shall fail with  
 18152 *errno* set to [ENOMEM]. In addition, the automatic stack growth fails  
 18153 with the effects outlined above.

18154 When using the *getrlimit()* function, if a resource limit can be represented correctly in an object  
 18155 of type **rlim\_t**, then its representation is returned; otherwise, if the value of the resource limit is  
 18156 equal to that of the corresponding saved hard limit, the value returned shall be  
 18157 RLIM\_SAVED\_MAX; otherwise, the value returned shall be RLIM\_SAVED\_CUR.

18158 When using the *setrlimit()* function, if the requested new limit is RLIM\_INFINITY, the new limit  
 18159 shall be “no limit”; otherwise, if the requested new limit is RLIM\_SAVED\_MAX, the new limit  
 18160 shall be the corresponding saved hard limit; otherwise, if the requested new limit is  
 18161 RLIM\_SAVED\_CUR, the new limit shall be the corresponding saved soft limit; otherwise, the  
 18162 new limit shall be the requested value. In addition, if the corresponding saved limit can be  
 18163 represented correctly in an object of type **rlim\_t** then it shall be overwritten with the new limit.

18164 The result of setting a limit to RLIM\_SAVED\_MAX or RLIM\_SAVED\_CUR is unspecified unless  
 18165 a previous call to *getrlimit()* returned that value as the soft or hard limit for the corresponding  
 18166 resource limit.

18167 The determination of whether a limit can be correctly represented in an object of type **rlim\_t** is  
 18168 implementation-defined. For example, some implementations permit a limit whose value is  
 18169 greater than RLIM\_INFINITY and others do not.

18170 The *exec* family of functions shall cause resource limits to be saved. |

18171 **RETURN VALUE**

18172 Upon successful completion, *getrlimit()* and *setrlimit()* shall return 0. Otherwise, these functions  
 18173 shall return -1 and set *errno* to indicate the error.

18174 **ERRORS**

18175 The *getrlimit()* and *setrlimit()* functions shall fail if:

18176 [EINVAL] An invalid *resource* was specified; or in a *setrlimit()* call, the new *rlim\_cur*  
 18177 exceeds the new *rlim\_max*.

18178 [EPERM] The limit specified to *setrlimit()* would have raised the maximum limit value,  
 18179 and the calling process does not have appropriate privileges.

18180 The *setrlimit()* function may fail if:

18181 [EINVAL] The limit specified cannot be lowered because current usage is already higher  
 18182 than the limit.

18183 **EXAMPLES**

18184           None.

18185 **APPLICATION USAGE**

18186           If a process attempts to set the hard limit or soft limit for RLIMIT\_NOFILE to less than the value  
18187           of `{_POSIX_OPEN_MAX}` from `<limits.h>`, unexpected behavior may occur.

18188 **RATIONALE**

18189           None.

18190 **FUTURE DIRECTIONS**

18191           None.

18192 **SEE ALSO**

18193           *exec*, *fork()*, *malloc()*, *open()*, *sigaltstack()*, *sysconf()*, *ulimit()*, the Base Definitions volume of  
18194           IEEE Std 1003.1-200x, `<stropts.h>`, `<sys/resource.h>`

18195 **CHANGE HISTORY**

18196           First released in Issue 4, Version 2.

18197 **Issue 5**

18198           Moved from X/OPEN UNIX extension to BASE and an APPLICATION USAGE section is added.

18199           Large File Summit extensions are added.

18200 **NAME**

18201        getrusage — get information about resource utilization

18202 **SYNOPSIS**

18203 XSI        #include &lt;sys/resource.h&gt;

18204        int getrusage(int who, struct rusage \*r\_usage);

18205

18206 **DESCRIPTION**

18207        The *getrusage()* function shall provide measures of the resources used by the current process or  
 18208        its terminated and waited-for child processes. If the value of the *who* argument is  
 18209        RUSAGE\_SELF, information shall be returned about resources used by the current process. If the  
 18210        value of the *who* argument is RUSAGE\_CHILDREN, information shall be returned about  
 18211        resources used by the terminated and waited-for children of the current process. If the child is  
 18212        never waited for (for example, if the parent has SA\_NOCLDWAIT set or sets SIGCHLD to  
 18213        SIG\_IGN), the resource information for the child process is discarded and not included in the  
 18214        resource information provided by *getrusage()*.

18215        The *r\_usage* argument is a pointer to an object of type **struct rusage** in which the returned  
 18216        information is stored.

18217 **RETURN VALUE**

18218        Upon successful completion, *getrusage()* shall return 0; otherwise, -1 shall be returned and *errno*  
 18219        set to indicate the error.

18220 **ERRORS**18221        The *getrusage()* function shall fail if:18222        [EINVAL]        The value of the *who* argument is not valid.18223 **EXAMPLES**18224        **Using getrusage()**

18225        The following example returns information about the resources used by the current process.

18226        #include &lt;sys/resource.h&gt;

18227        ...

18228        int who = RUSAGE\_SELF;

18229        struct rusage usage;

18230        int ret;

18231        ret = getrusage(who, &amp;usage);

18232 **APPLICATION USAGE**

18233        None.

18234 **RATIONALE**

18235        None.

18236 **FUTURE DIRECTIONS**

18237        None.

18238 **SEE ALSO**18239        *exit()*, *sigaction()*, *time()*, *times()*, *wait()*, the Base Definitions volume of IEEE Std 1003.1-200x,

18240        &lt;sys/resource.h&gt;

18241 **CHANGE HISTORY**

18242 First released in Issue 4, Version 2.

18243 **Issue 5**

18244 Moved from X/OPEN UNIX extension to BASE.

18245 **NAME**

18246 gets — get a string from a stdin stream

18247 **SYNOPSIS**

18248 #include &lt;stdio.h&gt;

18249 char \*gets(char \*s);

18250 **DESCRIPTION**

18251 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
18252 conflict between the requirements described here and the ISO C standard is unintentional. This  
18253 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

18254 The *gets()* function shall read bytes from the standard input stream, *stdin*, into the array pointed  
18255 to by *s*, until a newline is read or an end-of-file condition is encountered. Any <newline> shall  
18256 be discarded and a null byte shall be placed immediately after the last byte read into the array.

18257 cx The *gets()* function may mark the *st\_atime* field of the file associated with *stream* for update. The  
18258 *st\_atime* field shall be marked for update by the first successful execution of *fgetc()*, *fgets()*,  
18259 *fread()*, *getc()*, *getchar()*, *gets()*, *fscanf()*, or *scanf()* using *stream* that returns data not supplied by  
18260 a prior call to *ungetc()*.

18261 **RETURN VALUE**

18262 Upon successful completion, *gets()* shall return *s*. If the stream is at end-of-file, the end-of-file  
18263 indicator for the stream shall be set and *gets()* shall return a null pointer. If a read error occurs,  
18264 cx the error indicator for the stream shall be set, *gets()* shall return a null pointer and set *errno* to  
18265 indicate the error.

18266 **ERRORS**18267 Refer to *fgetc()*.18268 **EXAMPLES**

18269 None.

18270 **APPLICATION USAGE**

18271 Reading a line that overflows the array pointed to by *s* results in undefined behavior. The use of  
18272 *fgets()* is recommended.

18273 Since the user cannot specify the length of the buffer passed to *gets()*, use of this function is  
18274 discouraged. The length of the string read is unlimited. It is possible to overflow this buffer in  
18275 such a way as to cause applications to fail, or possible system security violations.

18276 It is recommended that the *fgets()* function should be used to read input lines.

18277 **RATIONALE**

18278 None.

18279 **FUTURE DIRECTIONS**

18280 None.

18281 **SEE ALSO**18282 *feof()*, *ferror()*, *fgets()*, the Base Definitions volume of IEEE Std 1003.1-200x, <stdio.h>18283 **CHANGE HISTORY**

18284 First released in Issue 1. Derived from Issue 1 of the SVID.

18285 **Issue 6**

18286 Extensions beyond the ISO C standard are now marked.

18287 **NAME**

18288        getservbyname — network services database functions

18289 **SYNOPSIS**

18290        #include <netdb.h>

18291        struct servent \*getservbyname(const char \*name, const char \*proto);

18292 **DESCRIPTION**

18293        Refer to *endservent()*.

18294 **NAME**

18295        getservbyport — network services database functions

18296 **SYNOPSIS**

18297        #include <netdb.h>

18298        struct servent \*getservbyport(int *port*, const char \**proto*);

18299 **DESCRIPTION**

18300        Refer to *endservent()*.

18301 **NAME**

18302        getservent — network services database functions

18303 **SYNOPSIS**

18304        #include <netdb.h>

18305        struct servent \*getservent(void);

18306 **DESCRIPTION**

18307        Refer to *endservent()*.

18308 **NAME**

18309            getsid — get the process group ID of a session leader

18310 **SYNOPSIS**

18311 XSI        #include <unistd.h>

18312            pid\_t getsid(pid\_t pid);

18313

18314 **DESCRIPTION**

18315            The *getsid()* function shall obtain the process group ID of the process that is the session leader of  
18316            the process specified by *pid*. If *pid* is (**pid\_t**)0, it specifies the calling process.

18317 **RETURN VALUE**

18318            Upon successful completion, *getsid()* shall return the process group ID of the session leader of  
18319            the specified process. Otherwise, it shall return (**pid\_t**)-1 and set *errno* to indicate the error.

18320 **ERRORS**

18321            The *getsid()* function shall fail if:

18322            [EPERM]            The process specified by *pid* is not in the same session as the calling process,  
18323            and the implementation does not allow access to the process group ID of the  
18324            session leader of that process from the calling process.

18325            [ESRCH]            There is no process with a process ID equal to *pid*.

18326 **EXAMPLES**

18327            None.

18328 **APPLICATION USAGE**

18329            None.

18330 **RATIONALE**

18331            None.

18332 **FUTURE DIRECTIONS**

18333            None.

18334 **SEE ALSO**

18335            *exec*, *fork()*, *getpid()*, *getpgid()*, *setpgid()*, *setsid()*, the Base Definitions volume of  
18336            IEEE Std 1003.1-200x, <unistd.h>

18337 **CHANGE HISTORY**

18338            First released in Issue 4, Version 2.

18339 **Issue 5**

18340            Moved from X/OPEN UNIX extension to BASE.

18341 **NAME**

18342 getsockname — get the socket name

18343 **SYNOPSIS**

18344 #include &lt;sys/socket.h&gt;

18345 int getsockname(int *socket*, struct sockaddr \*restrict *address*,  
18346 socklen\_t \*restrict *address\_len*);18347 **DESCRIPTION**18348 The *getsockname()* function shall retrieve the locally-bound name of the specified socket, store  
18349 this address in the **sockaddr** structure pointed to by the *address* argument, and store the length of  
18350 this address in the object pointed to by the *address\_len* argument.18351 If the actual length of the address is greater than the length of the supplied **sockaddr** structure,  
18352 the stored address shall be truncated.18353 If the socket has not been bound to a local name, the value stored in the object pointed to by  
18354 *address* is unspecified.18355 **RETURN VALUE**18356 Upon successful completion, 0 shall be returned, the *address* argument shall point to the address  
18357 of the socket, and the *address\_len* argument shall point to the length of the address. Otherwise, -1  
18358 shall be returned and *errno* set to indicate the error.18359 **ERRORS**18360 The *getsockname()* function shall fail if:18361 [EBADF] The *socket* argument is not a valid file descriptor.18362 [ENOTSOCK] The *socket* argument does not refer to a socket.

18363 [EOPNOTSUPP] The operation is not supported for this socket's protocol.

18364 The *getsockname()* function may fail if:

18365 [EINVAL] The socket has been shut down.

18366 [ENOBUFS] Insufficient resources were available in the system to complete the function.

18367 **EXAMPLES**

18368 None.

18369 **APPLICATION USAGE**

18370 None.

18371 **RATIONALE**

18372 None.

18373 **FUTURE DIRECTIONS**

18374 None.

18375 **SEE ALSO**18376 *accept()*, *bind()*, *getpeername()*, *socket()*, the Base Definitions volume of IEEE Std 1003.1-200x,  
18377 <sys/socket.h>18378 **CHANGE HISTORY**

18379 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

18380 The **restrict** keyword is added to the *getsockname()* prototype for alignment with the  
18381 ISO/IEC 9899:1999 standard.

## 18382 NAME

18383 getsockopt — get the socket options

## 18384 SYNOPSIS

18385 #include &lt;sys/socket.h&gt;

```
18386 int getsockopt(int socket, int level, int option_name,
18387               void *restrict option_value, socklen_t *restrict option_len);
```

## 18388 DESCRIPTION

18389 The *getsockopt()* function manipulates options associated with a socket.

18390 The *getsockopt()* function shall retrieve the value for the option specified by the *option\_name*  
 18391 argument for the socket specified by the *socket* argument. If the size of the option value is greater  
 18392 than *option\_len*, the value stored in the object pointed to by the *option\_value* argument shall be  
 18393 silently truncated. Otherwise, the object pointed to by the *option\_len* argument shall be modified  
 18394 to indicate the actual length of the value.

18395 The *level* argument specifies the protocol level at which the option resides. To retrieve options at  
 18396 the socket level, specify the *level* argument as SOL\_SOCKET. To retrieve options at other levels,  
 18397 supply the appropriate level identifier for the protocol controlling the option. For example, to  
 18398 indicate that an option is interpreted by the TCP (Transmission Control Protocol), set *level* to  
 18399 IPPROTO\_TCP as defined in the <netinet/in.h> header.

18400 The socket in use may require the process to have appropriate privileges to use the *getsockopt()*  
 18401 function.

18402 The *option\_name* argument specifies a single option to be retrieved. It can be one of the following  
 18403 values defined in <sys/socket.h>:

18404	SO_DEBUG	Reports whether debugging information is being recorded. This option	
18405		shall store an <b>int</b> value. This is a Boolean option.	
18406	SO_ACCEPTCONN	Reports whether socket listening is enabled. This option shall store an <b>int</b>	
18407		value. This is a Boolean option.	
18408	SO_BROADCAST	Reports whether transmission of broadcast messages is supported, if this	
18409		is supported by the protocol. This option shall store an <b>int</b> value. This is a	
18410		Boolean option.	
18411	SO_REUSEADDR	Reports whether the rules used in validating addresses supplied to <i>bind()</i>	
18412		should allow reuse of local addresses, if this is supported by the protocol.	
18413		This option shall store an <b>int</b> value. This is a Boolean option.	
18414	SO_KEEPALIVE	Reports whether connections are kept active with periodic transmission	
18415		of messages, if this is supported by the protocol.	
18416		If the connected socket fails to respond to these messages, the connection	
18417		shall be broken and threads writing to that socket shall be notified with a	
18418		SIGPIPE signal. This option shall store an <b>int</b> value. This is a Boolean	
18419		option.	
18420	SO_LINGER	Reports whether the socket lingers on <i>close()</i> if data is present. If	
18421		SO_LINGER is set, the system blocks the process during <i>close()</i> until it	
18422		can transmit the data or until the end of the interval indicated by the	
18423		<i>l_linger</i> member, whichever comes first. If SO_LINGER is not specified,	
18424		and <i>close()</i> is issued, the system handles the call in a way that allows the	
18425		process to continue as quickly as possible. This option shall store a <b>linger</b>	
18426		structure.	

18427	SO_OOBINLINE	Reports whether the socket leaves received out-of-band data (data marked urgent) inline. This option shall store an <b>int</b> value. This is a Boolean option.
18428		
18429		
18430	SO_SNDBUF	Reports send buffer size information. This option shall store an <b>int</b> value.
18431	SO_RCVBUF	Reports receive buffer size information. This option shall store an <b>int</b> value.
18432		
18433	SO_ERROR	Reports information about error status and clears it. This option shall store an <b>int</b> value.
18434		
18435	SO_TYPE	Reports the socket type. This option shall store an <b>int</b> value. Socket types are described in Section 2.10.6 (on page 509).
18436		
18437	SO_DONTROUTE	Reports whether outgoing messages bypass the standard routing facilities. The destination shall be on a directly-connected network, and messages are directed to the appropriate network interface according to the destination address. The effect, if any, of this option depends on what protocol is in use. This option shall store an <b>int</b> value. This is a Boolean option.
18438		
18439		
18440		
18441		
18442		
18443	SO_RCVLOWAT	Reports the minimum number of bytes to process for socket input operations. The default value for SO_RCVLOWAT is 1. If SO_RCVLOWAT is set to a larger value, blocking receive calls normally wait until they have received the smaller of the low water mark value or the requested amount. (They may return less than the low water mark if an error occurs, a signal is caught, or the type of data next in the receive queue is different from that returned; for example, out-of-band data.) This option shall store an <b>int</b> value. Note that not all implementations allow this option to be retrieved.
18444		
18445		
18446		
18447		
18448		
18449		
18450		
18451		
18452	SO_RCVTIMEO	Reports the timeout value for input operations. This option shall store a <b>timeval</b> structure with the number of seconds and microseconds specifying the limit on how long to wait for an input operation to complete. If a receive operation has blocked for this much time without receiving additional data, it shall return with a partial count or <i>errno</i> set to [EAGAIN] or [EWOULDBLOCK] if no data was received. The default for this option is zero, which indicates that a receive operation shall not time out. Note that not all implementations allow this option to be retrieved.
18453		
18454		
18455		
18456		
18457		
18458		
18459		
18460	SO_SNDLOWAT	Reports the minimum number of bytes to process for socket output operations. Non-blocking output operations shall process no data if flow control does not allow the smaller of the send low water mark value or the entire request to be processed. This option shall store an <b>int</b> value. Note that not all implementations allow this option to be retrieved.
18461		
18462		
18463		
18464		
18465	SO_SNDTIMEO	Reports the timeout value specifying the amount of time that an output function blocks because flow control prevents data from being sent. If a send operation has blocked for this time, it shall return with a partial count or with <i>errno</i> set to [EAGAIN] or [EWOULDBLOCK] if no data were sent. The default for this option is zero, which indicates that a send operation shall not time out. The option shall store a <b>timeval</b> structure. Note that not all implementations allow this option to be retrieved.
18466		
18467		
18468		
18469		
18470		
18471		
18472		
18473		
		For Boolean options, a zero value indicates that the option is disabled and a non-zero value indicates that the option is enabled.

18474 Options at other protocol levels vary in format and name.  
18475 The socket in use may require the process to have appropriate privileges to use the *getsockopt()*  
18476 function.

**18477 RETURN VALUE**

18478 Upon successful completion, *getsockopt()* shall return 0; otherwise, -1 shall be returned and *errno*  
18479 set to indicate the error.

**18480 ERRORS**

18481 The *getsockopt()* function shall fail if:

18482 [EBADF] The *socket* argument is not a valid file descriptor.

18483 [EINVAL] The specified option is invalid at the specified socket level.

18484 [ENOPROTOOPT]

18485 The option is not supported by the protocol.

18486 [ENOTSOCK] The *socket* argument does not refer to a socket.

18487 The *getsockopt()* function may fail if:

18488 [EACCES] The calling process does not have the appropriate privileges.

18489 [EINVAL] The socket has been shut down.

18490 [ENOBUFS] Insufficient resources are available in the system to complete the function.

**18491 EXAMPLES**

18492 None.

**18493 APPLICATION USAGE**

18494 None.

**18495 RATIONALE**

18496 None.

**18497 FUTURE DIRECTIONS**

18498 None.

**18499 SEE ALSO**

18500 *bind()*, *close()*, *endprotoent()*, *setsockopt()*, *socket()*, the Base Definitions volume of  
18501 IEEE Std 1003.1-200x, <sys/socket.h>, <netinet/in.h>

**18502 CHANGE HISTORY**

18503 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

18504 The **restrict** keyword is added to the *getsockopt()* prototype for alignment with the

18505 ISO/IEC 9899:1999 standard.

18506 **NAME**

18507 getsubopt — parse suboption arguments from a string

18508 **SYNOPSIS**18509 XSI `#include <stdlib.h>`18510 `int getsubopt(char **optionp, char * const *tokens, char **valuep);`

18511

18512 **DESCRIPTION**18513 The *getsubopt()* function shall parse suboption arguments in a flag argument. Such options often  
18514 result from the use of *getopt()*.18515 The *getsubopt()* argument *optionp* is a pointer to a pointer to the option argument string. The  
18516 suboption arguments shall be separated by commas and each may consist of either a single  
18517 token, or a token-value pair separated by an equal sign.18518 The *keylistp* argument shall be a pointer to a vector of strings. The end of the vector is identified  
18519 by a null pointer. Each entry in the vector is one of the possible tokens that might be found in  
18520 *\*optionp*. Since commas delimit suboption arguments in *optionp*, they should not appear in any of  
18521 the strings pointed to by *keylistp*. Similarly, because an equal sign separates a token from its  
18522 value, the application should not include an equal sign in any of the strings pointed to by  
18523 *keylistp*.18524 The *valuep* argument is the address of a value string pointer.18525 If a comma appears in *optionp*, it shall be interpreted as a suboption separator. After commas  
18526 have been processed, if there are one or more equal signs in a suboption string, the first equal  
18527 sign in any suboption string shall be interpreted as a separator between a token and a value.  
18528 Subsequent equal signs in a suboption string shall be interpreted as part of the value.18529 If the string at *\*optionp* contains only one suboption argument (equivalently, no commas),  
18530 *getsubopt()* shall update *\*optionp* to point to the nul character at the end of the string. Otherwise,  
18531 it shall isolate the suboption argument by replacing the comma separator with a nul character,  
18532 and shall update *\*optionp* to point to the start of the next suboption argument. If the suboption  
18533 argument has an associated value (equivalently, contains an equal sign), *getsubopt()* shall update  
18534 *\*valuep* to point to the value's first character. Otherwise, it shall set *\*valuep* to a null pointer. The  
18535 calling application may use this information to determine whether the presence or absence of a  
18536 value for the suboption is an error.18537 Additionally, when *getsubopt()* fails to match the suboption argument with a token in the *keylistp*  
18538 array, the calling application should decide if this is an error, or if the unrecognized option  
18539 should be processed in another way.18540 **RETURN VALUE**18541 The *getsubopt()* function shall return the index of the matched token string, or  $-1$  if no token  
18542 strings were matched.18543 **ERRORS**

18544 No errors are defined.

## 18545 EXAMPLES

```

18546     #include <stdio.h>
18547     #include <stdlib.h>

18548     int do_all;
18549     const char *type;
18550     int read_size;
18551     int write_size;
18552     int read_only;

18553     enum
18554     {
18555         RO_OPTION = 0,
18556         RW_OPTION,
18557         READ_SIZE_OPTION,
18558         WRITE_SIZE_OPTION
18559     };

18560     const char *mount_opts[] =
18561     {
18562         [RO_OPTION] = "ro",
18563         [RW_OPTION] = "rw",
18564         [READ_SIZE_OPTION] = "rsize",
18565         [WRITE_SIZE_OPTION] = "wsize",
18566         NULL
18567     };

18568     int
18569     main(int argc, char *argv[])
18570     {
18571         char *subopts, *value;
18572         int opt;

18573         while ((opt = getopt(argc, argv, "at:o:")) != -1)
18574             switch(opt)
18575             {
18576                 case 'a':
18577                     do_all = 1;
18578                     break;
18579                 case 't':
18580                     type = optarg;
18581                     break;
18582                 case 'o':
18583                     subopts = optarg;
18584                     while (*subopts != '\0')
18585                         switch(getsubopt(&subopts, mount_opts, &value))
18586                         {
18587                             case RO_OPTION:
18588                                 read_only = 1;
18589                                 break;
18590                             case RW_OPTION:
18591                                 read_only = 0;
18592                                 break;
18593                             case READ_SIZE_OPTION:

```

```

18594         if (value == NULL)
18595             abort();
18596         read_size = atoi(value);
18597         break;
18598     case WRITE_SIZE_OPTION:
18599         if (value == NULL)
18600             abort();
18601         write_size = atoi(value);
18602         break;
18603     default:
18604         /* Unknown suboption. */
18605         printf("Unknown suboption '%s'\n", value);
18606         break;
18607     }
18608     break;
18609     default:
18610         abort();
18611     }
18612     /* Do the real work. */
18613     return 0;
18614 }

```

### 18615 Parsing Suboptions

18616 The following example uses the *getsubopt()* function to parse a value argument in the *optarg*  
 18617 external variable returned by a call to *getopt()*.

```

18618 #include <stdlib.h>
18619 ...
18620 char *tokens[] = {"HOME", "PATH", "LOGNAME", (char *) NULL };
18621 char *value;
18622 int opt, index;
18623 while ((opt = getopt(argc, argv, "e:")) != -1) {
18624     switch(opt) {
18625     case 'e' :
18626         while ((index = getsubopt(&optarg, tokens, &value)) != -1) {
18627             switch(index) {
18628             ...
18629             }
18630             break;
18631         ...
18632     }
18633     }
18634     ...

```

### 18635 APPLICATION USAGE

18636 None.

### 18637 RATIONALE

18638 None.

18639 **FUTURE DIRECTIONS**

18640           None.

18641 **SEE ALSO**

18642           *getopt()*, the Base Definitions volume of IEEE Std 1003.1-200x, <**stdlib.h**>

18643 **CHANGE HISTORY**

18644           First released in Issue 4, Version 2.

18645 **Issue 5**

18646           Moved from X/OPEN UNIX extension to BASE.

18647 **NAME**

18648           gettimeofday — get the date and time

18649 **SYNOPSIS**

18650 XSI       #include &lt;sys/time.h&gt;

18651           int gettimeofday(struct timeval \*restrict tp, void \*restrict tzp);

18652

18653 **DESCRIPTION**

18654       The *gettimeofday()* function shall obtain the current time, expressed as seconds and |  
18655       microseconds since the Epoch, and store it in the **timeval** structure pointed to by *tp*. The |  
18656       resolution of the system clock is unspecified.

18657       If *tzp* is not a null pointer, the behavior is unspecified.

18658 **RETURN VALUE**

18659       The *gettimeofday()* function shall return 0 and no value shall be reserved to indicate an error.

18660 **ERRORS**

18661       No errors are defined.

18662 **EXAMPLES**

18663       None.

18664 **APPLICATION USAGE**

18665       None.

18666 **RATIONALE**

18667       None.

18668 **FUTURE DIRECTIONS**

18669       None.

18670 **SEE ALSO**

18671       *ctime()*, *ftime()*, the Base Definitions volume of IEEE Std 1003.1-200x, <sys/time.h>

18672 **CHANGE HISTORY**

18673       First released in Issue 4, Version 2.

18674 **Issue 5**

18675       Moved from X/OPEN UNIX extension to BASE.

18676 **Issue 6**

18677       The DESCRIPTION is updated to refer to “seconds since the Epoch” rather than “seconds since  
18678       00:00:00 UTC (Coordinated Universal Time), January 1 1970” for consistency with other *time*  
18679       functions.

18680       The **restrict** keyword is added to the *gettimeofday()* prototype for alignment with the  
18681       ISO/IEC 9899:1999 standard.

18682 **NAME**

18683           getuid — get a real user ID

18684 **SYNOPSIS**

18685           #include &lt;unistd.h&gt;

18686           uid\_t getuid(void);

18687 **DESCRIPTION**18688           The *getuid()* function shall return the real user ID of the calling process.18689 **RETURN VALUE**18690           The *getuid()* function shall always be successful and no return value is reserved to indicate the  
18691           error.18692 **ERRORS**

18693           No errors are defined.

18694 **EXAMPLES**18695           **Setting the Effective User ID to the Real User ID**18696           The following example sets the effective user ID and the real user ID of the current process to the  
18697           real user ID of the caller.18698           #include <unistd.h>  
18699           #include <sys/types.h>  
18700           ...  
18701           setreuid(getuid(), getuid());  
18702           ...18703 **APPLICATION USAGE**

18704           None.

18705 **RATIONALE**

18706           None.

18707 **FUTURE DIRECTIONS**

18708           None.

18709 **SEE ALSO**18710           *getegid()*, *geteuid()*, *getgid()*, *setegid()*, *seteuid()*, *setgid()*, *setregid()*, *setreuid()*, *setuid()*, the Base  
18711           Definitions volume of IEEE Std 1003.1-200x, <sys/types.h>, <unistd.h>18712 **CHANGE HISTORY**

18713           First released in Issue 1. Derived from Issue 1 of the SVID.

18714 **Issue 6**

18715           In the SYNOPSIS, the optional include of the &lt;sys/types.h&gt; header is removed.

18716           The following new requirements on POSIX implementations derive from alignment with the  
18717           Single UNIX Specification:

- 18718
- The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was  
18719           required for conforming implementations of previous POSIX specifications, it was not  
18720           required for UNIX applications.

18721 **NAME**

18722           getutxent, getutxid, getutxline — get user accounting database entries

18723 **SYNOPSIS**

18724 XSI       #include &lt;utmpx.h&gt;

18725           struct utmpx \*getutxent(void);

18726           struct utmpx \*getutxid(const struct utmpx \*id);

18727           struct utmpx \*getutxline(const struct utmpx \*line);

18728

18729 **DESCRIPTION**18730           Refer to *endutxent()*.

18731 **NAME**

18732           getwc — get a wide character from a stream

18733 **SYNOPSIS**

18734           #include <stdio.h>

18735           #include <wchar.h>

18736           wint\_t getwc(FILE \**stream*);

18737 **DESCRIPTION**

18738 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
18739 conflict between the requirements described here and the ISO C standard is unintentional. This  
18740 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

18741       The *getwc()* function shall be equivalent to *fgetwc()*, except that if it is implemented as a macro it  
18742 may evaluate *stream* more than once, so the argument should never be an expression with side  
18743 effects.

18744 **RETURN VALUE**

18745       Refer to *fgetwc()*.

18746 **ERRORS**

18747       Refer to *fgetwc()*.

18748 **EXAMPLES**

18749       None.

18750 **APPLICATION USAGE**

18751       Since it may be implemented as a macro, *getwc()* may treat incorrectly a *stream* argument with  
18752 side effects. In particular, *getwc(\*f++)* does not necessarily work as expected. Therefore, use of  
18753 this function is not recommended; *fgetwc()* should be used instead.

18754 **RATIONALE**

18755       None.

18756 **FUTURE DIRECTIONS**

18757       None.

18758 **SEE ALSO**

18759       *fgetwc()*, the Base Definitions volume of IEEE Std 1003.1-200x, <stdio.h>, <wchar.h>

18760 **CHANGE HISTORY**

18761       First released as a World-wide Portability Interface in Issue 4. Derived from the MSE working  
18762 draft.

18763 **Issue 5**

18764       The Optional Header (OH) marking is removed from <stdio.h>.

18765 **NAME**

18766       getwchar — get a wide character from a stdin stream

18767 **SYNOPSIS**

18768       #include <wchar.h>

18769       wint\_t getwchar(void);

18770 **DESCRIPTION**

18771 cx       The functionality described on this reference page is aligned with the ISO C standard. Any  
18772       conflict between the requirements described here and the ISO C standard is unintentional. This  
18773       volume of IEEE Std 1003.1-200x defers to the ISO C standard.

18774       The *getwchar()* function shall be equivalent to *getwc(stdin)*.

18775 **RETURN VALUE**

18776       Refer to *fgetwc()*.

18777 **ERRORS**

18778       Refer to *fgetwc()*.

18779 **EXAMPLES**

18780       None.

18781 **APPLICATION USAGE**

18782       If the **wint\_t** value returned by *getwchar()* is stored into a variable of type **wchar\_t** and then  
18783       compared against the **wint\_t** macro WEOF, the result may be incorrect. Only the **wint\_t** type is  
18784       guaranteed to be able to represent any wide character and WEOF.

18785 **RATIONALE**

18786       None.

18787 **FUTURE DIRECTIONS**

18788       None.

18789 **SEE ALSO**

18790       *fgetwc()*, *getwc()*, the Base Definitions volume of IEEE Std 1003.1-200x, <**wchar.h**>

18791 **CHANGE HISTORY**

18792       First released as a World-wide Portability Interface in Issue 4. Derived from the MSE working  
18793       draft.

18794 **NAME**18795 getwd — get the current working directory pathname (**LEGACY**) |18796 **SYNOPSIS**

18797 XSI #include &lt;unistd.h&gt;

18798 char \*getwd(char \*path\_name);

18799

18800 **DESCRIPTION**

18801 The *getwd()* function shall determine an absolute pathname of the current working directory of |  
18802 the calling process, and copy a string containing that pathname into the array pointed to by the |  
18803 *path\_name* argument.

18804 If the length of the pathname of the current working directory is greater than ({PATH\_MAX}+1) |  
18805 including the null byte, *getwd()* shall fail and return a null pointer.

18806 **RETURN VALUE**

18807 Upon successful completion, a pointer to the string containing the absolute pathname of the |  
18808 current working directory shall be returned. Otherwise, *getwd()* shall return a null pointer and |  
18809 the contents of the array pointed to by *path\_name* are undefined.

18810 **ERRORS**

18811 No errors are defined.

18812 **EXAMPLES**

18813 None.

18814 **APPLICATION USAGE**

18815 For applications portability, the *getcwd()* function should be used to determine the current  
18816 working directory instead of *getwd()*.

18817 **RATIONALE**

18818 Since the user cannot specify the length of the buffer passed to *getwd()*, use of this function is |  
18819 discouraged. The length of a pathname described in {PATH\_MAX} is file system-dependent and |  
18820 may vary from one mount point to another, or might even be unlimited. It is possible to  
18821 overflow this buffer in such a way as to cause applications to fail, or possible system security  
18822 violations.

18823 It is recommended that the *getcwd()* function should be used to determine the current working  
18824 directory.

18825 **FUTURE DIRECTIONS**

18826 This function may be withdrawn in a future version.

18827 **SEE ALSO**18828 *getcwd()*, the Base Definitions volume of IEEE Std 1003.1-200x, <unistd.h>18829 **CHANGE HISTORY**

18830 First released in Issue 4, Version 2.

18831 **Issue 5**

18832 Moved from X/OPEN UNIX extension to BASE.

18833 **Issue 6**

18834 This function is marked LEGACY.

## 18835 NAME

18836 glob, globfree — generate pathnames matching a pattern |

## 18837 SYNOPSIS

18838 #include &lt;glob.h&gt;

18839 int glob(const char \*restrict *pattern*, int *flags*,18840 int (\**errfunc*)(const char \**epath*, int *eerrno*),18841 glob\_t \*restrict *pglob*);18842 void globfree(glob\_t \**pglob*);

## 18843 DESCRIPTION

18844 The *glob()* function is a pathname generator that shall implement the rules defined in the Shell |

18845 and Utilities volume of IEEE Std 1003.1-200x, Section 2.13, Pattern Matching Notation, with |

18846 optional support for rule 3 in the Shell and Utilities volume of IEEE Std 1003.1-200x, Section |

18847 2.13.3, Patterns Used for Filename Expansion. |

18848 The structure type **glob\_t** is defined in <**glob.h**> and includes at least the following members: |

18849

18850

Member Type	Member Name	Description
size_t	<i>gl_pathc</i>	Count of paths matched by <i>pattern</i> .
char **	<i>gl_pathv</i>	Pointer to a list of matched pathnames.
size_t	<i>gl_offs</i>	Slots to reserve at the beginning of <i>gl_pathv</i> .

18851

18852

18853

18854 The argument *pattern* is a pointer to a pathname pattern to be expanded. The *glob()* function |

18855 shall match all accessible pathnames against this pattern and develop a list of all pathnames that |

18856 match. In order to have access to a pathname, *glob()* requires search permission on every |

18857 component of a path except the last, and read permission on each directory of any filename |

18858 component of *pattern* that contains any of the following special characters: '\*', '?', and '['.18859 The *glob()* function shall store the number of matched pathnames into *pglob->gl\_pathc* and a |18860 pointer to a list of pointers to path names into *pglob->gl\_pathv*. The pathnames shall be in sort |18861 order as defined by the current setting of the *LC\_COLLATE* category; see the Base Definitions |18862 volume of IEEE Std 1003.1-200x, Section 7.3.2, *LC\_COLLATE*. The first pointer after the last |

18863 pathname shall be a null pointer. If the pattern does not match any pathnames, the returned |

18864 number of matched paths is set to 0, and the contents of *pglob->gl\_pathv* are implementation- |

18865 defined.

18866 It is the caller's responsibility to create the structure pointed to by *pglob*. The *glob()* function shall |18867 allocate other space as needed, including the memory pointed to by *gl\_pathv*. The *globfree()* |18868 function shall free any space associated with *pglob* from a previous call to *glob()*.18869 The *flags* argument is used to control the behavior of *glob()*. The value of *flags* is a bitwise- |18870 inclusive OR of zero or more of the following constants, which are defined in <**glob.h**>: |18871 **GLOBAL\_APPEND** Append pathnames generated to the ones from a previous call to *glob()*. |18872 **GLOBAL\_DOOFFS** Make use of *pglob->gl\_offs*. If this flag is set, *pglob->gl\_offs* is used to |18873 specify how many null pointers to add to the beginning of *pglob->gl\_pathv*. In other words, |18874 *pglob->gl\_pathv* shall point to *pglob->gl\_offs* null |18875 pointers, followed by *pglob->gl\_pathc* pathname pointers, followed by a |

18876 null pointer.

18877 **GLOBAL\_ERR** Cause *glob()* to return when it encounters a directory that it cannot open |18878 or read. Ordinarily, *glob()* continues to find matches.

18879	GLOB_MARK	Each pathname that is a directory that matches <i>pattern</i> shall have a slash	
18880		appended.	
18881	GLOB_NOCHECK	Supports rule 3 in the Shell and Utilities volume of IEEE Std 1003.1-200x,	
18882		Section 2.13.3, Patterns Used for Filename Expansion. If <i>pattern</i> does not	
18883		match any pathname, then <i>glob()</i> shall return a list consisting of only	
18884		<i>pattern</i> , and the number of matched pathnames is 1.	
18885	GLOB_NOESCAPE	Disable backslash escaping.	
18886	GLOB_NOSORT	Ordinarily, <i>glob()</i> sorts the matching pathnames according to the current	
18887		setting of the <i>LC_COLLATE</i> category, see the Base Definitions volume of	
18888		IEEE Std 1003.1-200x, Section 7.3.2, <i>LC_COLLATE</i> . When this flag is	
18889		used, the order of path names returned is unspecified.	
18890	The GLOB_APPEND flag can be used to append a new set of pathnames to those found in a		
18891	previous call to <i>glob()</i> . The following rules apply to applications when two or more calls to		
18892	<i>glob()</i> are made with the same value of <i>pglob</i> and without intervening calls to <i>globfree()</i> :		
18893	1.	The first such call shall not set GLOB_APPEND. All subsequent calls shall set it.	
18894	2.	All the calls shall set GLOB_DOOFFS, or all shall not set it.	
18895	3.	After the second call, <i>pglob-&gt;gl_pathv</i> points to a list containing the following:	
18896	a.	Zero or more null pointers, as specified by GLOB_DOOFFS and <i>pglob-&gt;gl_offs</i> .	
18897	b.	Pointers to the pathnames that were in the <i>pglob-&gt;gl_pathv</i> list before the call, in the	
18898		same order as before.	
18899	c.	Pointers to the new pathnames generated by the second call, in the specified order.	
18900	4.	The count returned in <i>pglob-&gt;gl_pathc</i> shall be the total number of pathnames from the two	
18901		calls.	
18902	5.	The application can change any of the fields after a call to <i>glob()</i> . If it does, the application	
18903		shall reset them to the original value before a subsequent call, using the same <i>pglob</i> value,	
18904		to <i>globfree()</i> or <i>glob()</i> with the GLOB_APPEND flag.	
18905	If, during the search, a directory is encountered that cannot be opened or read and <i>errfunc</i> is not		
18906	a null pointer, <i>glob()</i> calls ( <i>*errfunc()</i> ) with two arguments:		
18907	1.	The <i>epath</i> argument is a pointer to the path that failed.	
18908	2.	The <i>errno</i> argument is the value of <i>errno</i> from the failure, as set by <i>opendir()</i> , <i>readdir()</i> , or	
18909		<i>stat()</i> . (Other values may be used to report other errors not explicitly documented for	
18910		those functions.)	
18911	If ( <i>*errfunc()</i> ) is called and returns non-zero, or if the GLOB_ERR flag is set in <i>flags</i> , <i>glob()</i> shall		
18912	stop the scan and return GLOB_ABORTED after setting <i>gl_pathc</i> and <i>gl_pathv</i> in <i>pglob</i> to reflect		
18913	the paths already scanned. If GLOB_ERR is not set and either <i>errfunc</i> is a null pointer or		
18914	<i>*errfunc()</i> returns 0, the error shall be ignored.		
18915	The <i>glob()</i> function shall not fail because of large files.		
18916	<b>RETURN VALUE</b>		
18917	Upon successful completion, <i>glob()</i> shall return 0. The argument <i>pglob-&gt;gl_pathc</i> shall return the		
18918	number of matched pathnames and the argument <i>pglob-&gt;gl_pathv</i> shall contain a pointer to a		
18919	null-terminated list of matched and sorted pathnames. However, if <i>pglob-&gt;gl_pathc</i> is 0, the		
18920	content of <i>pglob-&gt;gl_pathv</i> is undefined.		

18921 The *globfree()* function shall not return a value.

18922 If *glob()* terminates due to an error, it shall return one of the non-zero constants defined in  
18923 <glob.h>. The arguments *pglob->gl\_pathc* and *pglob->gl\_pathv* are still set as defined above.

**18924 ERRORS**

18925 The *glob()* function shall fail and return the corresponding value if:

18926	GLOB_ABORTED	The scan was stopped because GLOB_ERR was set or (*errfunc())
18927		returned non-zero.
18928	GLOB_NOMATCH	The pattern does not match any existing pathname, and
18929		GLOB_NOCHECK was not set in flags.
18930	GLOB_NOSPACE	An attempt to allocate memory failed.

**18931 EXAMPLES**

18932 One use of the GLOB\_DOOFFS flag is by applications that build an argument list for use with  
18933 *execv()*, *execve()*, or *execvp()*. Suppose, for example, that an application wants to do the  
18934 equivalent of:

```
18935 ls -l *.c
```

18936 but for some reason:

```
18937 system("ls -l *.c")
```

18938 is not acceptable. The application could obtain approximately the same result using the  
18939 sequence:

```
18940 globbuf.gl_offs = 2;
18941 glob("*.c", GLOB_DOOFFS, NULL, &globbuf);
18942 globbuf.gl_pathv[0] = "ls";
18943 globbuf.gl_pathv[1] = "-l";
18944 execvp("ls", &globbuf.gl_pathv[0]);
```

18945 Using the same example:

```
18946 ls -l *.c *.h
```

18947 could be approximately simulated using GLOB\_APPEND as follows:

```
18948 globbuf.gl_offs = 2;
18949 glob("*.c", GLOB_DOOFFS, NULL, &globbuf);
18950 glob("*.h", GLOB_DOOFFS|GLOB_APPEND, NULL, &globbuf);
18951 ...
```

**18952 APPLICATION USAGE**

18953 This function is not provided for the purpose of enabling utilities to perform pathname  
18954 expansion on their arguments, as this operation is performed by the shell, and utilities are  
18955 explicitly not expected to redo this. Instead, it is provided for applications that need to do  
18956 pathname expansion on strings obtained from other sources, such as a pattern typed by a user or  
18957 read from a file.

18958 If a utility needs to see if a pathname matches a given pattern, it can use *fnmatch()*.

18959 Note that *gl\_pathc* and *gl\_pathv* have meaning even if *glob()* fails. This allows *glob()* to report  
18960 partial results in the event of an error. However, if *gl\_pathc* is 0, *gl\_pathv* is unspecified even if  
18961 *glob()* did not return an error.

18962 The GLOB\_NOCHECK option could be used when an application wants to expand a pathname  
18963 if wildcards are specified, but wants to treat the pattern as just a string otherwise. The *sh* utility

18964 might use this for option-arguments, for example.

18965 The new pathnames generated by a subsequent call with GLOB\_APPEND are not sorted |  
18966 together with the previous pathnames. This mirrors the way that the shell handles pathname |  
18967 expansion when multiple expansions are done on a command line. |

18968 Applications that need tilde and parameter expansion should use *wordexp()*.

#### 18969 RATIONALE

18970 It was claimed that the GLOB\_DOOFFS flag is unnecessary because it could be simulated using:

```
18971 new = (char **)malloc((n + pglob->gl_pathc + 1)
18972     * sizeof(char *));
18973 (void) memcpy(new+n, pglob->gl_pathv,
18974     pglob->gl_pathc * sizeof(char *));
18975 (void) memset(new, 0, n * sizeof(char *));
18976 free(pglob->gl_pathv);
18977 pglob->gl_pathv = new;
```

18978 However, this assumes that the memory pointed to by *gl\_pathv* is a block that was separately  
18979 created using *malloc()*. This is not necessarily the case. An application should make no  
18980 assumptions about how the memory referenced by fields in *pglob* was allocated. It might have  
18981 been obtained from *malloc()* in a large chunk and then carved up within *glob()*, or it might have  
18982 been created using a different memory allocator. It is not the intent of the standard developers to  
18983 specify or imply how the memory used by *glob()* is managed.

18984 The GLOB\_APPEND flag would be used when an application wants to expand several different  
18985 patterns into a single list.

#### 18986 FUTURE DIRECTIONS

18987 None.

#### 18988 SEE ALSO

18989 *exec*, *fnmatch()*, *opendir()*, *readdir()*, *stat()*, *wordexp()*, the Base Definitions volume of  
18990 IEEE Std 1003.1-200x, <**glob.h**>, the Shell and Utilities volume of IEEE Std 1003.1-200x

#### 18991 CHANGE HISTORY

18992 First released in Issue 4. Derived from the ISO POSIX-2 standard.

#### 18993 Issue 5

18994 Moved from POSIX2 C-language Binding to BASE.

#### 18995 Issue 6

18996 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

18997 The **restrict** keyword is added to the *glob()* prototype for alignment with the ISO/IEC 9899:1999  
18998 standard.

18999 **NAME**

19000 gmtime, gmtime\_r — convert a time value to a broken-down UTC time

19001 **SYNOPSIS**

19002 #include <time.h>

19003 struct tm \*gmtime(const time\_t \*timer);

19004 TSF struct tm \*gmtime\_r(const time\_t \*restrict timer,

19005 struct tm \*restrict result);

19006

19007 **DESCRIPTION**

19008 CX For *gmtime()*: The functionality described on this reference page is aligned with the ISO C  
 19009 standard. Any conflict between the requirements described here and the ISO C standard is  
 19010 unintentional. This volume of IEEE Std 1003.1-200x defers to the ISO C standard.

19011 The *gmtime()* function shall convert the time in seconds since the Epoch pointed to by *timer* into  
 19012 a broken-down time, expressed as Coordinated Universal Time (UTC).

19013 CX The relationship between a time in seconds since the Epoch used as an argument to *gmtime()* |  
 19014 and the **tm** structure (defined in the <time.h> header) is that the result shall be as specified in the |  
 19015 expression given in the definition of seconds since the Epoch (see the Base Definitions volume of |  
 19016 IEEE Std 1003.1-200x, Section 4.14, Seconds Since the Epoch), where the names in the structure |  
 19017 and in the expression correspond. |

19018 TSF The same relationship shall apply for *gmtime\_r()*. |

19019 CX The *gmtime()* function need not be reentrant. A function that is not required to be reentrant is not |  
 19020 required to be thread-safe. |

19021 The *asctime()*, *ctime()*, *gmtime()*, and *localtime()* functions shall return values in one of two static |  
 19022 objects: a broken-down time structure and an array of type **char**. Execution of any of the |  
 19023 functions may overwrite the information returned in either of these objects by any of the other |  
 19024 functions. |

19025 TSF The *gmtime\_r()* function shall convert the time in seconds since the Epoch pointed to by *timer* |  
 19026 into a broken-down time expressed as Coordinated Universal Time (UTC). The broken-down |  
 19027 time is stored in the structure referred to by *result*. The *gmtime\_r()* function shall also return the |  
 19028 address of the same structure. |

19029 **RETURN VALUE**

19030 The *gmtime()* function shall return a pointer to a **struct tm**.

19031 TSF Upon successful completion, *gmtime\_r()* shall return the address of the structure pointed to by |  
 19032 the argument *result*. If an error is detected, or UTC is not available, *gmtime\_r()* shall return a null |  
 19033 pointer. |

19034 **ERRORS**

19035 No errors are defined.

19036 **EXAMPLES**

19037 None.

19038 **APPLICATION USAGE**

19039 The *gmtime\_r()* function is thread-safe and returns values in a user-supplied buffer instead of  
19040 possibly using a static data area that may be overwritten by each call.

19041 **RATIONALE**

19042 None.

19043 **FUTURE DIRECTIONS**

19044 None.

19045 **SEE ALSO**

19046 *asctime()*, *clock()*, *ctime()*, *difftime()*, *localtime()*, *mktime()*, *strftime()*, *strptime()*, *time()*, *utime()*,  
19047 the Base Definitions volume of IEEE Std 1003.1-200x, <**time.h**>

19048 **CHANGE HISTORY**

19049 First released in Issue 1. Derived from Issue 1 of the SVID.

19050 **Issue 5**

19051 A note indicating that the *gmtime()* function need not be reentrant is added to the  
19052 DESCRIPTION.

19053 The *gmtime\_r()* function is included for alignment with the POSIX Threads Extension.19054 **Issue 6**19055 The *gmtime\_r()* function is marked as part of the Thread-Safe Functions option.

19056 Extensions beyond the ISO C standard are now marked.

19057 The APPLICATION USAGE section is updated to include a note on the thread-safe function and  
19058 its avoidance of possibly using a static data area.

19059 The **restrict** keyword is added to the *gmtime\_r()* prototype for alignment with the  
19060 ISO/IEC 9899:1999 standard.

19061 **NAME**

19062 grantpt — grant access to the slave pseudo-terminal device

19063 **SYNOPSIS**

19064 XSI #include &lt;stdlib.h&gt;

19065 int grantpt(int *fildev*);

19066

19067 **DESCRIPTION**

19068 The *grantpt()* function shall change the mode and ownership of the slave pseudo-terminal  
19069 device associated with its master pseudo-terminal counterpart. The *fildev* argument is a file  
19070 descriptor that refers to a master pseudo-terminal device. The user ID of the slave shall be set to  
19071 the real UID of the calling process and the group ID shall be set to an unspecified group ID. The  
19072 permission mode of the slave pseudo-terminal shall be set to readable and writable by the  
19073 owner, and writable by the group.

19074 The behavior of the *grantpt()* function is unspecified if the application has installed a signal  
19075 handler to catch SIGCHLD signals.

19076 **RETURN VALUE**

19077 Upon successful completion, *grantpt()* shall return 0; otherwise, it shall return -1 and set *errno* to  
19078 indicate the error.

19079 **ERRORS**19080 The *grantpt()* function may fail if:19081 [EBADF] The *fildev* argument is not a valid open file descriptor.19082 [EINVAL] The *fildev* argument is not associated with a master pseudo-terminal device.

19083 [EACCES] The corresponding slave pseudo-terminal device could not be accessed.

19084 **EXAMPLES**

19085 None.

19086 **APPLICATION USAGE**

19087 None.

19088 **RATIONALE**

19089 None.

19090 **FUTURE DIRECTIONS**

19091 None.

19092 **SEE ALSO**19093 *open()*, *ptsname()*, *unlockpt()*, the Base Definitions volume of IEEE Std 1003.1-200x, <stdlib.h>19094 **CHANGE HISTORY**

19095 First released in Issue 4, Version 2.

19096 **Issue 5**

19097 Moved from X/OPEN UNIX extension to BASE.

19098 The last paragraph of the DESCRIPTION is moved from the APPLICATION USAGE section in  
19099 previous issues.

19100 **NAME**

19101 h\_errno — error return value for network database operations

19102 **SYNOPSIS**

```
19103 OB #include <netdb.h>
```

19104

19105 **DESCRIPTION**

19106 Note that this method of returning errors is used only in connection with obsolescent functions.

19107 The <netdb.h> header provides a declaration of *h\_errno* as a modifiable *l*-value of type **int**.

19108 It is unspecified whether *h\_errno* is a macro or an identifier declared with external linkage. If a  
19109 macro definition is suppressed in order to access an actual object, or a program defines an  
19110 identifier with the name *h\_errno*, the behavior is undefined.

19111 **RETURN VALUE**

19112 None.

19113 **ERRORS**

19114 No errors are defined.

19115 **EXAMPLES**

19116 None.

19117 **APPLICATION USAGE**

19118 Applications should obtain the definition of *h\_errno* by the inclusion of the <netdb.h> header.

19119 **RATIONALE**

19120 None.

19121 **FUTURE DIRECTIONS**

19122 *h\_errno* may be withdrawn in a future version.

19123 **SEE ALSO**

19124 *endhostent()*, *errno*, the Base Definitions volume of IEEE Std 1003.1-200x, <netdb.h>

19125 **CHANGE HISTORY**

19126 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

19127 **NAME**

19128       hcreate, hdestroy, hsearch — manage hash search table

19129 **SYNOPSIS**

```

19130 xSI      #include <search.h>
19131          int hcreate(size_t nel);
19132          void hdestroy(void);
19133          ENTRY *hsearch(ENTRY item, ACTION action);
19134

```

19135 **DESCRIPTION**19136       The *hcreate()*, *hdestroy()*, and *hsearch()* functions shall manage hash search tables. |

19137       The *hcreate()* function shall allocate sufficient space for the table, and the application shall |  
 19138       ensure it is called before *hsearch()* is used. The *nel* argument is an estimate of the maximum |  
 19139       number of entries that the table shall contain. This number may be adjusted upward by the |  
 19140       algorithm in order to obtain certain mathematically favorable circumstances.

19141       The *hdestroy()* function shall dispose of the search table, and may be followed by another call to |  
 19142       *hcreate()*. After the call to *hdestroy()*, the data can no longer be considered accessible.

19143       The *hsearch()* function is a hash-table search routine. It shall return a pointer into a hash table |  
 19144       indicating the location at which an entry can be found. The *item* argument is a structure of type |  
 19145       **ENTRY** (defined in the *<search.h>* header) containing two pointers: *item.key* points to the |  
 19146       comparison key (a **char** \*), and *item.data* (a **void** \*) points to any other data to be associated with |  
 19147       that key. The comparison function used by *hsearch()* is *strcmp()*. The *action* argument is a |  
 19148       member of an enumeration type **ACTION** indicating the disposition of the entry if it cannot be |  
 19149       found in the table. **ENTER** indicates that the item should be inserted in the table at an |  
 19150       appropriate point. **FIND** indicates that no entry should be made. Unsuccessful resolution is |  
 19151       indicated by the return of a null pointer.

19152       These functions need not be reentrant. A function that is not required to be reentrant is not |  
 19153       required to be thread-safe.

19154 **RETURN VALUE**

19155       The *hcreate()* function shall return 0 if it cannot allocate sufficient space for the table; otherwise, |  
 19156       it shall return non-zero.

19157       The *hdestroy()* function shall not return a value.

19158       The *hsearch()* function shall return a null pointer if either the action is **FIND** and the item could |  
 19159       not be found or the action is **ENTER** and the table is full.

19160 **ERRORS**

19161       The *hcreate()* and *hsearch()* functions may fail if:

19162       [ENOMEM]       Insufficient storage space is available.

19163 **EXAMPLES**

19164       The following example reads in strings followed by two numbers and stores them in a hash |  
 19165       table, discarding duplicates. It then reads in strings and finds the matching entry in the hash |  
 19166       table and prints it out.

```

19167          #include <stdio.h>
19168          #include <search.h>
19169          #include <string.h>
19170          struct info {          /* This is the info stored in the table */
19171              int age, room;    /* other than the key. */

```

```

19172     };
19173     #define NUM_EMPL    5000    /* # of elements in search table. */
19174     int main(void)
19175     {
19176         char string_space[NUM_EMPL*20];    /* Space to store strings. */
19177         struct info info_space[NUM_EMPL]; /* Space to store employee info. */
19178         char *str_ptr = string_space;     /* Next space in string_space. */
19179         struct info *info_ptr = info_space;
19180                                         /* Next space in info_space. */
19181         ENTRY item;
19182         ENTRY *found_item; /* Name to look for in table. */
19183         char name_to_find[30];
19184
19184         int i = 0;
19185
19185         /* Create table; no error checking is performed. */
19186         (void) hcreate(NUM_EMPL);
19187         while (scanf("%s%d%d", str_ptr, &info_ptr->age,
19188                    &info_ptr->room) != EOF && i++ < NUM_EMPL) {
19189
19189             /* Put information in structure, and structure in item. */
19190             item.key = str_ptr;
19191             item.data = info_ptr;
19192             str_ptr += strlen(str_ptr) + 1;
19193             info_ptr++;
19194
19194             /* Put item into table. */
19195             (void) hsearch(item, ENTER);
19196         }
19197
19197         /* Access table. */
19198         item.key = name_to_find;
19199         while (scanf("%s", item.key) != EOF) {
19200             if ((found_item = hsearch(item, FIND)) != NULL) {
19201
19201                 /* If item is in the table. */
19202                 (void)printf("found %s, age = %d, room = %d\n",
19203                             found_item->key,
19204                             ((struct info *)found_item->data)->age,
19205                             ((struct info *)found_item->data)->room);
19206             } else
19207                 (void)printf("no such employee %s\n", name_to_find);
19208         }
19209         return 0;
19210     }

```

#### 19211 APPLICATION USAGE

19212 The *hcreate()* and *hsearch()* functions may use *malloc()* to allocate space.

#### 19213 RATIONALE

19214 None.

19215 **FUTURE DIRECTIONS**

19216       None.

19217 **SEE ALSO**19218       *bsearch()*, *lsearch()*, *malloc()*, *strcmp()*, *tsearch()*, the Base Definitions volume of  
19219       IEEE Std 1003.1-200x, <**search.h**>19220 **CHANGE HISTORY**

19221       First released in Issue 1. Derived from Issue 1 of the SVID.

19222 **Issue 6**

19223       The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

19224       A note indicating that this function need not be reentrant is added to the DESCRIPTION.

19225 **NAME**

19226       hdestroy — manage hash search table

19227 **SYNOPSIS**

19228 XSI     #include <search.h>

19229       void hdestroy(void);

19230

19231 **DESCRIPTION**

19232       Refer to *hcreate()*.

19233 **NAME**

19234 hsearch — manage hash search table

19235 **SYNOPSIS**

19236 xSI #include &lt;search.h&gt;

19237 ENTRY \*hsearch(ENTRY *item*, ACTION *action*);

19238

19239 **DESCRIPTION**19240 Refer to *hcreate()*.

19241 **NAME**

19242 htonl, htons, ntohl, ntohs — convert values between host and network byte order

19243 **SYNOPSIS**

19244 #include <arpa/inet.h>

19245 uint32\_t htonl(uint32\_t *hostlong*);

19246 uint16\_t htons(uint16\_t *hostshort*);

19247 uint32\_t ntohl(uint32\_t *netlong*);

19248 uint16\_t ntohs(uint16\_t *netshort*);

19249 **DESCRIPTION**

19250 These functions shall convert 16-bit and 32-bit quantities between network byte order and host  
19251 byte order.

19252 On some implementations, these functions are defined as macros.

19253 The **uint32\_t** and **uint16\_t** types are defined in <inttypes.h>.

19254 **RETURN VALUE**

19255 The *htonl()* and *htons()* functions shall return the argument value converted from host to  
19256 network byte order.

19257 The *ntohl()* and *ntohs()* functions shall return the argument value converted from network to  
19258 host byte order.

19259 **ERRORS**

19260 No errors are defined.

19261 **EXAMPLES**

19262 None.

19263 **APPLICATION USAGE**

19264 These functions are most often used in conjunction with IPv4 addresses and ports as returned by  
19265 *gethostent()* and *getservent()*.

19266 **RATIONALE**

19267 None.

19268 **FUTURE DIRECTIONS**

19269 None.

19270 **SEE ALSO**

19271 *endhostent()*, *endservent()*, the Base Definitions volume of IEEE Std 1003.1-200x, <inttypes.h>,  
19272 <arpa/inet.h>

19273 **CHANGE HISTORY**

19274 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

19275 **NAME**

19276       htons — convert values between host and network byte order

19277 **SYNOPSIS**

19278       #include <arpa/inet.h>

19279       uint16\_t htons(uint16\_t *hostshort*);

19280 **DESCRIPTION**

19281       Refer to *htonl()*.

19282 **NAME**

19283 hypot, hypotf, hypotl — Euclidean distance function

19284 **SYNOPSIS**

19285 #include &lt;math.h&gt;

19286 double hypot(double x, double y);

19287 float hypotf(float x, float y);

19288 long double hypotl(long double x, long double y);

19289 **DESCRIPTION**

19290 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 19291 conflict between the requirements described here and the ISO C standard is unintentional. This  
 19292 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

19293 These functions shall compute the value of the square root of  $x^2+y^2$  without undue overflow or  
 19294 underflow.

19295 An application wishing to check for error situations should set *errno* to zero and call  
 19296 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 19297 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 19298 zero, an error has occurred.

19299 **RETURN VALUE**

19300 Upon successful completion, these functions shall return the length of the hypotenuse of a  
 19301 right-angled triangle with sides of length *x* and *y*.

19302 If the correct value would cause overflow, a range error shall occur and *hypot()*, *hypotf()*, and  
 19303 *hypotl()* shall return the value of the macro HUGE\_VAL, HUGE\_VALF, and HUGE\_VALL,  
 19304 respectively.

19305 **MX** If *x* or *y* is  $\pm\text{Inf}$ ,  $+\text{Inf}$  shall be returned (even if one of *x* or *y* is NaN).

19306 If *x* or *y* is NaN, and the other is not  $\pm\text{Inf}$ , a NaN shall be returned.

19307 If both arguments are subnormal and the correct result is subnormal, a range error may occur  
 19308 and the correct result is returned.

19309 **ERRORS**

19310 These functions shall fail if:

19311 **Range Error** The result overflows.

19312 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero, |  
 19313 then *errno* shall be set to [ERANGE]. If the integer expression |  
 19314 (math\_errhandling & MATH\_ERREXCEPT) is non-zero, then the overflow |  
 19315 floating-point exception shall be raised. |

19316 These functions may fail if:

19317 **MX** **Range Error** The result underflows.

19318 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero, |  
 19319 then *errno* shall be set to [ERANGE]. If the integer expression |  
 19320 (math\_errhandling & MATH\_ERREXCEPT) is non-zero, then the underflow |  
 19321 floating-point exception shall be raised. |

19322 **EXAMPLES**

19323 None.

19324 **APPLICATION USAGE**19325 *hypot(x,y)*, *hypot(y,x)*, and *hypot(x, -y)* are equivalent.19326 *hypot(x, ±0)* is equivalent to *fabs(x)*.19327 Underflow only happens when both *x* and *y* are subnormal and the (inexact) result is also  
19328 subnormal.19329 These functions take precautions against overflow during intermediate steps of the  
19330 computation.19331 On error, the expressions (math\_errhandling & MATH\_ERRNO) and (math\_errhandling &  
19332 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.19333 **RATIONALE**

19334 None.

19335 **FUTURE DIRECTIONS**

19336 None.

19337 **SEE ALSO**19338 *feclearexcept()*, *fetestexcept()*, *isnan()*, *sqrt()*, the Base Definitions volume of IEEE Std 1003.1-200x, |  
19339 Section 4.18, Treatment of Error Conditions for Mathematical Functions, <math.h> |19340 **CHANGE HISTORY**

19341 First released in Issue 1. Derived from Issue 1 of the SVID.

19342 **Issue 5**19343 The DESCRIPTION is updated to indicate how an application should check for an error. This  
19344 text was previously published in the APPLICATION USAGE section.19345 **Issue 6**19346 The *hypot()* function is no longer marked as an extension.19347 The *hypotf()* and *hypotl()* functions are added for alignment with the ISO/IEC 9899:1999  
19348 standard.19349 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
19350 revised to align with the ISO/IEC 9899:1999 standard.19351 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
19352 marked.

## 19353 NAME

19354 iconv — codeset conversion function

## 19355 SYNOPSIS

19356 XSI `#include <iconv.h>`

```
19357     size_t iconv(iconv_t cd, char **restrict inbuf,
19358                size_t *restrict inbytesleft, char **restrict outbuf,
19359                size_t *restrict outbytesleft);
```

19360

## 19361 DESCRIPTION

19362 The *iconv()* function shall convert the sequence of characters from one codeset, in the array  
 19363 specified by *inbuf*, into a sequence of corresponding characters in another codeset, in the array  
 19364 specified by *outbuf*. The codesets are those specified in the *iconv\_open()* call that returned the  
 19365 conversion descriptor, *cd*. The *inbuf* argument points to a variable that points to the first  
 19366 character in the input buffer and *inbytesleft* indicates the number of bytes to the end of the buffer  
 19367 to be converted. The *outbuf* argument points to a variable that points to the first available byte in  
 19368 the output buffer and *outbytesleft* indicates the number of the available bytes to the end of the  
 19369 buffer.

19370 For state-dependent encodings, the conversion descriptor *cd* is placed into its initial shift state by  
 19371 a call for which *inbuf* is a null pointer, or for which *inbuf* points to a null pointer. When *iconv()* is  
 19372 called in this way, and if *outbuf* is not a null pointer or a pointer to a null pointer, and *outbytesleft*  
 19373 points to a positive value, *iconv()* shall place, into the output buffer, the byte sequence to change  
 19374 the output buffer to its initial shift state. If the output buffer is not large enough to hold the  
 19375 entire reset sequence, *iconv()* shall fail and set *errno* to [E2BIG]. Subsequent calls with *inbuf* as  
 19376 other than a null pointer or a pointer to a null pointer cause the conversion to take place from  
 19377 the current state of the conversion descriptor.

19378 If a sequence of input bytes does not form a valid character in the specified codeset, conversion |  
 19379 shall stop after the previous successfully converted character. If the input buffer ends with an |  
 19380 incomplete character or shift sequence, conversion shall stop after the previous successfully |  
 19381 converted bytes. If the output buffer is not large enough to hold the entire converted input, |  
 19382 conversion shall stop just prior to the input bytes that would cause the output buffer to |  
 19383 overflow. The variable pointed to by *inbuf* shall be updated to point to the byte following the last |  
 19384 byte successfully used in the conversion. The value pointed to by *inbytesleft* shall be |  
 19385 decremented to reflect the number of bytes still not converted in the input buffer. The variable |  
 19386 pointed to by *outbuf* shall be updated to point to the byte following the last byte of converted |  
 19387 output data. The value pointed to by *outbytesleft* shall be decremented to reflect the number of |  
 19388 bytes still available in the output buffer. For state-dependent encodings, the conversion |  
 19389 descriptor shall be updated to reflect the shift state in effect at the end of the last successfully |  
 19390 converted byte sequence.

19391 If *iconv()* encounters a character in the input buffer that is valid, but for which an identical |  
 19392 character does not exist in the target codeset, *iconv()* shall perform an implementation-defined |  
 19393 conversion on this character. |

## 19394 RETURN VALUE

19395 The *iconv()* function shall update the variables pointed to by the arguments to reflect the extent  
 19396 of the conversion and return the number of non-identical conversions performed. If the entire  
 19397 string in the input buffer is converted, the value pointed to by *inbytesleft* shall be 0. If the input  
 19398 conversion is stopped due to any conditions mentioned above, the value pointed to by *inbytesleft*  
 19399 shall be non-zero and *errno* shall be set to indicate the condition. If an error occurs *iconv()* shall  
 19400 return (*size\_t*)-1 and set *errno* to indicate the error.

19401 **ERRORS**

19402 The *iconv()* function shall fail if:

19403 [EILSEQ] Input conversion stopped due to an input byte that does not belong to the  
19404 input codeset.

19405 [E2BIG] Input conversion stopped due to lack of space in the output buffer.

19406 [EINVAL] Input conversion stopped due to an incomplete character or shift sequence at  
19407 the end of the input buffer.

19408 The *iconv()* function may fail if:

19409 [EBADF] The *cd* argument is not a valid open conversion descriptor.

19410 **EXAMPLES**

19411 None.

19412 **APPLICATION USAGE**

19413 The *inbuf* argument indirectly points to the memory area which contains the conversion input  
19414 data. The *outbuf* argument indirectly points to the memory area which is to contain the result of  
19415 the conversion. The objects indirectly pointed to by *inbuf* and *outbuf* are not restricted to  
19416 containing data that is directly representable in the ISO C standard language **char** data type. The  
19417 type of *inbuf* and *outbuf*, **char \*\***, does not imply that the objects pointed to are interpreted as  
19418 null-terminated C strings or arrays of characters. Any interpretation of a byte sequence that  
19419 represents a character in a given character set encoding scheme is done internally within the  
19420 codeset converters. For example, the area pointed to indirectly by *inbuf* and/or *outbuf* can  
19421 contain all zero octets that are not interpreted as string terminators but as coded character data  
19422 according to the respective codeset encoding scheme. The type of the data (**char**, **short**, **long**, and  
19423 so on) read or stored in the objects is not specified, but may be inferred for both the input and  
19424 output data by the converters determined by the *fromcode* and *toctype* arguments of *iconv\_open()*.

19425 Regardless of the data type inferred by the converter, the size of the remaining space in both  
19426 input and output objects (the *inbytesleft* and *outbytesleft* arguments) is always measured in bytes.

19427 For implementations that support the conversion of state-dependent encodings, the conversion  
19428 descriptor must be able to accurately reflect the shift-state in effect at the end of the last  
19429 successful conversion. It is not required that the conversion descriptor itself be updated, which  
19430 would require it to be a pointer type. Thus, implementations are free to implement the  
19431 descriptor as a handle (other than a pointer type) by which the conversion information can be  
19432 accessed and updated.

19433 **RATIONALE**

19434 None.

19435 **FUTURE DIRECTIONS**

19436 None.

19437 **SEE ALSO**

19438 *iconv\_open()*, *iconv\_close()*, the Base Definitions volume of IEEE Std 1003.1-200x, <**iconv.h**>

19439 **CHANGE HISTORY**

19440 First released in Issue 4. Derived from the HP-UX Manual.

19441 **Issue 6**

19442 The SYNOPSIS has been corrected to align with the <**iconv.h**> reference page.

19443 The **restrict** keyword is added to the *iconv()* prototype for alignment with the  
19444 ISO/IEC 9899:1999 standard.

19445 **NAME**

19446 iconv\_close — codeset conversion deallocation function

19447 **SYNOPSIS**19448 XSI `#include <iconv.h>`19449 `int iconv_close(iconv_t cd);`

19450

19451 **DESCRIPTION**19452 The *iconv\_close()* function shall deallocate the conversion descriptor *cd* and all other associated |  
19453 resources allocated by *iconv\_open()*.19454 If a file descriptor is used to implement the type **iconv\_t**, that file descriptor shall be closed. |19455 **RETURN VALUE**19456 Upon successful completion, 0 shall be returned; otherwise, -1 shall be returned and *errno* set to |  
19457 indicate the error.19458 **ERRORS**19459 The *iconv\_close()* function may fail if:

19460 [EBADF] The conversion descriptor is invalid.

19461 **EXAMPLES**

19462 None.

19463 **APPLICATION USAGE**

19464 None.

19465 **RATIONALE**

19466 None.

19467 **FUTURE DIRECTIONS**

19468 None.

19469 **SEE ALSO**19470 *iconv()*, *iconv\_open()*, the Base Definitions volume of IEEE Std 1003.1-200x, <**iconv.h**>19471 **CHANGE HISTORY**

19472 First released in Issue 4. Derived from the HP-UX Manual.

19473 **NAME**

19474 iconv\_open — codeset conversion allocation function

19475 **SYNOPSIS**19476 XSI `#include <iconv.h>`19477 `iconv_t iconv_open(const char *tocode, const char *fromcode);`

19478

19479 **DESCRIPTION**

19480 The `iconv_open()` function shall return a conversion descriptor that describes a conversion from  
 19481 the codeset specified by the string pointed to by the `fromcode` argument to the codeset specified  
 19482 by the string pointed to by the `tocode` argument. For state-dependent encodings, the conversion  
 19483 descriptor shall be in a codeset-dependent initial shift state, ready for immediate use with  
 19484 `iconv()`.

19485 Settings of `fromcode` and `tocode` and their permitted combinations are implementation-defined.

19486 A conversion descriptor shall remain valid until it is closed by `iconv_close()` or an implicit close.

19487 If a file descriptor is used to implement conversion descriptors, the `FD_CLOEXEC` flag shall be  
 19488 set; see `<fcntl.h>`.

19489 **RETURN VALUE**

19490 Upon successful completion, `iconv_open()` shall return a conversion descriptor for use on  
 19491 subsequent calls to `iconv()`. Otherwise, `iconv_open()` shall return `(iconv_t)-1` and set `errno` to  
 19492 indicate the error.

19493 **ERRORS**

19494 The `iconv_open()` function may fail if:

19495 [EMFILE] {OPEN\_MAX} file descriptors are currently open in the calling process.

19496 [ENFILE] Too many files are currently open in the system.

19497 [ENOMEM] Insufficient storage space is available.

19498 [EINVAL] The conversion specified by `fromcode` and `tocode` is not supported by the  
 19499 implementation.

19500 **EXAMPLES**

19501 None.

19502 **APPLICATION USAGE**

19503 Some implementations of `iconv_open()` use `malloc()` to allocate space for internal buffer areas.  
 19504 The `iconv_open()` function may fail if there is insufficient storage space to accommodate these  
 19505 buffers.

19506 Conforming applications must assume that conversion descriptors are not valid after a call to  
 19507 one of the `exec` functions.

19508 **RATIONALE**

19509 None.

19510 **FUTURE DIRECTIONS**

19511 None.

19512 **SEE ALSO**

19513 `iconv()`, `iconv_close()`, the Base Definitions volume of IEEE Std 1003.1-200x, `<fcntl.h>`, `<iconv.h>`

19514 **CHANGE HISTORY**

19515 First released in Issue 4. Derived from the HP-UX Manual.

19516 **NAME**

19517 if\_freenameindex — free memory allocated by *if\_nameindex()*

19518 **SYNOPSIS**

19519 #include <net/if.h>

19520 void if\_freenameindex(struct if\_nameindex \*ptr);

19521 **DESCRIPTION**

19522 The *if\_freenameindex()* function shall free the memory allocated by *if\_nameindex()*. The *ptr*  
19523 argument shall be a pointer that was returned by *if\_nameindex()*. After *if\_freenameindex()* has  
19524 been called, the application shall not use the array of which *ptr* is the address. |

19525 **RETURN VALUE**

19526 None.

19527 **ERRORS**

19528 No errors are defined.

19529 **EXAMPLES**

19530 None.

19531 **APPLICATION USAGE**

19532 None.

19533 **RATIONALE**

19534 None.

19535 **FUTURE DIRECTIONS**

19536 None.

19537 **SEE ALSO**

19538 *getsockopt()*, *if\_indextoname()*, *if\_nameindex()*, *if\_nametoindex()*, *setsockopt()*, the Base Definitions  
19539 volume of IEEE Std 1003.1-200x, <net/if.h>

19540 **CHANGE HISTORY**

19541 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

19542 **NAME**

19543 if\_indextoname — map a network interface index to its corresponding name

19544 **SYNOPSIS**

19545 #include <net/if.h>

19546 char \*if\_indextoname(unsigned *ifindex*, char \**ifname*);

19547 **DESCRIPTION**

19548 The *if\_indextoname*() function shall map an interface index to its corresponding name.

19549 When this function is called, *ifname* shall point to a buffer of at least {IFNAMSIZ} bytes. The  
19550 function shall place in this buffer the name of the interface with index *ifindex*.

19551 **RETURN VALUE**

19552 If *ifindex* is an interface index, then the function shall return the value supplied in *ifname*, which  
19553 points to a buffer now containing the interface name. Otherwise, the function shall return a  
19554 NULL pointer and set *errno* to indicate the error.

19555 **ERRORS**

19556 The *if\_indextoname*() function shall fail if:

19557 [ENXIO] The interface does not exist.

19558 **EXAMPLES**

19559 None.

19560 **APPLICATION USAGE**

19561 None.

19562 **RATIONALE**

19563 None.

19564 **FUTURE DIRECTIONS**

19565 None.

19566 **SEE ALSO**

19567 *getsockopt*(), *if\_freenameindex*(), *if\_nameindex*(), *if\_nametoindex*(), *setsockopt*(), the Base  
19568 Definitions volume of IEEE Std 1003.1-200x, <net/if.h>

19569 **CHANGE HISTORY**

19570 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

19571 **NAME**

19572 if\_nameindex — return all network interface names and indexes

19573 **SYNOPSIS**

19574 #include <net/if.h>

19575 struct if\_nameindex \*if\_nameindex(void);

19576 **DESCRIPTION**

19577 The *if\_nameindex()* function shall return an array of *if\_nameindex* structures, one structure per  
19578 interface. The end of the array is indicated by a structure with an *if\_index* field of zero and an  
19579 *if\_name* field of NULL.

19580 Applications should call *if\_freenameindex()* to release the memory that may be dynamically  
19581 allocated by this function, after they have finished using it.

19582 **RETURN VALUE**

19583 Array of structures identifying local interfaces. A NULL pointer is returned upon an error, with  
19584 *errno* set to indicate the error.

19585 **ERRORS**

19586 The *if\_nameindex()* function may fail if:

19587 [ENOBUFS] Insufficient resources are available to complete the function.

19588 **EXAMPLES**

19589 None.

19590 **APPLICATION USAGE**

19591 None.

19592 **RATIONALE**

19593 None.

19594 **FUTURE DIRECTIONS**

19595 None.

19596 **SEE ALSO**

19597 *getsockopt()*, *if\_freenameindex()*, *if\_indextoname()*, *if\_nametoindex()*, *setsockopt()*, the Base  
19598 Definitions volume of IEEE Std 1003.1-200x, <net/if.h>

19599 **CHANGE HISTORY**

19600 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

19601 **NAME**

19602       if\_nametoindex — map a network interface name to its corresponding index

19603 **SYNOPSIS**

19604       #include <net/if.h>

19605       unsigned if\_nametoindex(const char \*ifname);

19606 **DESCRIPTION**

19607       The *if\_nametoindex()* function shall return the interface index corresponding to name *ifname*.

19608 **RETURN VALUE**

19609       The corresponding index if *ifname* is the name of an interface; otherwise, zero.

19610 **ERRORS**

19611       No errors are defined.

19612 **EXAMPLES**

19613       None.

19614 **APPLICATION USAGE**

19615       None.

19616 **RATIONALE**

19617       None.

19618 **FUTURE DIRECTIONS**

19619       None.

19620 **SEE ALSO**

19621       *getsockopt()*, *if\_freenameindex()*, *if\_indextoname()*, *if\_nameindex()*, *setsockopt()*, the Base  
19622       Definitions volume of IEEE Std 1003.1-200x, <net/if.h>

19623 **CHANGE HISTORY**

19624       First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

19625 **NAME**

19626           ilogb, ilogbf, ilogbl — return an unbiased exponent

19627 **SYNOPSIS**

```
19628           #include <math.h>
19629           int ilogb(double x);
19630           int ilogbf(float x);
19631           int ilogbl(long double x);
```

19632 **DESCRIPTION**

19633 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
 19634 conflict between the requirements described here and the ISO C standard is unintentional. This  
 19635 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

19636       These functions shall return the exponent part of their argument  $x$ . Formally, the return value is  
 19637 the integral part of  $\log_r |x|$  as a signed integral value, for non-zero  $x$ , where  $r$  is the radix of the  
 19638 machine's floating-point arithmetic, which is the value of `FLT_RADIX` defined in `<float.h>`.

19639       An application wishing to check for error situations should set `errno` to zero and call  
 19640 `feclearexcept(FE_ALL_EXCEPT)` before calling these functions. On return, if `errno` is non-zero or  
 19641 `fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW)` is non-  
 19642 zero, an error has occurred.

19643 **RETURN VALUE**

19644       Upon successful completion, these functions shall return the exponent part of  $x$  as a signed  
 19645 integer value. They are equivalent to calling the corresponding `logb()` function and casting the  
 19646 returned value to type `int`.

19647 **XSI**       If  $x$  is 0, a domain error shall occur, and the value `FP_ILOGB0` shall be returned.

19648 **XSI**       If  $x$  is  $\pm\text{Inf}$ , a domain error shall occur, and the value `{INT_MAX}` shall be returned.

19649 **XSI**       If  $x$  is a NaN, a domain error shall occur, and the value `FP_ILOGBNAN` shall be returned.

19650 **XSI**       If the correct value is greater than `{INT_MAX}`, `{INT_MAX}` shall be returned and a domain error  
 19651 shall occur.

19652       If the correct value is less than `{INT_MIN}`, `{INT_MIN}` shall be returned and a domain error  
 19653 shall occur.

19654 **ERRORS**

19655       These functions shall fail if:

19656 <b>XSI</b>	<b>Domain Error</b>	The $x$ argument is zero, NaN, or $\pm\text{Inf}$ , or the correct value is not representable as an integer.
------------------	---------------------	--

19658	If the integer expression <code>(math_errhandling &amp; MATH_ERRNO)</code> is non-zero,	
19659	then <code>errno</code> shall be set to <code>[EDOM]</code> . If the integer expression <code>(math_errhandling</code>	
19660	& <code>MATH_ERREXCEPT)</code> is non-zero, then the invalid floating-point exception	
19661	shall be raised.	

19662 **EXAMPLES**

19663           None.

19664 **APPLICATION USAGE**

19665           On error, the expressions (math\_errhandling & MATH\_ERRNO) and (math\_errhandling &  
19666           MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

19667 **RATIONALE**

19668           The errors come from taking the expected floating-point value and converting it to **int**, which is  
19669           invalid operation in IEEE Std 754-1985 (since overflow, infinity, and NaN are not representable  
19670           in a type **int**), so should be a domain error.

19671           There are no known implementations that overflow. For overflow to happen, {INT\_MAX} must  
19672           be less than  $LDBL\_MAX\_EXP * \log_2(\text{FLT\_RADIX})$  or {INT\_MIN} must be greater than  
19673            $LDBL\_MIN\_EXP * \log_2(\text{FLT\_RADIX})$  if subnormals are not supported, or {INT\_MIN} must be  
19674           greater than  $(LDBL\_MIN\_EXP - LDBL\_MANT\_DIG) * \log_2(\text{FLT\_RADIX})$  if subnormals are  
19675           supported.

19676 **FUTURE DIRECTIONS**

19677           None.

19678 **SEE ALSO**

19679           *feclearexcept()*, *fetestexcept()*, *logb()*, *scalb()*, the Base Definitions volume of IEEE Std 1003.1-200x, |  
19680           Section 4.18, Treatment of Error Conditions for Mathematical Functions, <float.h>, <math.h> |

19681 **CHANGE HISTORY**

19682           First released in Issue 4, Version 2.

19683 **Issue 5**

19684           Moved from X/OPEN UNIX extension to BASE.

19685 **Issue 6**19686           The *ilogb()* function is no longer marked as an extension.

19687           The *ilogbf()* and *ilogbl()* functions are added for alignment with the ISO/IEC 9899:1999  
19688           standard.

19689           The RETURN VALUE section is revised for alignment with the ISO/IEC 9899:1999 standard.

19690           XSI extensions are marked.

19691 **NAME**

19692           imaxabs — return absolute value

19693 **SYNOPSIS**

19694           #include &lt;inttypes.h&gt;

19695           intmax\_t imaxabs(intmax\_t j);

19696 **DESCRIPTION**

19697 cx       The functionality described on this reference page is aligned with the ISO C standard. Any  
19698       conflict between the requirements described here and the ISO C standard is unintentional. This  
19699       volume of IEEE Std 1003.1-200x defers to the ISO C standard.

19700       The *imaxabs()* function shall compute the absolute value of an integer *j*. If the result cannot be  
19701       represented, the behavior is undefined.

19702 **RETURN VALUE**19703       The *imaxabs()* function shall return the absolute value.19704 **ERRORS**

19705       No errors are defined.

19706 **EXAMPLES**

19707       None.

19708 **APPLICATION USAGE**

19709       The absolute value of the most negative number cannot be represented in two's complement.

19710 **RATIONALE**

19711       None.

19712 **FUTURE DIRECTIONS**

19713       None.

19714 **SEE ALSO**19715       *imaxdiv()*, the Base Definitions volume of IEEE Std 1003.1-200x, <inttypes.h>19716 **CHANGE HISTORY**

19717       First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

19718 **NAME**

19719 imaxdiv — return quotient and remainder

19720 **SYNOPSIS**

19721 #include <inttypes.h>

19722 imaxdiv\_t imaxdiv(intmax\_t numer, intmax\_t denom);

19723 **DESCRIPTION**

19724 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
19725 conflict between the requirements described here and the ISO C standard is unintentional. This  
19726 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

19727 The *imaxdiv()* function shall compute *numer / denom* and *numer % denom* in a single operation.

19728 **RETURN VALUE**

19729 The *imaxdiv()* function shall return a structure of type **imaxdiv\_t**, comprising both the quotient  
19730 and the remainder. The structure shall contain (in either order) the members *quot* (the quotient)  
19731 and *rem* (the remainder), each of which has type **intmax\_t**.

19732 If either part of the result cannot be represented, the behavior is undefined.

19733 **ERRORS**

19734 No errors are defined.

19735 **EXAMPLES**

19736 None.

19737 **APPLICATION USAGE**

19738 None.

19739 **RATIONALE**

19740 None.

19741 **FUTURE DIRECTIONS**

19742 None.

19743 **SEE ALSO**

19744 *imaxabs()*, the Base Definitions volume of IEEE Std 1003.1-200x, <inttypes.h>

19745 **CHANGE HISTORY**

19746 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

19747 **NAME**

19748 index — character string operations (**LEGACY**)

19749 **SYNOPSIS**

```
19750 xsi #include <strings.h>
```

```
19751 char *index(const char *s, int c);
```

19752

19753 **DESCRIPTION**

19754 The *index()* function shall be equivalent to *strchr()*.

19755 **RETURN VALUE**

19756 See *strchr()*.

19757 **ERRORS**

19758 See *strchr()*.

19759 **EXAMPLES**

19760 None.

19761 **APPLICATION USAGE**

19762 *strchr()* is preferred over this function.

19763 For maximum portability, it is recommended to replace the function call to *index()* as follows:

```
19764 #define index(a,b) strchr((a),(b))
```

19765 **RATIONALE**

19766 None.

19767 **FUTURE DIRECTIONS**

19768 This function may be withdrawn in a future version.

19769 **SEE ALSO**

19770 *strchr()*, the Base Definitions volume of IEEE Std 1003.1-200x, <**strings.h**>

19771 **CHANGE HISTORY**

19772 First released in Issue 4, Version 2.

19773 **Issue 5**

19774 Moved from X/OPEN UNIX extension to BASE.

19775 **Issue 6**

19776 This function is marked LEGACY.

19777 **NAME**

19778 inet\_addr, inet\_ntoa — IPv4 address manipulation

19779 **SYNOPSIS**

19780 #include &lt;arpa/inet.h&gt;

19781 inet\_addr\_t inet\_addr(const char \*cp);

19782 char \*inet\_ntoa(struct in\_addr in);

19783 **DESCRIPTION**19784 The *inet\_addr()* function shall convert the string pointed to by *cp*, in the standard IPv4 dotted decimal notation, to an integer value suitable for use as an Internet address.19786 The *inet\_ntoa()* function shall convert the Internet host address specified by *in* to a string in the Internet standard dot notation.19788 The *inet\_ntoa()* function need not be reentrant. A function that is not required to be reentrant is not required to be thread-safe.

19790 All Internet addresses shall be returned in network order (bytes ordered from left to right).

19791 Values specified using IPv4 dotted decimal notation take one of the following forms:

19792 a.b.c.d When four parts are specified, each shall be interpreted as a byte of data and |  
19793 assigned, from left to right, to the four bytes of an Internet address. |19794 a.b.c When a three-part address is specified, the last part shall be interpreted as a 16-bit |  
19795 quantity and placed in the rightmost two bytes of the network address. This makes |  
19796 the three-part address format convenient for specifying Class B network addresses |  
19797 as **128.net.host**. |19798 a.b When a two-part address is supplied, the last part shall be interpreted as a 24-bit |  
19799 quantity and placed in the rightmost three bytes of the network address. This |  
19800 makes the two-part address format convenient for specifying Class A network |  
19801 addresses as **net.host**. |19802 a When only one part is given, the value shall be stored directly in the network |  
19803 address without any byte rearrangement. |19804 All numbers supplied as parts in IPv4 dotted decimal notation may be decimal, octal, or  
19805 hexadecimal, as specified in the ISO C standard (that is, a leading 0x or 0X implies hexadecimal;  
19806 otherwise, a leading '0' implies octal; otherwise, the number is interpreted as decimal).19807 **RETURN VALUE**19808 Upon successful completion, *inet\_addr()* shall return the Internet address. Otherwise, it shall  
19809 return (**in\_addr\_t**)(-1).19810 The *inet\_ntoa()* function shall return a pointer to the network address in Internet standard dot  
19811 notation.19812 **ERRORS**

19813 No errors are defined.

19814 **EXAMPLES**

19815           None.

19816 **APPLICATION USAGE**

19817           The return value of *inet\_ntoa()* may point to static data that may be overwritten by subsequent  
19818           calls to *inet\_ntoa()*.

19819 **RATIONALE**

19820           None.

19821 **FUTURE DIRECTIONS**

19822           None.

19823 **SEE ALSO**19824           *endhostent()*, *endnetent()*, the Base Definitions volume of IEEE Std 1003.1-200x, <arpa/inet.h>19825 **CHANGE HISTORY**

19826           First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

19827 **NAME**

19828       inet\_ntoa — IPv4 address manipulation

19829 **SYNOPSIS**

19830       #include <arpa/inet.h>

19831       char \*inet\_ntoa(struct in\_addr *in*);

19832 **DESCRIPTION**

19833       Refer to *inet\_addr()*.

19834 **NAME**

19835 `inet_ntop`, `inet_pton` — convert IPv4 and IPv6 addresses between binary and text form

19836 **SYNOPSIS**

19837 `#include <arpa/inet.h>`

19838 `const char *inet_ntop(int af, const void *restrict src,`  
 19839 `char *restrict dst, socklen_t size);`

19840 `int inet_pton(int af, const char *restrict src, void *restrict dst);`

19841 **DESCRIPTION**

19842 The `inet_ntop()` function shall convert a numeric address into a text string suitable for |  
 19843 IP6 presentation. The `af` argument shall specify the family of the address. This can be AF\_INET or |  
 19844 AF\_INET6. The `src` argument points to a buffer holding an IPv4 address if the `af` argument is  
 19845 IP6 AF\_INET, or an IPv6 address if the `af` argument is AF\_INET6. The `dst` argument points to a  
 19846 buffer where the function stores the resulting text string; it shall not be NULL. The `size` argument  
 19847 specifies the size of this buffer, which shall be large enough to hold the text string  
 19848 IP6 (INET\_ADDRSTRLEN characters for IPv4, INET6\_ADDRSTRLEN characters for IPv6).

19849 The `inet_pton()` function shall convert an address in its standard text presentation form into its |  
 19850 IP6 numeric binary form. The `af` argument shall specify the family of the address. The AF\_INET and |  
 19851 AF\_INET6 address families shall be supported. The `src` argument points to the string being |  
 19852 passed in. The `dst` argument points to a buffer into which the function stores the numeric  
 19853 IP6 address; this shall be large enough to hold the numeric address (32 bits for AF\_INET, 128 bits for  
 19854 AF\_INET6).

19855 If the `af` argument of `inet_pton()` is AF\_INET, the `src` string shall be in the standard IPv4 dotted-  
 19856 decimal form:

19857 `ddd.ddd.ddd.ddd`

19858 where "ddd" is a one to three digit decimal number between 0 and 255 (see `inet_addr()`). The  
 19859 `inet_pton()` function does not accept other formats (such as the octal numbers, hexadecimal  
 19860 numbers, and fewer than four numbers that `inet_addr()` accepts).

19861 IP6 If the `af` argument of `inet_pton()` is AF\_INET6, the `src` string shall be in one of the following  
 19862 standard IPv6 text forms:

19863 1. The preferred form is "`x:x:x:x:x:x:x:x`", where the 'x's are the hexadecimal values  
 19864 of the eight 16-bit pieces of the address. Leading zeros in individual fields can be omitted,  
 19865 but there shall be at least one numeral in every field.

19866 2. A string of contiguous zero fields in the preferred form can be shown as "`:::`". The "`:::`"  
 19867 can only appear once in an address. Unspecified addresses ("`0:0:0:0:0:0:0:0`") may  
 19868 be represented simply as "`:::`".

19869 3. A third form that is sometimes more convenient when dealing with a mixed environment  
 19870 of IPv4 and IPv6 nodes is "`x:x:x:x:x:x.d.d.d.d`", where the 'x's are the  
 19871 hexadecimal values of the six high-order 16-bit pieces of the address, and the 'd's are the  
 19872 decimal values of the four low-order 8-bit pieces of the address (standard IPv4  
 19873 representation).

19874 **Note:** A more extensive description of the standard representations of IPv6 addresses can be found in  
 19875 RFC 2373.

19876

19877 **RETURN VALUE**

19878 The *inet\_ntop()* function shall return a pointer to the buffer containing the text string if the  
19879 conversion succeeds, and NULL otherwise, and set *errno* to indicate the error.

19880 The *inet\_pton()* function shall return 1 if the conversion succeeds, with the address pointed to by  
19881 *dst* in network byte order. It shall return 0 if the input is not a valid IPv4 dotted-decimal string or  
19882 a valid IPv6 address string, or -1 with *errno* set to [EAFNOSUPPORT] if the *af* argument is  
19883 unknown.

19884 **ERRORS**

19885 The *inet\_ntop()* and *inet\_pton()* functions shall fail if:

19886 [EAFNOSUPPORT]

19887 The *af* argument is invalid.

19888 [ENOSPC] The size of the *inet\_ntop()* result buffer is inadequate.

19889 **EXAMPLES**

19890 None.

19891 **APPLICATION USAGE**

19892 None.

19893 **RATIONALE**

19894 None.

19895 **FUTURE DIRECTIONS**

19896 None.

19897 **SEE ALSO**

19898 The Base Definitions volume of IEEE Std 1003.1-200x, <[arpa/inet.h](#)>

19899 **CHANGE HISTORY**

19900 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

19901 IPv6 extensions are marked.

19902 The **restrict** keyword is added to the *inet\_ntop()* and *inet\_pton()* prototypes for alignment with  
19903 the ISO/IEC 9899:1999 standard.

19904 **NAME**

19905           initstate, random, setstate, srandom — pseudo-random number functions

19906 **SYNOPSIS**

```
19907 xSI       #include <stdlib.h>
19908           char *initstate(unsigned seed, char *state, size_t size);
19909           long random(void);
19910           char *setstate(const char *state);
19911           void srandom(unsigned seed);
```

19912

19913 **DESCRIPTION**

19914       The *random()* function shall use a non-linear additive feedback random-number generator |  
 19915       employing a default state array size of 31 **long** integers to return successive pseudo-random |  
 19916       numbers in the range from 0 to  $2^{31}-1$ . The period of this random-number generator is |  
 19917       approximately  $16 \times (2^{31}-1)$ . The size of the state array determines the period of the random- |  
 19918       number generator. Increasing the state array size shall increase the period. |

19919       With 256 bytes of state information, the period of the random-number generator shall be greater |  
 19920       than  $2^{69}$ . |

19921       Like *rand()*, *random()* shall produce by default a sequence of numbers that can be duplicated by |  
 19922       calling *srandom()* with 1 as the seed. |

19923       The *srandom()* function shall initialize the current state array using the value of *seed*. |

19924       The *initstate()* and *setstate()* functions handle restarting and changing random-number |  
 19925       generators. The *initstate()* function allows a state array, pointed to by the *state* argument, to be |  
 19926       initialized for future use. The *size* argument, which specifies the size in bytes of the state array, |  
 19927       shall be used by *initstate()* to decide what type of random-number generator to use; the larger |  
 19928       the state array, the more random the numbers. Values for the amount of state information are 8, |  
 19929       32, 64, 128, and 256 bytes. Other values greater than 8 bytes are rounded down to the nearest one |  
 19930       of these values. If *initstate()* is called with  $8 \leq \textit{size} < 32$ , then *random()* shall use a simple linear |  
 19931       congruential random number generator. The *seed* argument specifies a starting point for the |  
 19932       random-number sequence and provides for restarting at the same point. The *initstate()* function |  
 19933       shall return a pointer to the previous state information array. |

19934       If *initstate()* has not been called, then *random()* shall behave as though *initstate()* had been called |  
 19935       with *seed*=1 and *size*=128. |

19936       Once a state has been initialized, *setstate()* allows switching between state arrays. The array |  
 19937       defined by the *state* argument shall be used for further random-number generation until |  
 19938       *initstate()* is called or *setstate()* is called again. The *setstate()* function shall return a pointer to the |  
 19939       previous state array. |

19940 **RETURN VALUE**

19941       If *initstate()* is called with *size* less than 8, it shall return NULL.

19942       The *random()* function shall return the generated pseudo-random number.

19943       The *srandom()* function shall not return a value.

19944       Upon successful completion, *initstate()* and *setstate()* shall return a pointer to the previous state |  
 19945       array; otherwise, a null pointer shall be returned. |

19946 **ERRORS**

19947 No errors are defined.

19948 **EXAMPLES**

19949 None.

19950 **APPLICATION USAGE**

19951 After initialization, a state array can be restarted at a different point in one of two ways:

- 19952 1. The *initstate()* function can be used, with the desired seed, state array, and size of the  
19953 array.
- 19954 2. The *setstate()* function, with the desired state, can be used, followed by *srandom()* with the  
19955 desired seed. The advantage of using both of these functions is that the size of the state  
19956 array does not have to be saved once it is initialized.

19957 Although some implementations of *random()* have written messages to standard error, such  
19958 implementations do not conform to this volume of IEEE Std 1003.1-200x.

19959 Issue 5 restores the historical behavior of this function.

19960 Threaded applications should use *rand\_r()*, *erand48()*, *nrand48()*, or *jrand48()* instead of  
19961 *random()* when an independent random number sequence in multiple threads is required.19962 **RATIONALE**

19963 None.

19964 **FUTURE DIRECTIONS**

19965 None.

19966 **SEE ALSO**19967 *drand48()*, *rand()*, the Base Definitions volume of IEEE Std 1003.1-200x, <stdlib.h>19968 **CHANGE HISTORY**

19969 First released in Issue 4, Version 2.

19970 **Issue 5**

19971 Moved from X/OPEN UNIX extension to BASE.

19972 In the DESCRIPTION, the phrase “values smaller than 8” is replaced with “values greater than  
19973 or equal to 8, or less than 32”, “*size*<8” is replaced with “ $8 \leq \textit{size} < 32$ ”, and a new first paragraph  
19974 is added to the RETURN VALUE section. A note is added to the APPLICATION USAGE  
19975 indicating that these changes restore the historical behavior of the function.

19976 **Issue 6**

19977 In the DESCRIPTION, duplicate text “For values greater than or equal to 8 . . .” is removed.

19978 **NAME**

19979           insque, remque — insert or remove an element in a queue

19980 **SYNOPSIS**

19981 xSI       #include &lt;search.h&gt;

19982           void insque(void \*element, void \*pred);

19983           void remque(void \*element);

19984

19985 **DESCRIPTION**

19986       The *insque()* and *remque()* functions shall manipulate queues built from doubly-linked lists. The  
 19987       queue can be either circular or linear. An application using *insque()* or *remque()* shall ensure it  
 19988       defines a structure in which the first two members of the structure are pointers to the same type  
 19989       of structure, and any further members are application-specific. The first member of the structure  
 19990       is a forward pointer to the next entry in the queue. The second member is a backward pointer to  
 19991       the previous entry in the queue. If the queue is linear, the queue is terminated with null  
 19992       pointers. The names of the structure and of the pointer members are not subject to any special  
 19993       restriction.

19994       The *insque()* function shall insert the element pointed to by *element* into a queue immediately  
 19995       after the element pointed to by *pred*.

19996       The *remque()* function shall remove the element pointed to by *element* from a queue.

19997       If the queue is to be used as a linear list, invoking *insque(&element, NULL)*, where *element* is the  
 19998       initial element of the queue, shall initialize the forward and backward pointers of *element* to null  
 19999       pointers.

20000       If the queue is to be used as a circular list, the application shall ensure it initializes the forward  
 20001       pointer and the backward pointer of the initial element of the queue to the element's own  
 20002       address.

20003 **RETURN VALUE**20004       The *insque()* and *remque()* functions do not return a value.20005 **ERRORS**

20006       No errors are defined.

20007 **EXAMPLES**20008       **Creating a Linear Linked List**

20009       The following example creates a linear linked list.

20010       #include &lt;search.h&gt;

20011       ...

20012       struct myque element1;

20013       struct myque element2;

20014       char \*data1 = "DATA1";

20015       char \*data2 = "DATA2";

20016       ...

20017       element1.data = data1;

20018       element2.data = data2;

20019       insque (&amp;element1, NULL);

20020       insque (&amp;element2, &amp;element1);

20021 **Creating a Circular Linked List**

20022 The following example creates a circular linked list.

```
20023 #include <search.h>
20024 ...
20025 struct myque element1;
20026 struct myque element2;

20027 char *data1 = "DATA1";
20028 char *data2 = "DATA2";
20029 ...
20030 element1.data = data1;
20031 element2.data = data2;

20032 element1.fwd = &element1;
20033 element1.bck = &element1;

20034 insque (&element2, &element1);
```

20035 **Removing an Element**

20036 The following example removes the element pointed to by *element1*.

```
20037 #include <search.h>
20038 ...
20039 struct myque element1;
20040 ...
20041 remque (&element1);
```

20042 **APPLICATION USAGE**

20043 The historical implementations of these functions described the arguments as being of type  
20044 **struct qelem \*** rather than as being of type **void \*** as defined here. In those implementations,  
20045 **struct qelem** was commonly defined in **<search.h>** as:

```
20046 struct qelem {
20047     struct qelem *q_forw;
20048     struct qelem *q_back;
20049 };
```

20050 Applications using these functions, however, were never able to use this structure directly since  
20051 it provided no room for the actual data contained in the elements. Most applications defined  
20052 structures that contained the two pointers as the initial elements and also provided space for, or  
20053 pointers to, the object's data. Applications that used these functions to update more than one  
20054 type of table also had the problem of specifying two or more different structures with the same  
20055 name, if they literally used **struct qelem** as specified.

20056 As described here, the implementations were actually expecting a structure type where the first  
20057 two members were forward and backward pointers to structures. With C compilers that didn't  
20058 provide function prototypes, applications used structures as specified in the DESCRIPTION  
20059 above and the compiler did what the application expected.

20060 If this method had been carried forward with an ISO C standard compiler and the historical  
20061 function prototype, most applications would have to be modified to cast pointers to the  
20062 structures actually used to be pointers to **struct qelem** to avoid compilation warnings. By  
20063 specifying **void \*** as the argument type, applications do not need to change (unless they  
20064 specifically referenced **struct qelem** and depended on it being defined in **<search.h>**).

20065 **RATIONALE**

20066           None.

20067 **FUTURE DIRECTIONS**

20068           None.

20069 **SEE ALSO**

20070           The Base Definitions volume of IEEE Std 1003.1-200x, <**search.h**>

20071 **CHANGE HISTORY**

20072           First released in Issue 4, Version 2.

20073 **Issue 5**

20074           Moved from X/OPEN UNIX extension to BASE.

20075 **Issue 6**

20076           The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

## 20077 NAME

20078        ioctl — control a STREAMS device (**STREAMS**)

## 20079 SYNOPSIS

20080 XSR        #include &lt;stropts.h&gt;

20081        int ioctl(int *fildev*, int *request*, ... /\* *arg* \*/);

20082

## 20083 DESCRIPTION

20084        The *ioctl()* function shall perform a variety of control functions on STREAMS devices. For non- |  
 20085        STREAMS devices, the functions performed by this call are unspecified. The *request* argument |  
 20086        and an optional third argument (with varying type) shall be passed to and interpreted by the  
 20087        appropriate part of the STREAM associated with *fildev*.

20088        The *fildev* argument is an open file descriptor that refers to a device.

20089        The *request* argument selects the control function to be performed and shall depend on the  
 20090        STREAMS device being addressed.

20091        The *arg* argument represents additional information that is needed by this specific STREAMS  
 20092        device to perform the requested function. The type of *arg* depends upon the particular control  
 20093        request, but it shall be either an integer or a pointer to a device-specific data structure.

20094        The *ioctl()* commands applicable to STREAMS, their arguments, and error conditions that apply  
 20095        to each individual command are described below.

20096        The following *ioctl()* commands, with error values indicated, are applicable to all STREAMS  
 20097        files:

20098        I\_PUSH        Pushes the module whose name is pointed to by *arg* onto the top of the  
 20099        current STREAM, just below the STREAM head. It then calls the *open()*  
 20100        function of the newly-pushed module.

20101        The *ioctl()* function with the I\_PUSH command shall fail if:

20102                [EINVAL]        Invalid module name.

20103                [ENXIO]        Open function of new module failed.

20104                [ENXIO]        Hangup received on *fildev*.

20105        I\_POP        Removes the module just below the STREAM head of the STREAM pointed to  
 20106        by *fildev*. The *arg* argument should be 0 in an I\_POP request.

20107        The *ioctl()* function with the I\_POP command shall fail if:

20108                [EINVAL]        No module present in the STREAM.

20109                [ENXIO]        Hangup received on *fildev*.

20110        I\_LOOK       Retrieves the name of the module just below the STREAM head of the  
 20111        STREAM pointed to by *fildev*, and places it in a character string pointed to by  
 20112        *arg*. The buffer pointed to by *arg* should be at least FMNAMESZ+1 bytes long,  
 20113        where FMNAMESZ is defined in <stropts.h>.

20114        The *ioctl()* function with the I\_LOOK command shall fail if:

20115                [EINVAL]        No module present in the STREAM.

20116        I\_FLUSH       Flushes read and/or write queues, depending on the value of *arg*. Valid *arg* |  
 20117        values are:

20118	FLUSHR	Flush all read queues.
20119	FLUSHW	Flush all write queues.
20120	FLUSHRW	Flush all read and all write queues.
20121	The <i>ioctl()</i> function with the <code>L_FLUSH</code> command shall fail if:	
20122	[EINVAL]	Invalid <i>arg</i> value.
20123	[EAGAIN] or [ENOSR]	Unable to allocate buffers for flush message.
20124		
20125	[ENXIO]	Hangup received on <i>fildev</i> .
20126	I_FLUSHBAND	Flushes a particular band of messages. The <i>arg</i> argument points to a <b>bandinfo</b> structure. The <i>bi_flag</i> member may be one of FLUSHR, FLUSHW, or FLUSHRW as described above. The <i>bi_pri</i> member determines the priority band to be flushed.
20127		
20128		
20129		
20130	I_SETSIG	Requests that the STREAMS implementation send the SIGPOLL signal to the calling process when a particular event has occurred on the STREAM associated with <i>fildev</i> . I_SETSIG supports an asynchronous processing capability in STREAMS. The value of <i>arg</i> is a bitmask that specifies the events for which the process should be signaled. It is the bitwise-inclusive OR of any combination of the following constants:
20131		
20132		
20133		
20134		
20135		
20136	S_RDNORM	A normal (priority band set to 0) message has arrived at the head of a STREAM head read queue. A signal shall be generated even if the message is of zero length.
20137		
20138		
20139	S_RDBAND	A message with a non-zero priority band has arrived at the head of a STREAM head read queue. A signal shall be generated even if the message is of zero length.
20140		
20141		
20142	S_INPUT	A message, other than a high-priority message, has arrived at the head of a STREAM head read queue. A signal shall be generated even if the message is of zero length.
20143		
20144		
20145	S_HIPRI	A high-priority message is present on a STREAM head read queue. A signal shall be generated even if the message is of zero length.
20146		
20147		
20148	S_OUTPUT	The write queue for normal data (priority band 0) just below the STREAM head is no longer full. This notifies the process that there is room on the queue for sending (or writing) normal data downstream.
20149		
20150		
20151		
20152	S_WRNORM	Equivalent to S_OUTPUT.
20153	S_WRBAND	The write queue for a non-zero priority band just below the STREAM head is no longer full. This notifies the process that there is room on the queue for sending (or writing) priority data downstream.
20154		
20155		
20156		
20157	S_MSG	A STREAMS signal message that contains the SIGPOLL signal has reached the front of the STREAM head read queue.
20158		
20159		
20160	S_ERROR	Notification of an error condition has reached the STREAM head.
20161		

20162		S_HANGUP	Notification of a hangup has reached the STREAM head.
20163		S_BANDURG	When used in conjunction with S_RDBAND, SIGURG is generated instead of SIGPOLL when a priority message reaches the front of the STREAM head read queue.
20164			
20165			
20166			If <i>arg</i> is 0, the calling process shall be unregistered and shall not receive further SIGPOLL signals for the stream associated with <i>fildev</i> .
20167			
20168			Processes that wish to receive SIGPOLL signals shall ensure that they explicitly register to receive them using I_SETSIG. If several processes register to receive this signal for the same event on the same STREAM, each process shall be signaled when the event occurs.
20169			
20170			
20171			
20172			The <i>ioctl()</i> function with the I_SETSIG command shall fail if:
20173		[EINVAL]	The value of <i>arg</i> is invalid.
20174		[EINVAL]	The value of <i>arg</i> is 0 and the calling process is not registered to receive the SIGPOLL signal.
20175			
20176		[EAGAIN]	There were insufficient resources to store the signal request.
20177	I_GETSIG		Returns the events for which the calling process is currently registered to be sent a SIGPOLL signal. The events are returned as a bitmask in an <i>int</i> pointed to by <i>arg</i> , where the events are those specified in the description of I_SETSIG above.
20178			
20179			
20180			
20181			The <i>ioctl()</i> function with the I_GETSIG command shall fail if:
20182		[EINVAL]	Process is not registered to receive the SIGPOLL signal.
20183	I_FIND		Compares the names of all modules currently present in the STREAM to the name pointed to by <i>arg</i> , and returns 1 if the named module is present in the STREAM, or returns 0 if the named module is not present.
20184			
20185			
20186			The <i>ioctl()</i> function with the I_FIND command shall fail if:
20187		[EINVAL]	<i>arg</i> does not contain a valid module name.
20188	I_PEEK		Retrieves the information in the first message on the STREAM head read queue without taking the message off the queue. It is analogous to <i>getmsg()</i> except that this command does not remove the message from the queue. The <i>arg</i> argument points to a <b>strpeek</b> structure.
20189			
20190			
20191			
20192			The application shall ensure that the <i>maxlen</i> member in the <b>ctlbuf</b> and <b>databuf</b> structures is set to the number of bytes of control information and/or data information, respectively, to retrieve. The <i>flags</i> member may be marked RS_HIPRI or 0, as described by <i>getmsg()</i> . If the process sets <i>flags</i> to RS_HIPRI, for example, I_PEEK shall only look for a high-priority message on the STREAM head read queue.
20193			
20194			
20195			
20196			
20197			
20198			I_PEEK returns 1 if a message was retrieved, and returns 0 if no message was found on the STREAM head read queue, or if the RS_HIPRI flag was set in <i>flags</i> and a high-priority message was not present on the STREAM head read queue. It does not wait for a message to arrive. On return, <b>ctlbuf</b> specifies information in the control buffer, <b>databuf</b> specifies information in the data buffer, and <i>flags</i> contains the value RS_HIPRI or 0.
20199			
20200			
20201			
20202			
20203			
20204	I_SRDOPT		Sets the read mode using the value of the argument <i>arg</i> . Read modes are described in <i>read()</i> . Valid <i>arg</i> flags are:
20205			

20206		RNORM	Byte-stream mode, the default.
20207		RMSGD	Message-discard mode.
20208		RMSGN	Message-nondiscard mode.
20209			The bitwise-inclusive OR of RMSGD and RMSGN shall return [EINVAL]. The
20210			bitwise-inclusive OR of RNORM and either RMSGD or RMSGN shall result in
20211			the other flag overriding RNORM which is the default.
20212			In addition, treatment of control messages by the STREAM head may be
20213			changed by setting any of the following flags in <i>arg</i> :
20214		RPROTNORM	Fail <i>read()</i> with [EBADMSG] if a message containing a
20215			control part is at the front of the STREAM head read queue.
20216		RPROTDAT	Deliver the control part of a message as data when a
20217			process issues a <i>read()</i> .
20218		RPROTDIS	Discard the control part of a message, delivering any data
20219			portion, when a process issues a <i>read()</i> .
20220			The <i>ioctl()</i> function with the I_SRDOPT command shall fail if:
20221		[EINVAL]	The <i>arg</i> argument is not valid.
20222	I_GRDOPT		Returns the current read mode setting as, described above, in an <b>int</b> pointed to
20223			by the argument <i>arg</i> . Read modes are described in <i>read()</i> .
20224	I_NREAD		Counts the number of data bytes in the data part of the first message on the
20225			STREAM head read queue and places this value in the <b>int</b> pointed to by <i>arg</i> .
20226			The return value for the command shall be the number of messages on the
20227			STREAM head read queue. For example, if 0 is returned in <i>arg</i> , but the <i>ioctl()</i>
20228			return value is greater than 0, this indicates that a zero-length message is next
20229			on the queue.
20230	I_FDINSERT		Creates a message from specified buffer(s), adds information about another
20231			STREAM, and sends the message downstream. The message contains a
20232			control part and an optional data part. The data and control parts to be sent
20233			are distinguished by placement in separate buffers, as described below. The
20234			<i>arg</i> argument points to a <b>strfdinsert</b> structure.
20235			The application shall ensure that the <i>len</i> member in the <b>ctlbuf strbuf</b> structure
20236			is set to the size of a <b>t_uscalar_t</b> plus the number of bytes of control
20237			information to be sent with the message. The <i>fildev</i> member specifies the file
20238			descriptor of the other STREAM, and the <i>offset</i> member, which must be
20239			suitably aligned for use as a <b>t_uscalar_t</b> , specifies the offset from the start of
20240			the control buffer where I_FDINSERT shall store a <b>t_uscalar_t</b> whose
20241			interpretation is specific to the STREAM end. The application shall ensure that
20242			the <i>len</i> member in the <b>datbuf strbuf</b> structure is set to the number of bytes of
20243			data information to be sent with the message, or to 0 if no data part is to be
20244			sent.
20245			The <i>flags</i> member specifies the type of message to be created. A normal
20246			message is created if <i>flags</i> is set to 0, and a high-priority message is created if
20247			<i>flags</i> is set to RS_HIPRI. For non-priority messages, I_FDINSERT shall block if
20248			the STREAM write queue is full due to internal flow control conditions. For
20249			priority messages, I_FDINSERT does not block on this condition. For non-
20250			priority messages, I_FDINSERT does not block when the write queue is full

20251		and O_NONBLOCK is set. Instead, it fails and sets <i>errno</i> to [EAGAIN].
20252		I_FDINSERT also blocks, unless prevented by lack of internal resources,
20253		waiting for the availability of message blocks in the STREAM, regardless of
20254		priority or whether O_NONBLOCK has been specified. No partial message is
20255		sent.
20256		The <i>ioctl()</i> function with the I_FDINSERT command shall fail if:
20257	[EAGAIN]	A non-priority message is specified, the O_NONBLOCK
20258		flag is set, and the STREAM write queue is full due to
20259		internal flow control conditions.
20260	[EAGAIN] or [ENOSR]	
20261		Buffers cannot be allocated for the message that is to be
20262		created.
20263	[EINVAL]	One of the following:
20264		— The <i>fildev</i> member of the <b>strfdinsert</b> structure is not a
20265		valid, open STREAM file descriptor.
20266		— The size of a <b>t_uscalar_t</b> plus <i>offset</i> is greater than the <i>len</i>
20267		member for the buffer specified through <b>ctlbuf</b> .
20268		— The <i>offset</i> member does not specify a properly-aligned
20269		location in the data buffer.
20270		— An undefined value is stored in <i>flags</i> .
20271	[ENXIO]	Hangup received on the STREAM identified by either the
20272		<i>fildev</i> argument or the <i>fildev</i> member of the <b>strfdinsert</b>
20273		structure.
20274	[ERANGE]	The <i>len</i> member for the buffer specified through <b>databuf</b>
20275		does not fall within the range specified by the maximum
20276		and minimum packet sizes of the topmost STREAM module
20277		or the <i>len</i> member for the buffer specified through <b>databuf</b>
20278		is larger than the maximum configured size of the data part
20279		of a message; or the <i>len</i> member for the buffer specified
20280		through <b>ctlbuf</b> is larger than the maximum configured size
20281		of the control part of a message.
20282	I_STR	Constructs an internal STREAMS <i>ioctl()</i> message from the data pointed to by
20283		<i>arg</i> , and sends that message downstream.
20284		This mechanism is provided to send <i>ioctl()</i> requests to downstream modules
20285		and drivers. It allows information to be sent with <i>ioctl()</i> , and returns to the
20286		process any information sent upstream by the downstream recipient. I_STR
20287		shall block until the system responds with either a positive or negative
20288		acknowledgement message, or until the request times out after some period of
20289		time. If the request times out, it shall fail with <i>errno</i> set to [ETIME].
20290		At most, one I_STR can be active on a STREAM. Further I_STR calls shall
20291		block until the active I_STR completes at the STREAM head. The default
20292		timeout interval for these requests is 15 seconds. The O_NONBLOCK flag has
20293		no effect on this call.
20294		To send requests downstream, the application shall ensure that <i>arg</i> points to a
20295		<b>striocctl</b> structure.

20296 The *ic\_cmd* member is the internal *ioctl()* command intended for a  
 20297 downstream module or driver and *ic\_timeout* is the number of seconds  
 20298 (−1=infinite, 0=use implementation-defined timeout interval, >0=as specified)  
 20299 an I\_STR request shall wait for acknowledgement before timing out. *ic\_len* is  
 20300 the number of bytes in the data argument, and *ic\_dp* is a pointer to the data  
 20301 argument. The *ic\_len* member has two uses: on input, it contains the length of  
 20302 the data argument passed in, and on return from the command, it contains the  
 20303 number of bytes being returned to the process (the buffer pointed to by *ic\_dp*  
 20304 should be large enough to contain the maximum amount of data that any  
 20305 module or the driver in the STREAM can return).

20306 The STREAM head shall convert the information pointed to by the **strioc**  
 20307 structure to an internal *ioctl()* command message and sends it downstream.

20308 The *ioctl()* function with the I\_STR command shall fail if:

20309 [EAGAIN] or [ENOSR]  
 20310 Unable to allocate buffers for the *ioctl()* message.

20311 [EINVAL] The *ic\_len* member is less than 0 or larger than the  
 20312 maximum configured size of the data part of a message, or  
 20313 *ic\_timeout* is less than −1.

20314 [ENXIO] Hangup received on *fil*des.

20315 [ETIME] A downstream *ioctl()* timed out before acknowledgement  
 20316 was received.

20317 An I\_STR can also fail while waiting for an acknowledgement if a message  
 20318 indicating an error or a hangup is received at the STREAM head. In addition,  
 20319 an error code can be returned in the positive or negative acknowledgement  
 20320 message, in the event the *ioctl()* command sent downstream fails. For these  
 20321 cases, I\_STR shall fail with *errno* set to the value in the message.

20322 I\_SWROPT Sets the write mode using the value of the argument *arg*. Valid bit settings for  
 20323 *arg* are:

20324 SNDZERO Send a zero-length message downstream when a *write()* of  
 20325 0 bytes occurs. To not send a zero-length message when a  
 20326 *write()* of 0 bytes occurs, the application shall ensure that  
 20327 this bit is not set in *arg* (for example, *arg* would be set to 0).

20328 The *ioctl()* function with the I\_SWROPT command shall fail if:

20329 [EINVAL] *arg* is not the above value.

20330 I\_GWROPT Returns the current write mode setting, as described above, in the **int** that is  
 20331 pointed to by the argument *arg*.

20332 I\_SENDFD Creates a new reference to the open file description associated with the file |  
 20333 descriptor *arg*, and writes a message on the STREAMS-based pipe *fil*des |  
 20334 containing this reference, together with the user ID and group ID of the calling  
 20335 process.

20336 The *ioctl()* function with the I\_SENDFD command shall fail if:

20337 [EAGAIN] The sending STREAM is unable to allocate a message block  
 20338 to contain the file pointer; or the read queue of the receiving  
 20339 STREAM head is full and cannot accept the message sent by  
 20340 I\_SENDFD.

20341		[EBADF]	The <i>arg</i> argument is not a valid, open file descriptor.
20342		[EINVAL]	The <i>fildev</i> argument is not connected to a STREAM pipe.
20343		[ENXIO]	Hangup received on <i>fildev</i> .
20344	I_RECVFD		Retrieves the reference to an open file description from a message written to a STREAMS-based pipe using the I_SENDFD command, and allocates a new file descriptor in the calling process that refers to this open file description. The <i>arg</i> argument is a pointer to a <b>strrecvfd</b> data structure as defined in <b>&lt;stropts.h&gt;</b> .
20345			
20346			
20347			
20348			
20349			The <i>fd</i> member is a file descriptor. The <i>uid</i> and <i>gid</i> members are the effective user ID and effective group ID, respectively, of the sending process.
20350			
20351			If O_NONBLOCK is not set, I_RECVFD shall block until a message is present at the STREAM head. If O_NONBLOCK is set, I_RECVFD shall fail with <i>errno</i> set to [EAGAIN] if no message is present at the STREAM head.
20352			
20353			
20354			If the message at the STREAM head is a message sent by an I_SENDFD, a new file descriptor shall be allocated for the open file descriptor referenced in the message. The new file descriptor is placed in the <i>fd</i> member of the <b>strrecvfd</b> structure pointed to by <i>arg</i> .
20355			
20356			
20357			
20358			The <i>ioctl()</i> function with the I_RECVFD command shall fail if:
20359		[EAGAIN]	A message is not present at the STREAM head read queue and the O_NONBLOCK flag is set.
20360			
20361		[EBADMSG]	The message at the STREAM head read queue is not a message containing a passed file descriptor.
20362			
20363		[EMFILE]	The process has the maximum number of file descriptors currently open that it is allowed.
20364			
20365		[ENXIO]	Hangup received on <i>fildev</i> .
20366	I_LIST		Allows the process to list all the module names on the STREAM, up to and including the topmost driver name. If <i>arg</i> is a null pointer, the return value shall be the number of modules, including the driver, that are on the STREAM pointed to by <i>fildev</i> . This lets the process allocate enough space for the module names. Otherwise, it should point to a <b>str_list</b> structure.
20367			
20368			
20369			
20370			
20371			The <i>sl_nmods</i> member indicates the number of entries the process has allocated in the array. Upon return, the <i>sl_modlist</i> member of the <b>str_list</b> structure shall contain the list of module names, and the number of entries that have been filled into the <i>sl_modlist</i> array is found in the <i>sl_nmods</i> member (the number includes the number of modules including the driver). The return value from <i>ioctl()</i> shall be 0. The entries are filled in starting at the top of the STREAM and continuing downstream until either the end of the STREAM is reached, or the number of requested modules ( <i>sl_nmods</i> ) is satisfied.
20372			
20373			
20374			
20375			
20376			
20377			
20378			
20379			The <i>ioctl()</i> function with the I_LIST command shall fail if:
20380		[EINVAL]	The <i>sl_nmods</i> member is less than 1.
20381		[EAGAIN] or [ENOSR]	Unable to allocate buffers.
20382			
20383	I_ATMARK		Allows the process to see if the message at the head of the STREAM head read queue is marked by some module downstream. The <i>arg</i> argument determines
20384			

20385		how the checking is done when there may be multiple marked messages on
20386		the STREAM head read queue. It may take on the following values:
20387	ANYMARK	Check if the message is marked.
20388	LASTMARK	Check if the message is the last one marked on the queue.
20389		The bitwise-inclusive OR of the flags ANYMARK and LASTMARK is
20390		permitted.
20391		The return value shall be 1 if the mark condition is satisfied; otherwise, the
20392		value shall be 0.
20393		The <i>ioctl()</i> function with the L_ATMARK command shall fail if:
20394		[EINVAL] Invalid <i>arg</i> value.
20395	I_CKBAND	Checks if the message of a given priority band exists on the STREAM head
20396		read queue. This shall return 1 if a message of the given priority exists, 0 if no
20397		such message exists, or -1 on error. <i>arg</i> should be of type <b>int</b> .
20398		The <i>ioctl()</i> function with the I_CKBAND command shall fail if:
20399		[EINVAL] Invalid <i>arg</i> value.
20400	I_GETBAND	Returns the priority band of the first message on the STREAM head read
20401		queue in the integer referenced by <i>arg</i> .
20402		The <i>ioctl()</i> function with the I_GETBAND command shall fail if:
20403		[ENODATA] No message on the STREAM head read queue.
20404	I_CANPUT	Checks if a certain band is writable. <i>arg</i> is set to the priority band in question.
20405		The return value shall be 0 if the band is flow-controlled, 1 if the band is
20406		writable, or -1 on error.
20407		The <i>ioctl()</i> function with the I_CANPUT command shall fail if:
20408		[EINVAL] Invalid <i>arg</i> value.
20409	I_SETCLTIME	This request allows the process to set the time the STREAM head shall delay
20410		when a STREAM is closing and there is data on the write queues. Before
20411		closing each module or driver, if there is data on its write queue, the STREAM
20412		head shall delay for the specified amount of time to allow the data to drain. If,
20413		after the delay, data is still present, it shall be flushed. The <i>arg</i> argument is a
20414		pointer to an integer specifying the number of milliseconds to delay, rounded
20415		up to the nearest valid value. If I_SETCLTIME is not performed on a STREAM,
20416		an implementation-defined default timeout interval is used.
20417		The <i>ioctl()</i> function with the I_SETCLTIME command shall fail if:
20418		[EINVAL] Invalid <i>arg</i> value.
20419	I_GETCLTIME	Returns the close time delay in the integer pointed to by <i>arg</i> .

20420 **Multiplexed STREAMS Configurations**

20421 The following commands are used for connecting and disconnecting multiplexed STREAMS  
20422 configurations. These commands use an implementation-defined default timeout interval.

20423 **I\_LINK** Connects two STREAMs, where *filde*s is the file descriptor of the STREAM  
20424 connected to the multiplexing driver, and *arg* is the file descriptor of the  
20425 STREAM connected to another driver. The STREAM designated by *arg* is  
20426 connected below the multiplexing driver. I\_LINK requires the multiplexing  
20427 driver to send an acknowledgement message to the STREAM head regarding  
20428 the connection. This call shall return a multiplexer ID number (an identifier  
20429 used to disconnect the multiplexer; see I\_UNLINK) on success, and -1 on  
20430 failure.

20431 The *ioctl()* function with the I\_LINK command shall fail if:

20432 [ENXIO] Hangup received on *filde*s.  
20433 [ETIME] Timeout before acknowledgement message was received at  
20434 STREAM head.  
20435 [EAGAIN] or [ENOSR]  
20436 Unable to allocate STREAMS storage to perform the  
20437 I\_LINK.  
20438 [EBADF] The *arg* argument is not a valid, open file descriptor.  
20439 [EINVAL] The *filde*s argument does not support multiplexing; or *arg* is  
20440 not a STREAM or is already connected downstream from a  
20441 multiplexer; or the specified I\_LINK operation would  
20442 connect the STREAM head in more than one place in the  
20443 multiplexed STREAM.

20444 An I\_LINK can also fail while waiting for the multiplexing driver to  
20445 acknowledge the request, if a message indicating an error or a hangup is  
20446 received at the STREAM head of *filde*s. In addition, an error code can be  
20447 returned in the positive or negative acknowledgement message. For these  
20448 cases, I\_LINK fails with *errno* set to the value in the message.

20449 **I\_UNLINK** Disconnects the two STREAMs specified by *filde*s and *arg*. *filde*s is the file  
20450 descriptor of the STREAM connected to the multiplexing driver. The *arg*  
20451 argument is the multiplexer ID number that was returned by the I\_LINK  
20452 *ioctl()* command when a STREAM was connected downstream from the  
20453 multiplexing driver. If *arg* is MUXID\_ALL, then all STREAMs that were  
20454 connected to *filde*s shall be disconnected. As in I\_LINK, this command  
20455 requires acknowledgement.

20456 The *ioctl()* function with the I\_UNLINK command shall fail if:

20457 [ENXIO] Hangup received on *filde*s.  
20458 [ETIME] Timeout before acknowledgement message was received at  
20459 STREAM head.  
20460 [EAGAIN] or [ENOSR]  
20461 Unable to allocate buffers for the acknowledgement  
20462 message.  
20463 [EINVAL] Invalid multiplexer ID number.

20464 An I\_UNLINK can also fail while waiting for the multiplexing driver to  
 20465 acknowledge the request if a message indicating an error or a hangup is  
 20466 received at the STREAM head of *filde*s. In addition, an error code can be  
 20467 returned in the positive or negative acknowledgement message. For these  
 20468 cases, I\_UNLINK shall fail with *errno* set to the value in the message.

20469 I\_PLINK Creates a *persistent connection* between two STREAMs, where *filde*s is the file  
 20470 descriptor of the STREAM connected to the multiplexing driver, and *arg* is the  
 20471 file descriptor of the STREAM connected to another driver. This call shall  
 20472 create a persistent connection which can exist even if the file descriptor *filde*s  
 20473 associated with the upper STREAM to the multiplexing driver is closed. The  
 20474 STREAM designated by *arg* gets connected via a persistent connection below  
 20475 the multiplexing driver. I\_PLINK requires the multiplexing driver to send an  
 20476 acknowledgement message to the STREAM head. This call shall return a  
 20477 multiplexer ID number (an identifier that may be used to disconnect the  
 20478 multiplexer; see I\_PUNLINK) on success, and -1 on failure.

20479 The *ioctl*() function with the I\_PLINK command shall fail if:

20480 [ENXIO] Hangup received on *filde*s.

20481 [ETIME] Timeout before acknowledgement message was received at  
 20482 STREAM head.

20483 [EAGAIN] or [ENOSR]  
 20484 Unable to allocate STREAMS storage to perform the  
 20485 I\_PLINK.

20486 [EBADF] The *arg* argument is not a valid, open file descriptor.

20487 [EINVAL] The *filde*s argument does not support multiplexing; or *arg* is  
 20488 not a STREAM or is already connected downstream from a  
 20489 multiplexer; or the specified I\_PLINK operation would  
 20490 connect the STREAM head in more than one place in the  
 20491 multiplexed STREAM.

20492 An I\_PLINK can also fail while waiting for the multiplexing driver to  
 20493 acknowledge the request, if a message indicating an error or a hangup is  
 20494 received at the STREAM head of *filde*s. In addition, an error code can be  
 20495 returned in the positive or negative acknowledgement message. For these  
 20496 cases, I\_PLINK shall fail with *errno* set to the value in the message.

20497 I\_PUNLINK Disconnects the two STREAMs specified by *filde*s and *arg* from a persistent  
 20498 connection. The *filde*s argument is the file descriptor of the STREAM  
 20499 connected to the multiplexing driver. The *arg* argument is the multiplexer ID  
 20500 number that was returned by the I\_PLINK *ioctl*() command when a STREAM  
 20501 was connected downstream from the multiplexing driver. If *arg* is  
 20502 MUXID\_ALL, then all STREAMs which are persistent connections to *filde*s  
 20503 shall be disconnected. As in I\_PLINK, this command requires the multiplexing  
 20504 driver to acknowledge the request.

20505 The *ioctl*() function with the I\_PUNLINK command shall fail if:

20506 [ENXIO] Hangup received on *filde*s.

20507 [ETIME] Timeout before acknowledgement message was received at  
 20508 STREAM head.

20509 [EAGAIN] or [ENOSR]  
 20510 Unable to allocate buffers for the acknowledgement  
 20511 message.  
 20512 [EINVAL] Invalid multiplexer ID number.  
 20513 An I\_PUNLINK can also fail while waiting for the multiplexing driver to  
 20514 acknowledge the request if a message indicating an error or a hangup is  
 20515 received at the STREAM head of *fildev*. In addition, an error code can be  
 20516 returned in the positive or negative acknowledgement message. For these  
 20517 cases, I\_PUNLINK shall fail with *errno* set to the value in the message.

20518 **RETURN VALUE**

20519 Upon successful completion, *ioctl()* shall return a value other than  $-1$  that depends upon the  
 20520 STREAMS device control function. Otherwise, it shall return  $-1$  and set *errno* to indicate the  
 20521 error.

20522 **ERRORS**

20523 Under the following general conditions, *ioctl()* shall fail if:

20524 [EBADF] The *fildev* argument is not a valid open file descriptor.  
 20525 [EINTR] A signal was caught during the *ioctl()* operation.  
 20526 [EINVAL] The STREAM or multiplexer referenced by *fildev* is linked (directly or  
 20527 indirectly) downstream from a multiplexer.

20528 If an underlying device driver detects an error, then *ioctl()* shall fail if:

20529 [EINVAL] The *request* or *arg* argument is not valid for this device.  
 20530 [EIO] Some physical I/O error has occurred.  
 20531 [ENOTTY] The *fildev* argument is not associated with a STREAMS device that accepts  
 20532 control functions.  
 20533 [ENXIO] The *request* and *arg* arguments are valid for this device driver, but the service  
 20534 requested cannot be performed on this particular sub-device.  
 20535 [ENODEV] The *fildev* argument refers to a valid STREAMS device, but the corresponding  
 20536 device driver does not support the *ioctl()* function.

20537 If a STREAM is connected downstream from a multiplexer, any *ioctl()* command except  
 20538 I\_UNLINK and I\_PUNLINK shall set *errno* to [EINVAL].

20539 **EXAMPLES**

20540 None.

20541 **APPLICATION USAGE**

20542 The implementation-defined timeout interval for STREAMS has historically been 15 seconds.

20543 **RATIONALE**

20544 None.

20545 **FUTURE DIRECTIONS**

20546 None.

20547 **SEE ALSO**

20548 *close()*, *fcntl()*, *getmsg()*, *open()*, *pipe()*, *poll()*, *putmsg()*, *read()*, *sigaction()*, *write()*, the Base  
 20549 Definitions volume of IEEE Std 1003.1-200x, <**stropts.h**>, Section 2.6 (on page 488)

20550 **CHANGE HISTORY**

20551 First released in Issue 4, Version 2.

20552 **Issue 5**

20553 Moved from X/OPEN UNIX extension to BASE.

20554 **Issue 6**

20555 The Open Group Corrigendum U028/4 is applied, correcting text in the I\_FDINSERT, [EINVAL] case to refer to *ctlbuf*.

20557 This function is marked as part of the XSI STREAMS Option Group.

20558 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

20559 **NAME**

20560 isalnum — test for an alphanumeric character

20561 **SYNOPSIS**

20562 #include <ctype.h>

20563 int isalnum(int c);

20564 **DESCRIPTION**

20565 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
20566 conflict between the requirements described here and the ISO C standard is unintentional. This  
20567 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

20568 The *isalnum()* function shall test whether *c* is a character of class **alpha** or **digit** in the program's  
20569 current locale; see the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 7, Locale.

20570 The *c* argument is an **int**, the value of which the application shall ensure is representable as an  
20571 **unsigned char** or equal to the value of the macro EOF. If the argument has any other value, the  
20572 behavior is undefined.

20573 **RETURN VALUE**

20574 The *isalnum()* function shall return non-zero if *c* is an alphanumeric character; otherwise, it shall  
20575 return 0.

20576 **ERRORS**

20577 No errors are defined.

20578 **EXAMPLES**

20579 None.

20580 **APPLICATION USAGE**

20581 To ensure applications portability, especially across natural languages, only this function and  
20582 those listed in the SEE ALSO section should be used for character classification.

20583 **RATIONALE**

20584 None.

20585 **FUTURE DIRECTIONS**

20586 None.

20587 **SEE ALSO**

20588 *isalpha()*, *iscntrl()*, *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isspace()*, *isupper()*, *isxdigit()*,  
20589 *setlocale()*, the Base Definitions volume of IEEE Std 1003.1-200x, <ctype.h>, <stdio.h>, the Base  
20590 Definitions volume of IEEE Std 1003.1-200x, Chapter 7, Locale

20591 **CHANGE HISTORY**

20592 First released in Issue 1. Derived from Issue 1 of the SVID.

20593 **Issue 6**

20594 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

20595 **NAME**

20596           isalpha — test for an alphabetic character

20597 **SYNOPSIS**

20598           #include <ctype.h>

20599           int isalpha(int c);

20600 **DESCRIPTION**

20601 cx       The functionality described on this reference page is aligned with the ISO C standard. Any  
20602       conflict between the requirements described here and the ISO C standard is unintentional. This  
20603       volume of IEEE Std 1003.1-200x defers to the ISO C standard.

20604       The *isalpha()* function shall test whether *c* is a character of class **alpha** in the program's current  
20605       locale; see the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 7, Locale.

20606       The *c* argument is an **int**, the value of which the application shall ensure is representable as an  
20607       **unsigned char** or equal to the value of the macro EOF. If the argument has any other value, the  
20608       behavior is undefined.

20609 **RETURN VALUE**

20610       The *isalpha()* function shall return non-zero if *c* is an alphabetic character; otherwise, it shall  
20611       return 0.

20612 **ERRORS**

20613       No errors are defined.

20614 **EXAMPLES**

20615       None.

20616 **APPLICATION USAGE**

20617       To ensure applications portability, especially across natural languages, only this function and  
20618       those listed in the SEE ALSO section should be used for character classification.

20619 **RATIONALE**

20620       None.

20621 **FUTURE DIRECTIONS**

20622       None.

20623 **SEE ALSO**

20624       *isalnum()*, *isctrl()*, *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isspace()*, *isupper()*,  
20625       *isxdigit()*, *setlocale()*, the Base Definitions volume of IEEE Std 1003.1-200x, <ctype.h>, <stdio.h>,  
20626       the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 7, Locale

20627 **CHANGE HISTORY**

20628       First released in Issue 1. Derived from Issue 1 of the SVID.

20629 **Issue 6**

20630       The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

20631 **NAME**20632        **isascii** — test for a 7-bit US-ASCII character20633 **SYNOPSIS**20634 **XSI**        #include <ctype.h>

20635        int isascii(int c);

20636

20637 **DESCRIPTION**20638        The *isascii()* function shall test whether *c* is a 7-bit US-ASCII character code.20639        The *isascii()* function is defined on all integer values.20640 **RETURN VALUE**20641        The *isascii()* function shall return non-zero if *c* is a 7-bit US-ASCII character code between 0 and

20642        octal 0177 inclusive; otherwise, it shall return 0.

20643 **ERRORS**

20644        No errors are defined.

20645 **EXAMPLES**

20646        None.

20647 **APPLICATION USAGE**

20648        None.

20649 **RATIONALE**

20650        None.

20651 **FUTURE DIRECTIONS**

20652        None.

20653 **SEE ALSO**

20654        The Base Definitions volume of IEEE Std 1003.1-200x, &lt;ctype.h&gt;

20655 **CHANGE HISTORY**

20656        First released in Issue 1. Derived from Issue 1 of the SVID.

20657 **NAME**20658 isastream — test a file descriptor (**STREAMS**)20659 **SYNOPSIS**20660 XSR `#include <stropts.h>`20661 `int isastream(int fildev);`

20662

20663 **DESCRIPTION**20664 The *isastream()* function shall test whether *fildev*, an open file descriptor, is associated with a  
20665 STREAMS-based file.20666 **RETURN VALUE**20667 Upon successful completion, *isastream()* shall return 1 if *fildev* refers to a STREAMS-based file  
20668 and 0 if not. Otherwise, *isastream()* shall return -1 and set *errno* to indicate the error.20669 **ERRORS**20670 The *isastream()* function shall fail if:20671 [EBADF] The *fildev* argument is not a valid open file descriptor.20672 **EXAMPLES**

20673 None.

20674 **APPLICATION USAGE**

20675 None.

20676 **RATIONALE**

20677 None.

20678 **FUTURE DIRECTIONS**

20679 None.

20680 **SEE ALSO**20681 The Base Definitions volume of IEEE Std 1003.1-200x, <**stropts.h**>20682 **CHANGE HISTORY**

20683 First released in Issue 4, Version 2.

20684 **Issue 5**

20685 Moved from X/OPEN UNIX extension to BASE.

20686 **NAME**

20687           isatty — test for a terminal device

20688 **SYNOPSIS**

20689           #include <unistd.h>

20690           int isatty(int *fdes*);

20691 **DESCRIPTION**

20692           The *isatty()* function shall test whether *fdes*, an open file descriptor, is associated with a  
20693           terminal device.

20694 **RETURN VALUE**

20695           The *isatty()* function shall return 1 if *fdes* is associated with a terminal; otherwise, it shall return  
20696           0 and may set *errno* to indicate the error.

20697 **ERRORS**

20698           The *isatty()* function may fail if:

20699           [EBADF]           The *fdes* argument is not a valid open file descriptor.

20700           [ENOTTY]         The *fdes* argument is not associated with a terminal.

20701 **EXAMPLES**

20702           None.

20703 **APPLICATION USAGE**

20704           The *isatty()* function does not necessarily indicate that a human being is available for interaction  
20705           via *fdes*. It is quite possible that non-terminal devices are connected to the communications  
20706           line.

20707 **RATIONALE**

20708           None.

20709 **FUTURE DIRECTIONS**

20710           None.

20711 **SEE ALSO**

20712           The Base Definitions volume of IEEE Std 1003.1-200x, <unistd.h>

20713 **CHANGE HISTORY**

20714           First released in Issue 1. Derived from Issue 1 of the SVID.

20715 **Issue 6**

20716           The following new requirements on POSIX implementations derive from alignment with the  
20717           Single UNIX Specification:

- 20718           • The optional setting of *errno* to indicate an error is added.
- 20719           • The [EBADF] and [ENOTTY] optional error conditions are added.

20720 **NAME**

20721           isblank — test for a blank character

20722 **SYNOPSIS**

20723           #include <ctype.h>

20724           int isblank(int c);

20725 **DESCRIPTION**

20726 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
20727       conflict between the requirements described here and the ISO C standard is unintentional. This  
20728       volume of IEEE Std 1003.1-200x defers to the ISO C standard.

20729       The *isblank()* function shall test whether *c* is a character of class **blank** in the program's current  
20730       locale; see the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 7, Locale.

20731       The *c* argument is a type **int**, the value of which the application shall ensure is a character  
20732       representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has  
20733       any other value, the behavior is undefined.

20734 **RETURN VALUE**

20735       The *isblank()* function shall return non-zero if *c* is a <blank>; otherwise, it shall return 0.

20736 **ERRORS**

20737       No errors are defined.

20738 **EXAMPLES**

20739       None.

20740 **APPLICATION USAGE**

20741       To ensure applications portability, especially across natural languages, only this function and  
20742       those listed in the SEE ALSO section should be used for character classification.

20743 **RATIONALE**

20744       None.

20745 **FUTURE DIRECTIONS**

20746       None.

20747 **SEE ALSO**

20748       *isalnum()*, *isalpha()*, *iscntrl()*, *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isspace()*, *isupper()*,  
20749       *isxdigit()*, *setlocale()*, the Base Definitions volume of IEEE Std 1003.1-200x, <ctype.h>

20750 **CHANGE HISTORY**

20751       First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

20752 **NAME**

20753           isctrl — test for a control character

20754 **SYNOPSIS**

20755           #include &lt;ctype.h&gt;

20756           int isctrl(int c);

20757 **DESCRIPTION**

20758 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
20759       conflict between the requirements described here and the ISO C standard is unintentional. This  
20760       volume of IEEE Std 1003.1-200x defers to the ISO C standard.

20761       The *isctrl()* function shall test whether *c* is a character of class **cntrl** in the program's current  
20762       locale; see the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 7, Locale.

20763       The *c* argument is a type **int**, the value of which the application shall ensure is a character  
20764       representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has  
20765       any other value, the behavior is undefined.

20766 **RETURN VALUE**20767       The *isctrl()* function shall return non-zero if *c* is a control character; otherwise, it shall return 0.20768 **ERRORS**

20769       No errors are defined.

20770 **EXAMPLES**

20771       None.

20772 **APPLICATION USAGE**

20773       To ensure applications portability, especially across natural languages, only this function and  
20774       those listed in the SEE ALSO section should be used for character classification.

20775 **RATIONALE**

20776       None.

20777 **FUTURE DIRECTIONS**

20778       None.

20779 **SEE ALSO**

20780       *isalnum()*, *isalpha()*, *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isspace()*, *isupper()*,  
20781       *isxdigit()*, *setlocale()*, the Base Definitions volume of IEEE Std 1003.1-200x, <ctype.h>, the Base  
20782       Definitions volume of IEEE Std 1003.1-200x, Chapter 7, Locale

20783 **CHANGE HISTORY**

20784       First released in Issue 1. Derived from Issue 1 of the SVID.

20785 **Issue 6**

20786       The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

20787 **NAME**

20788            isdigit — test for a decimal digit

20789 **SYNOPSIS**

20790            #include &lt;ctype.h&gt;

20791            int isdigit(int c);

20792 **DESCRIPTION**

20793 **CX**        The functionality described on this reference page is aligned with the ISO C standard. Any  
20794            conflict between the requirements described here and the ISO C standard is unintentional. This  
20795            volume of IEEE Std 1003.1-200x defers to the ISO C standard.

20796            The *isdigit()* function shall test whether *c* is a character of class **digit** in the program's current  
20797            locale; see the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 7, Locale.

20798            The *c* argument is an **int**, the value of which the application shall ensure is a character  
20799            representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has  
20800            any other value, the behavior is undefined.

20801 **RETURN VALUE**20802            The *isdigit()* function shall return non-zero if *c* is a decimal digit; otherwise, it shall return 0.20803 **ERRORS**

20804            No errors are defined.

20805 **EXAMPLES**

20806            None.

20807 **APPLICATION USAGE**

20808            To ensure applications portability, especially across natural languages, only this function and  
20809            those listed in the SEE ALSO section should be used for character classification.

20810 **RATIONALE**

20811            None.

20812 **FUTURE DIRECTIONS**

20813            None.

20814 **SEE ALSO**

20815            *isalnum()*, *isalpha()*, *iscntrl()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isspace()*, *isupper()*,  
20816            *isxdigit()*, the Base Definitions volume of IEEE Std 1003.1-200x, <ctype.h>

20817 **CHANGE HISTORY**

20818            First released in Issue 1. Derived from Issue 1 of the SVID.

20819 **Issue 6**

20820            The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

20821 **NAME**

20822           isfinite — test for finite value

20823 **SYNOPSIS**

20824           #include &lt;math.h&gt;

20825           int isfinite(real-floating x);

20826 **DESCRIPTION**

20827 *cx*       The functionality described on this reference page is aligned with the ISO C standard. Any  
20828 conflict between the requirements described here and the ISO C standard is unintentional. This  
20829 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

20830       The *isfinite()* macro shall determine whether its argument has a finite value (zero, subnormal, or  
20831 normal, and not infinite or NaN). First, an argument represented in a format wider than its  
20832 semantic type is converted to its semantic type. Then determination is based on the type of the  
20833 argument.

20834 **RETURN VALUE**20835       The *isfinite()* macro shall return a non-zero value if and only if its argument has a finite value.20836 **ERRORS**

20837       No errors are defined.

20838 **EXAMPLES**

20839       None.

20840 **APPLICATION USAGE**

20841       None.

20842 **RATIONALE**

20843       None.

20844 **FUTURE DIRECTIONS**

20845       None.

20846 **SEE ALSO**

20847       *fpclassify()*, *isinf()*, *isnan()*, *isnormal()*, *signbit()*, the Base Definitions volume of  
20848 IEEE Std 1003.1-200x <math.h>

20849 **CHANGE HISTORY**

20850       First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

20851 **NAME**

20852           isgraph — test for a visible character

20853 **SYNOPSIS**

20854           #include <ctype.h>

20855           int isgraph(int c);

20856 **DESCRIPTION**

20857 *cx*       The functionality described on this reference page is aligned with the ISO C standard. Any  
20858       conflict between the requirements described here and the ISO C standard is unintentional. This  
20859       volume of IEEE Std 1003.1-200x defers to the ISO C standard.

20860       The *isgraph()* function shall test whether *c* is a character of class **graph** in the program's current  
20861       locale; see the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 7, Locale.

20862       The *c* argument is an **int**, the value of which the application shall ensure is a character  
20863       representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has  
20864       any other value, the behavior is undefined.

20865 **RETURN VALUE**

20866       The *isgraph()* function shall return non-zero if *c* is a character with a visible representation;  
20867       otherwise, it shall return 0.

20868 **ERRORS**

20869       No errors are defined.

20870 **EXAMPLES**

20871       None.

20872 **APPLICATION USAGE**

20873       To ensure applications portability, especially across natural languages, only this function and  
20874       those listed in the SEE ALSO section should be used for character classification.

20875 **RATIONALE**

20876       None.

20877 **FUTURE DIRECTIONS**

20878       None.

20879 **SEE ALSO**

20880       *isalnum()*, *isalpha()*, *iscntrl()*, *isdigit()*, *islower()*, *isprint()*, *ispunct()*, *isspace()*, *isupper()*, *isxdigit()*,  
20881       *setlocale()*, the Base Definitions volume of IEEE Std 1003.1-200x, <ctype.h>, the Base Definitions  
20882       volume of IEEE Std 1003.1-200x, Chapter 7, Locale

20883 **CHANGE HISTORY**

20884       First released in Issue 1. Derived from Issue 1 of the SVID.

20885 **Issue 6**

20886       The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

20887 **NAME**

20888           isgreater — test if x greater than y

20889 **SYNOPSIS**

20890           #include &lt;math.h&gt;

20891           int isgreater(real-floating x, real-floating y);

20892 **DESCRIPTION**

20893 cx       The functionality described on this reference page is aligned with the ISO C standard. Any  
20894       conflict between the requirements described here and the ISO C standard is unintentional. This  
20895       volume of IEEE Std 1003.1-200x defers to the ISO C standard.

20896       The *isgreater()* macro shall determine whether its first argument is greater than its second  
20897       argument. The value of *isgreater(x, y)* shall be equal to  $(x) > (y)$ ; however, unlike  $(x) > (y)$ ,  
20898       *isgreater(x, y)* shall not raise the invalid floating-point exception when *x* and *y* are unordered.

20899 **RETURN VALUE**20900       Upon successful completion, the *isgreater()* macro shall return the value of  $(x) > (y)$ .20901       If *x* or *y* is NaN, 0 shall be returned.20902 **ERRORS**

20903       No errors are defined.

20904 **EXAMPLES**

20905       None.

20906 **APPLICATION USAGE**

20907       The relational and equality operators support the usual mathematical relationships between  
20908       numeric values. For any ordered pair of numeric values, exactly one of the relationships (less,  
20909       greater, and equal) is true. Relational operators may raise the invalid floating-point exception  
20910       when argument values are NaNs. For a NaN and a numeric value, or for two NaNs, just the  
20911       unordered relationship is true. This macro is a quiet (non-floating-point exception raising)  
20912       version of a relational operator. It facilitates writing efficient code that accounts for NaNs  
20913       without suffering the invalid floating-point exception. In the SYNOPSIS section, **real-floating**  
20914       indicates that the argument shall be an expression of **real-floating** type.

20915 **RATIONALE**

20916       None.

20917 **FUTURE DIRECTIONS**

20918       None.

20919 **SEE ALSO**

20920       *isgreaterequal()*, *isless()*, *islessequal()*, *islessgreater()*, *isunordered()*, the Base Definitions volume of  
20921       IEEE Std 1003.1-200x <math.h>

20922 **CHANGE HISTORY**

20923       First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

20924 **NAME**

20925           isgreaterequal — test if *x* greater than or equal to *y*

20926 **SYNOPSIS**

20927           #include <math.h>

20928           int isgreaterequal(real-floating *x*, real-floating *y*);

20929 **DESCRIPTION**

20930 *cx*       The functionality described on this reference page is aligned with the ISO C standard. Any  
20931 conflict between the requirements described here and the ISO C standard is unintentional. This  
20932 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

20933       The *isgreaterequal()* macro shall determine whether its first argument is greater than or equal to  
20934 its second argument. The value of *isgreaterequal(x, y)* shall be equal to  $(x) \geq (y)$ ; however, unlike  
20935  $(x) \geq (y)$ , *isgreaterequal(x, y)* shall not raise the invalid floating-point exception when *x* and *y* are  
20936 unordered.

20937 **RETURN VALUE**

20938       Upon successful completion, the *isgreaterequal()* macro shall return the value of  $(x) \geq (y)$ .

20939       If *x* or *y* is NaN, 0 shall be returned.

20940 **ERRORS**

20941       No errors are defined.

20942 **EXAMPLES**

20943       None.

20944 **APPLICATION USAGE**

20945       The relational and equality operators support the usual mathematical relationships between  
20946 numeric values. For any ordered pair of numeric values, exactly one of the relationships (less,  
20947 greater, and equal) is true. Relational operators may raise the invalid floating-point exception  
20948 when argument values are NaNs. For a NaN and a numeric value, or for two NaNs, just the  
20949 unordered relationship is true. This macro is a quiet (non-floating-point exception raising)  
20950 version of a relational operator. It facilitates writing efficient code that accounts for NaNs  
20951 without suffering the invalid floating-point exception. In the SYNOPSIS section, **real-floating**  
20952 indicates that the argument shall be an expression of **real-floating** type.

20953 **RATIONALE**

20954       None.

20955 **FUTURE DIRECTIONS**

20956       None.

20957 **SEE ALSO**

20958       *isgreater()*, *isless()*, *islessequal()*, *islessgreater()*, *isunordered()*, the Base Definitions volume of  
20959 IEEE Std 1003.1-200x <math.h>

20960 **CHANGE HISTORY**

20961       First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

20962 **NAME**

20963           isinf — test for infinity

20964 **SYNOPSIS**

20965           #include &lt;math.h&gt;

20966           int isinf(real-floating x);

20967 **DESCRIPTION**

20968 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
20969 conflict between the requirements described here and the ISO C standard is unintentional. This  
20970 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

20971       The *isinf()* macro shall determine whether its argument value is an infinity (positive or  
20972 negative). First, an argument represented in a format wider than its semantic type is converted  
20973 to its semantic type. Then determination is based on the type of the argument.

20974 **RETURN VALUE**20975       The *isinf()* macro shall return a non-zero value if and only if its argument has an infinite value.20976 **ERRORS**

20977       No errors are defined.

20978 **EXAMPLES**

20979       None.

20980 **APPLICATION USAGE**

20981       None.

20982 **RATIONALE**

20983       None.

20984 **FUTURE DIRECTIONS**

20985       None.

20986 **SEE ALSO**

20987       *fpclassify()*, *isfinite()*, *isnan()*, *isnormal()*, *signbit()*, the Base Definitions volume of  
20988 IEEE Std 1003.1-200x <math.h>

20989 **CHANGE HISTORY**

20990       First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

20991 **NAME**

20992           isless — test if *x* is less than *y*

20993 **SYNOPSIS**

20994           #include <math.h>

20995           int isless(real-floating *x*, real-floating *y*);

20996 **DESCRIPTION**

20997 *cx*       The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-200x defers to the ISO C standard.

21000       The *isless()* macro shall determine whether its first argument is less than its second argument. The value of *isless(x, y)* shall be equal to  $(x) < (y)$ ; however, unlike  $(x) < (y)$ , *isless(x, y)* shall not raise the invalid floating-point exception when *x* and *y* are unordered.

21003 **RETURN VALUE**

21004       Upon successful completion, the *isless()* macro shall return the value of  $(x) < (y)$ .

21005       If *x* or *y* is NaN, 0 shall be returned.

21006 **ERRORS**

21007       No errors are defined.

21008 **EXAMPLES**

21009       None.

21010 **APPLICATION USAGE**

21011       The relational and equality operators support the usual mathematical relationships between numeric values. For any ordered pair of numeric values, exactly one of the relationships (less, greater, and equal) is true. Relational operators may raise the invalid floating-point exception when argument values are NaNs. For a NaN and a numeric value, or for two NaNs, just the unordered relationship is true. This macro is a quiet (non-floating-point exception raising) version of a relational operator. It facilitates writing efficient code that accounts for NaNs without suffering the invalid floating-point exception. In the SYNOPSIS section, **real-floating** indicates that the argument shall be an expression of **real-floating** type.

21019 **RATIONALE**

21020       None.

21021 **FUTURE DIRECTIONS**

21022       None.

21023 **SEE ALSO**

21024       *isgreater()*, *isgreaterequal()*, *islessequal()*, *islessgreater()*, *isunordered()*, the Base Definitions volume of IEEE Std 1003.1-200x, <math.h>

21026 **CHANGE HISTORY**

21027       First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

21028 **NAME**

21029 islessequal — test if *x* is less than or equal to *y*

21030 **SYNOPSIS**

21031 #include <math.h>

21032 int islessequal(real-floating *x*, real-floating *y*);

21033 **DESCRIPTION**

21034 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
21035 conflict between the requirements described here and the ISO C standard is unintentional. This  
21036 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

21037 The *islessequal()* macro shall determine whether its first argument is less than or equal to its  
21038 second argument. The value of *islessequal(x, y)* shall be equal to  $(x) \leq (y)$ ; however, unlike  
21039  $(x) \leq (y)$ , *islessequal(x, y)* shall not raise the invalid floating-point exception when *x* and *y* are  
21040 unordered.

21041 **RETURN VALUE**

21042 Upon successful completion, the *islessequal()* macro shall return the value of  $(x) \leq (y)$ .

21043 If *x* or *y* is NaN, 0 shall be returned.

21044 **ERRORS**

21045 No errors are defined.

21046 **EXAMPLES**

21047 None.

21048 **APPLICATION USAGE**

21049 The relational and equality operators support the usual mathematical relationships between  
21050 numeric values. For any ordered pair of numeric values, exactly one of the relationships (less,  
21051 greater, and equal) is true. Relational operators may raise the invalid floating-point exception  
21052 when argument values are NaNs. For a NaN and a numeric value, or for two NaNs, just the  
21053 unordered relationship is true. This macro is a quiet (non-floating-point exception raising)  
21054 version of a relational operator. It facilitates writing efficient code that accounts for NaNs  
21055 without suffering the invalid floating-point exception. In the SYNOPSIS section, **real-floating**  
21056 indicates that the argument shall be an expression of **real-floating** type.

21057 **RATIONALE**

21058 None.

21059 **FUTURE DIRECTIONS**

21060 None.

21061 **SEE ALSO**

21062 *isgreater()*, *isgreaterequal()*, *isless()*, *islessgreater()*, *isunordered()*, the Base Definitions volume of  
21063 IEEE Std 1003.1-200x <math.h>

21064 **CHANGE HISTORY**

21065 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

21066 **NAME**

21067 `islessgreater` — test if  $x$  is less than or greater than  $y$

21068 **SYNOPSIS**

21069 `#include <math.h>`

21070 `int islessgreater(real-floating  $x$ , real-floating  $y$ );`

21071 **DESCRIPTION**

21072 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
21073 conflict between the requirements described here and the ISO C standard is unintentional. This  
21074 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

21075 The `islessgreater()` macro shall determine whether its first argument is less than or greater than  
21076 its second argument. The `islessgreater( $x$ ,  $y$ )` macro is similar to  $(x) < (y) \mid \mid (x) > (y)$ ; however,  
21077 `islessgreater( $x$ ,  $y$ )` shall not raise the invalid floating-point exception when  $x$  and  $y$  are unordered  
21078 (nor shall it evaluate  $x$  and  $y$  twice).

21079 **RETURN VALUE**

21080 Upon successful completion, the `islessgreater()` macro shall return the value of  
21081  $(x) < (y) \mid \mid (x) > (y)$ .

21082 If  $x$  or  $y$  is NaN, 0 shall be returned.

21083 **ERRORS**

21084 No errors are defined.

21085 **EXAMPLES**

21086 None.

21087 **APPLICATION USAGE**

21088 The relational and equality operators support the usual mathematical relationships between  
21089 numeric values. For any ordered pair of numeric values, exactly one of the relationships (less,  
21090 greater, and equal) is true. Relational operators may raise the invalid floating-point exception  
21091 when argument values are NaNs. For a NaN and a numeric value, or for two NaNs, just the  
21092 unordered relationship is true. This macro is a quiet (non-floating-point exception raising)  
21093 version of a relational operator. It facilitates writing efficient code that accounts for NaNs  
21094 without suffering the invalid floating-point exception. In the SYNOPSIS section, **real-floating**  
21095 indicates that the argument shall be an expression of **real-floating** type.

21096 **RATIONALE**

21097 None.

21098 **FUTURE DIRECTIONS**

21099 None.

21100 **SEE ALSO**

21101 `isgreater()`, `isgreaterequal()`, `isless()`, `islessequal()`, `isunordered()`, the Base Definitions volume of  
21102 IEEE Std 1003.1-200x **<math.h>**

21103 **CHANGE HISTORY**

21104 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

21105 **NAME**

21106 islower — test for a lowercase letter

21107 **SYNOPSIS**

21108 #include &lt;ctype.h&gt;

21109 int islower(int c);

21110 **DESCRIPTION**

21111 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
21112 conflict between the requirements described here and the ISO C standard is unintentional. This  
21113 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

21114 The *islower()* function shall test whether *c* is a character of class **lower** in the program's current  
21115 locale; see the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 7, Locale.

21116 The *c* argument is an **int**, the value of which the application shall ensure is a character  
21117 representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has  
21118 any other value, the behavior is undefined.

21119 **RETURN VALUE**21120 The *islower()* function shall return non-zero if *c* is a lowercase letter; otherwise, it shall return 0.21121 **ERRORS**

21122 No errors are defined.

21123 **EXAMPLES**21124 **Testing for a Lowercase Letter**

21125 The following example tests whether the value is a lowercase letter, based on the locale of the  
21126 user, then uses it as part of a key value.

```
21127 #include <ctype.h>
21128 #include <stdlib.h>
21129 #include <locale.h>
21130 ...
21131 char *keyst;
21132 int elementlen, len;
21133 char c;
21134 ...
21135 setlocale(LC_ALL, "");
21136 ...
21137 len = 0;
21138 while (len < elementlen) {
21139     c = (char) (rand() % 256);
21140     ...
21141     if (islower(c))
21142         keyst[len++] = c;
21143 }
21144 ...
```

21145 **APPLICATION USAGE**

21146 To ensure applications portability, especially across natural languages, only this function and  
21147 those listed in the SEE ALSO section should be used for character classification.

21148 **RATIONALE**

21149 None.

21150 **FUTURE DIRECTIONS**

21151 None.

21152 **SEE ALSO**21153 *isalnum()*, *isalpha()*, *iscntrl()*, *isdigit()*, *isgraph()*, *isprint()*, *ispunct()*, *isspace()*, *isupper()*,21154 *isxdigit()*, *setlocale()*, the Base Definitions volume of IEEE Std 1003.1-200x, <ctype.h>, the Base

21155 Definitions volume of IEEE Std 1003.1-200x, Chapter 7, Locale

21156 **CHANGE HISTORY**

21157 First released in Issue 1. Derived from Issue 1 of the SVID.

21158 **Issue 6**

21159 The DESCRIPTION is updated to avoid use of the term “must” for application requirements. |

21160 An example is added. |

21161 **NAME**

21162           isnan — test for a NaN

21163 **SYNOPSIS**

21164           #include &lt;math.h&gt;

21165           int isnan(real-floating x);

21166 **DESCRIPTION**

21167 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
21168 conflict between the requirements described here and the ISO C standard is unintentional. This  
21169 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

21170       The *isnan()* macro shall determine whether its argument value is a NaN. First, an argument  
21171 represented in a format wider than its semantic type is converted to its semantic type. Then  
21172 determination is based on the type of the argument.

21173 **RETURN VALUE**21174       The *isnan()* macro shall return a non-zero value if and only if its argument has a NaN value.21175 **ERRORS**

21176       No errors are defined.

21177 **EXAMPLES**

21178       None.

21179 **APPLICATION USAGE**

21180       None.

21181 **RATIONALE**

21182       None.

21183 **FUTURE DIRECTIONS**

21184       None.

21185 **SEE ALSO**

21186       *fpclassify()*, *isfinite()*, *isinf()*, *isnormal()*, *signbit()*, the Base Definitions volume of  
21187 IEEE Std 1003.1-200x, <math.h>

21188 **CHANGE HISTORY**

21189       First released in Issue 3.

21190 **Issue 5**

21191       The DESCRIPTION is updated to indicate the return value when NaN is not supported. This  
21192 text was previously published in the APPLICATION USAGE section.

21193 **Issue 6**

21194       Entry re-written for alignment with the ISO/IEC 9899:1999 standard.

21195 **NAME**

21196           isnormal — test for a normal value

21197 **SYNOPSIS**

21198           #include <math.h>

21199           int isnormal(real-floating x);

21200 **DESCRIPTION**

21201 cx       The functionality described on this reference page is aligned with the ISO C standard. Any  
21202       conflict between the requirements described here and the ISO C standard is unintentional. This  
21203       volume of IEEE Std 1003.1-200x defers to the ISO C standard.

21204       The *isnormal()* macro shall determine whether its argument value is normal (neither zero,  
21205       subnormal, infinite, nor NaN). First, an argument represented in a format wider than its  
21206       semantic type is converted to its semantic type. Then determination is based on the type of the  
21207       argument.

21208 **RETURN VALUE**

21209       The *isnormal()* macro shall return a non-zero value if and only if its argument has a normal  
21210       value.

21211 **ERRORS**

21212       No errors are defined.

21213 **EXAMPLES**

21214       None.

21215 **APPLICATION USAGE**

21216       None.

21217 **RATIONALE**

21218       None.

21219 **FUTURE DIRECTIONS**

21220       None.

21221 **SEE ALSO**

21222       *fpclassify()*, *isfinite()*, *isinf()*, *isnan()*, *signbit()*, the Base Definitions volume of  
21223       IEEE Std 1003.1-200x, <math.h>

21224 **CHANGE HISTORY**

21225       First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

21226 **NAME**

21227 isprint — test for a printable character |

21228 **SYNOPSIS**

21229 #include &lt;ctype.h&gt;

21230 int isprint(int c);

21231 **DESCRIPTION**

21232 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
21233 conflict between the requirements described here and the ISO C standard is unintentional. This  
21234 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

21235 The *isprint()* function shall test whether *c* is a character of class **print** in the program's current  
21236 locale; see the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 7, Locale.

21237 The *c* argument is an **int**, the value of which the application shall ensure is a character  
21238 representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has  
21239 any other value, the behavior is undefined. |

21240 **RETURN VALUE**

21241 The *isprint()* function shall return non-zero if *c* is a printable character; otherwise, it shall return  
21242 0. |

21243 **ERRORS**

21244 No errors are defined.

21245 **EXAMPLES**

21246 None.

21247 **APPLICATION USAGE**

21248 To ensure applications portability, especially across natural languages, only this function and  
21249 those listed in the SEE ALSO section should be used for character classification.

21250 **RATIONALE**

21251 None.

21252 **FUTURE DIRECTIONS**

21253 None.

21254 **SEE ALSO**

21255 *isalnum()*, *isalpha()*, *iscntrl()*, *isdigit()*, *isgraph()*, *islower()*, *ispunct()*, *isspace()*, *isupper()*,  
21256 *isxdigit()*, *setlocale()*, the Base Definitions volume of IEEE Std 1003.1-200x, <ctype.h>, the Base  
21257 Definitions volume of IEEE Std 1003.1-200x, Chapter 7, Locale

21258 **CHANGE HISTORY**

21259 First released in Issue 1. Derived from Issue 1 of the SVID.

21260 **Issue 6**

21261 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

21262 **NAME**

21263           ispunct — test for a punctuation character

21264 **SYNOPSIS**

21265           #include &lt;ctype.h&gt;

21266           int ispunct(int c);

21267 **DESCRIPTION**

21268 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
21269 conflict between the requirements described here and the ISO C standard is unintentional. This  
21270 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

21271       The *ispunct()* function shall test whether *c* is a character of class **punct** in the program's current  
21272 locale; see the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 7, Locale.

21273       The *c* argument is an **int**, the value of which the application shall ensure is a character  
21274 representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has  
21275 any other value, the behavior is undefined.

21276 **RETURN VALUE**

21277       The *ispunct()* function shall return non-zero if *c* is a punctuation character; otherwise, it shall  
21278 return 0.

21279 **ERRORS**

21280       No errors are defined.

21281 **EXAMPLES**

21282       None.

21283 **APPLICATION USAGE**

21284       To ensure applications portability, especially across natural languages, only this function and  
21285 those listed in the SEE ALSO section should be used for character classification.

21286 **RATIONALE**

21287       None.

21288 **FUTURE DIRECTIONS**

21289       None.

21290 **SEE ALSO**

21291       *isalnum()*, *isalpha()*, *iscntrl()*, *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *isspace()*, *isupper()*, *isxdigit()*,  
21292 *setlocale()*, the Base Definitions volume of IEEE Std 1003.1-200x, <ctype.h>, the Base Definitions  
21293 volume of IEEE Std 1003.1-200x, Chapter 7, Locale

21294 **CHANGE HISTORY**

21295       First released in Issue 1. Derived from Issue 1 of the SVID.

21296 **Issue 6**

21297       The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

21298 **NAME**

21299           isspace — test for a white-space character

21300 **SYNOPSIS**

21301           #include <ctype.h>

21302           int isspace(int c);

21303 **DESCRIPTION**

21304 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
21305 conflict between the requirements described here and the ISO C standard is unintentional. This  
21306 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

21307       The *isspace()* function shall test whether *c* is a character of class **space** in the program's current  
21308 locale; see the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 7, Locale.

21309       The *c* argument is an **int**, the value of which the application shall ensure is a character  
21310 representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has  
21311 any other value, the behavior is undefined.

21312 **RETURN VALUE**

21313       The *isspace()* function shall return non-zero if *c* is a white-space character; otherwise, it shall  
21314 return 0.

21315 **ERRORS**

21316       No errors are defined.

21317 **EXAMPLES**

21318       None.

21319 **APPLICATION USAGE**

21320       To ensure applications portability, especially across natural languages, only this function and  
21321 those listed in the SEE ALSO section should be used for character classification.

21322 **RATIONALE**

21323       None.

21324 **FUTURE DIRECTIONS**

21325       None.

21326 **SEE ALSO**

21327       *isalnum()*, *isalpha()*, *iscntrl()*, *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isupper()*,  
21328 *isxdigit()*, *setlocale()*, the Base Definitions volume of IEEE Std 1003.1-200x, <ctype.h>, the Base  
21329 Definitions volume of IEEE Std 1003.1-200x, Chapter 7, Locale

21330 **CHANGE HISTORY**

21331       First released in Issue 1. Derived from Issue 1 of the SVID.

21332 **Issue 6**

21333       The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

21334 **NAME**

21335            **isunordered** — test if arguments are unordered

21336 **SYNOPSIS**

21337            #include <math.h>

21338            int isunordered(real-floating x, real-floating y);

21339 **DESCRIPTION**

21340 **CX**        The functionality described on this reference page is aligned with the ISO C standard. Any  
21341            conflict between the requirements described here and the ISO C standard is unintentional. This  
21342            volume of IEEE Std 1003.1-200x defers to the ISO C standard.

21343            The *isunordered()* macro shall determine whether its arguments are unordered.

21344 **RETURN VALUE**

21345            Upon successful completion, the *isunordered()* macro shall return 1 if its arguments are  
21346            unordered, and 0 otherwise.

21347            If *x* or *y* is NaN, 0 shall be returned.

21348 **ERRORS**

21349            No errors are defined.

21350 **EXAMPLES**

21351            None.

21352 **APPLICATION USAGE**

21353            The relational and equality operators support the usual mathematical relationships between  
21354            numeric values. For any ordered pair of numeric values, exactly one of the relationships (less,  
21355            greater, and equal) is true. Relational operators may raise the invalid floating-point exception  
21356            when argument values are NaNs. For a NaN and a numeric value, or for two NaNs, just the  
21357            unordered relationship is true. This macro is a quiet (non-floating-point exception raising)  
21358            version of a relational operator. It facilitates writing efficient code that accounts for NaNs  
21359            without suffering the invalid floating-point exception. In the SYNOPSIS section, **real-floating**  
21360            indicates that the argument shall be an expression of **real-floating** type.

21361 **RATIONALE**

21362            None.

21363 **FUTURE DIRECTIONS**

21364            None.

21365 **SEE ALSO**

21366            *isgreater()*, *isgreaterequal()*, *isless()*, *islessequal()*, *islessgreater()*, the Base Definitions volume of  
21367            IEEE Std 1003.1-200x, <math.h>

21368 **CHANGE HISTORY**

21369            First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

21370 **NAME**

21371           isupper — test for an uppercase letter

21372 **SYNOPSIS**

21373           #include <ctype.h>

21374           int isupper(int c);

21375 **DESCRIPTION**

21376 **cx**       The functionality described on this reference page is aligned with the ISO C standard. Any  
21377       conflict between the requirements described here and the ISO C standard is unintentional. This  
21378       volume of IEEE Std 1003.1-200x defers to the ISO C standard.

21379       The *isupper()* function shall test whether *c* is a character of class **upper** in the program's current  
21380       locale; see the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 7, Locale.

21381       The *c* argument is an **int**, the value of which the application shall ensure is a character  
21382       representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has  
21383       any other value, the behavior is undefined.

21384 **RETURN VALUE**

21385       The *isupper()* function shall return non-zero if *c* is an uppercase letter; otherwise, it shall return 0.

21386 **ERRORS**

21387       No errors are defined.

21388 **EXAMPLES**

21389       None.

21390 **APPLICATION USAGE**

21391       To ensure applications portability, especially across natural languages, only this function and  
21392       those listed in the SEE ALSO section should be used for character classification.

21393 **RATIONALE**

21394       None.

21395 **FUTURE DIRECTIONS**

21396       None.

21397 **SEE ALSO**

21398       *isalnum()*, *isalpha()*, *iscntrl()*, *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isspace()*, *isxdigit()*,  
21399       *setlocale()*, the Base Definitions volume of IEEE Std 1003.1-200x, <ctype.h>, the Base Definitions  
21400       volume of IEEE Std 1003.1-200x, Chapter 7, Locale

21401 **CHANGE HISTORY**

21402       First released in Issue 1. Derived from Issue 1 of the SVID.

21403 **Issue 6**

21404       The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

21405 **NAME**

21406           iswalnum — test for an alphanumeric wide-character code

21407 **SYNOPSIS**

21408           #include <wctype.h>

21409           int iswalnum(wint\_t wc);

21410 **DESCRIPTION**

21411 cx       The functionality described on this reference page is aligned with the ISO C standard. Any  
21412       conflict between the requirements described here and the ISO C standard is unintentional. This  
21413       volume of IEEE Std 1003.1-200x defers to the ISO C standard.

21414       The *iswalnum()* function shall test whether *wc* is a wide-character code representing a character  
21415       of class **alpha** or **digit** in the program's current locale; see the Base Definitions volume of  
21416       IEEE Std 1003.1-200x, Chapter 7, Locale.

21417       The *wc* argument is a **wint\_t**, the value of which the application shall ensure is a wide-character  
21418       code corresponding to a valid character in the current locale, or equal to the value of the macro  
21419       WEOF. If the argument has any other value, the behavior is undefined.

21420 **RETURN VALUE**

21421       The *iswalnum()* function shall return non-zero if *wc* is an alphanumeric wide-character code;  
21422       otherwise, it shall return 0.

21423 **ERRORS**

21424       No errors are defined.

21425 **EXAMPLES**

21426       None.

21427 **APPLICATION USAGE**

21428       To ensure applications portability, especially across natural languages, only this function and  
21429       those listed in the SEE ALSO section should be used for classification of wide-character codes.

21430 **RATIONALE**

21431       None.

21432 **FUTURE DIRECTIONS**

21433       None.

21434 **SEE ALSO**

21435       *iswalphabet()*, *iswcntrl()*, *iswctype()*, *iswdigit()*, *iswgraph()*, *iswlower()*, *iswprint()*, *iswpunct()*,  
21436       *iswspace()*, *iswupper()*, *iswxdigit()*, *setlocale()*, the Base Definitions volume of  
21437       IEEE Std 1003.1-200x, <wctype.h>, <wchar.h>, <stdio.h>, the Base Definitions volume of  
21438       IEEE Std 1003.1-200x, Chapter 7, Locale

21439 **CHANGE HISTORY**

21440       First released as a World-wide Portability Interface in Issue 4.

21441 **Issue 5**

21442       The following change has been made in this issue for alignment with  
21443       ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 21444       • The SYNOPSIS has been changed to indicate that this function and associated data types are  
21445       now made visible by inclusion of the <wctype.h> header rather than <wchar.h>.

21446 **Issue 6**

21447

The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

21448 **NAME**

21449           iswalpha — test for an alphabetic wide-character code

21450 **SYNOPSIS**

21451           #include <wctype.h>

21452           int iswalpha(wint\_t wc);

21453 **DESCRIPTION**

21454 **cx**       The functionality described on this reference page is aligned with the ISO C standard. Any  
21455 conflict between the requirements described here and the ISO C standard is unintentional. This  
21456 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

21457       The *iswalpha()* function shall test whether *wc* is a wide-character code representing a character of  
21458 class **alpha** in the program's current locale; see the Base Definitions volume of  
21459 IEEE Std 1003.1-200x, Chapter 7, Locale.

21460       The *wc* argument is a **wint\_t**, the value of which the application shall ensure is a wide-character  
21461 code corresponding to a valid character in the current locale, or equal to the value of the macro  
21462 WEOF. If the argument has any other value, the behavior is undefined.

21463 **RETURN VALUE**

21464       The *iswalpha()* function shall return non-zero if *wc* is an alphabetic wide-character code;  
21465 otherwise, it shall return 0.

21466 **ERRORS**

21467       No errors are defined.

21468 **EXAMPLES**

21469       None.

21470 **APPLICATION USAGE**

21471       To ensure applications portability, especially across natural languages, only this function and  
21472 those listed in the SEE ALSO section should be used for classification of wide-character codes.

21473 **RATIONALE**

21474       None.

21475 **FUTURE DIRECTIONS**

21476       None.

21477 **SEE ALSO**

21478       *iswalnum()*, *iswcntrl()*, *iswctype()*, *iswdigit()*, *iswgraph()*, *iswlower()*, *iswprint()*, *iswpunct()*,  
21479 *iswspace()*, *iswupper()*, *iswxdigit()*, *setlocale()*, the Base Definitions volume of  
21480 IEEE Std 1003.1-200x, <wctype.h>, <wchar.h>, <stdio.h>, the Base Definitions volume of  
21481 IEEE Std 1003.1-200x, Chapter 7, Locale

21482 **CHANGE HISTORY**

21483       First released in Issue 4.

21484 **Issue 5**

21485       The following change has been made in this issue for alignment with  
21486 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 21487       • The SYNOPSIS has been changed to indicate that this function and associated data types are  
21488       now made visible by inclusion of the <wctype.h> header rather than <wchar.h>.

21489 **Issue 6**

21490

The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

21491 **NAME**

21492           iswblank — test for a blank wide-character code

21493 **SYNOPSIS**

21494           #include <wctype.h>

21495           int iswblank(wint\_t wc);

21496 **DESCRIPTION**

21497 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
21498       conflict between the requirements described here and the ISO C standard is unintentional. This  
21499       volume of IEEE Std 1003.1-200x defers to the ISO C standard.

21500       The *iswblank()* function shall test whether *wc* is a wide-character code representing a character of  
21501       class **blank** in the program's current locale; see the Base Definitions volume of  
21502       IEEE Std 1003.1-200x, Chapter 7, Locale.

21503       The *wc* argument is a **wint\_t**, the value of which the application shall ensure is a wide-character  
21504       code corresponding to a valid character in the current locale, or equal to the value of the macro  
21505       WEOF. If the argument has any other value, the behavior is undefined.

21506 **RETURN VALUE**

21507       The *iswblank()* function shall return non-zero if *wc* is a blank wide-character code; otherwise, it  
21508       shall return 0.

21509 **ERRORS**

21510       No errors are defined.

21511 **EXAMPLES**

21512       None.

21513 **APPLICATION USAGE**

21514       To ensure applications portability, especially across natural languages, only this function and  
21515       those listed in the SEE ALSO section should be used for classification of wide-character codes.

21516 **RATIONALE**

21517       None.

21518 **FUTURE DIRECTIONS**

21519       None.

21520 **SEE ALSO**

21521       *iswalnum()*, *iswalpha()*, *iswcntrl()*, *iswctype()*, *iswdigit()*, *iswgraph()*, *iswlower()*, *iswprint()*,  
21522       *iswpunct()*, *iswspace()*, *iswupper()*, *iswxdigit()*, *setlocale()*, the Base Definitions volume of  
21523       IEEE Std 1003.1-200x, <wchar.h>, <wctype.h>, <stdio.h>

21524 **CHANGE HISTORY**

21525       First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

21526 **NAME**

21527           iswcntrl — test for a control wide-character code

21528 **SYNOPSIS**

21529           #include <wctype.h>

21530           int iswcntrl(wint\_t wc);

21531 **DESCRIPTION**

21532 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
21533 conflict between the requirements described here and the ISO C standard is unintentional. This  
21534 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

21535       The *iswcntrl()* function shall test whether *wc* is a wide-character code representing a character of  
21536 class **cntrl** in the program's current locale; see the Base Definitions volume of  
21537 IEEE Std 1003.1-200x, Chapter 7, Locale.

21538       The *wc* argument is a **wint\_t**, the value of which the application shall ensure is a wide-character  
21539 code corresponding to a valid character in the current locale, or equal to the value of the macro  
21540 WEOF. If the argument has any other value, the behavior is undefined.

21541 **RETURN VALUE**

21542       The *iswcntrl()* function shall return non-zero if *wc* is a control wide-character code; otherwise, it  
21543 shall return 0.

21544 **ERRORS**

21545       No errors are defined.

21546 **EXAMPLES**

21547       None.

21548 **APPLICATION USAGE**

21549       To ensure applications portability, especially across natural languages, only this function and  
21550 those listed in the SEE ALSO section should be used for classification of wide-character codes.

21551 **RATIONALE**

21552       None.

21553 **FUTURE DIRECTIONS**

21554       None.

21555 **SEE ALSO**

21556       *iswalnum()*, *iswalpha()*, *iswctype()*, *iswdigit()*, *iswgraph()*, *iswlower()*, *iswprint()*, *iswpunct()*,  
21557 *iswspace()*, *iswupper()*, *iswxdigit()*, *setlocale()*, the Base Definitions volume of  
21558 IEEE Std 1003.1-200x, <wctype.h>, <wchar.h>, the Base Definitions volume of  
21559 IEEE Std 1003.1-200x, Chapter 7, Locale

21560 **CHANGE HISTORY**

21561       First released in Issue 4.

21562 **Issue 5**

21563       The following change has been made in this issue for alignment with  
21564 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 21565       • The SYNOPSIS has been changed to indicate that this function and associated data types are  
21566       now made visible by inclusion of the <wctype.h> header rather than <wchar.h>.

21567 **Issue 6**

21568

The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

21569 **NAME**

21570 iswctype — test character for a specified class

21571 **SYNOPSIS**

21572 #include &lt;wctype.h&gt;

21573 int iswctype(wint\_t *wc*, wctype\_t *charclass*);21574 **DESCRIPTION**

21575 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 21576 conflict between the requirements described here and the ISO C standard is unintentional. This  
 21577 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

21578 The *iswctype()* function shall determine whether the wide-character code *wc* has the character  
 21579 class *charclass*, returning true or false. The *iswctype()* function is defined on WEOF and wide-  
 21580 character codes corresponding to the valid character encodings in the current locale. If the *wc*  
 21581 argument is not in the domain of the function, the result is undefined. If the value of *charclass* is  
 21582 invalid (that is, not obtained by a call to *wctype()* or *charclass* is invalidated by a subsequent call  
 21583 to *setlocale()* that has affected category *LC\_CTYPE*) the result is unspecified.

21584 **RETURN VALUE**

21585 The *iswctype()* function shall return non-zero (true) if and only if *wc* has the property described  
 21586 **CX** by *charclass*. If *charclass* is 0, *iswctype()* shall return 0.

21587 **ERRORS**

21588 No errors are defined.

21589 **EXAMPLES**21590 **Testing for a Valid Character**

```
21591 #include <wctype.h>
21592 ...
21593 int yes_or_no;
21594 wint_t wc;
21595 wctype_t valid_class;
21596 ...
21597 if ((valid_class=wctype("vowel")) == (wctype_t)0)
21598     /* Invalid character class. */
21599     yes_or_no=iswctype(wc,valid_class);
```

21600 **APPLICATION USAGE**

21601 The twelve strings "alnum", "alpha", "blank", "cntrl", "digit", "graph", "lower",  
 21602 "print", "punct", "space", "upper", and "xdigit" are reserved for the standard  
 21603 character classes. In the table below, the functions in the left column are equivalent to the  
 21604 functions in the right column.

21605	<i>iswalnum(wc)</i>	<i>iswctype(wc, wctype("alnum"))</i>
21606	<i>iswalpha(wc)</i>	<i>iswctype(wc, wctype("alpha"))</i>
21607	<i>iswblank(wc)</i>	<i>iswctype(wc, wctype("blank"))</i>
21608	<i>iswcntrl(wc)</i>	<i>iswctype(wc, wctype("cntrl"))</i>
21609	<i>iswdigit(wc)</i>	<i>iswctype(wc, wctype("digit"))</i>
21610	<i>iswgraph(wc)</i>	<i>iswctype(wc, wctype("graph"))</i>
21611	<i>iswlower(wc)</i>	<i>iswctype(wc, wctype("lower"))</i>
21612	<i>iswprint(wc)</i>	<i>iswctype(wc, wctype("print"))</i>
21613	<i>iswpunct(wc)</i>	<i>iswctype(wc, wctype("punct"))</i>
21614	<i>iswspace(wc)</i>	<i>iswctype(wc, wctype("space"))</i>

21615            `iswupper(wc)`      `iswctype(wc, wctype("upper"))`  
 21616            `iswxdigit(wc)`    `iswctype(wc, wctype("xdigit"))`

21617 **RATIONALE**

21618            None.

21619 **FUTURE DIRECTIONS**

21620            None.

21621 **SEE ALSO**

21622            `iswalnum()`, `iswalpha()`, `iswcntrl()`, `iswdigit()`, `iswgraph()`, `iswlower()`, `iswprint()`, `iswpunct()`,  
 21623            `iswspace()`, `iswupper()`, `iswxdigit()`, `setlocale()`, `wctype()`, the Base Definitions volume of  
 21624            IEEE Std 1003.1-200x, `<wctype.h>`, `<wchar.h>`

21625 **CHANGE HISTORY**

21626            First released as World-wide Portability Interfaces in Issue 4.

21627 **Issue 5**

21628            The following change has been made in this issue for alignment with  
 21629            ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 21630            • The SYNOPSIS has been changed to indicate that this function and associated data types are  
 21631            now made visible by inclusion of the `<wctype.h>` header rather than `<wchar.h>`.

21632 **Issue 6**

21633            The behavior of  $n=0$  is now described. |

21634            An example is added. |

21635            A new function, `iswblank()`, is added to the list in the APPLICATION USAGE. |

21636 **NAME**

21637 iswdigit — test for a decimal digit wide-character code

21638 **SYNOPSIS**

21639 #include &lt;wctype.h&gt;

21640 int iswdigit(wint\_t wc);

21641 **DESCRIPTION**

21642 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
21643 conflict between the requirements described here and the ISO C standard is unintentional. This  
21644 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

21645 The *iswdigit()* function shall test whether *wc* is a wide-character code representing a character of  
21646 class **digit** in the program's current locale; see the Base Definitions volume of  
21647 IEEE Std 1003.1-200x, Chapter 7, Locale.

21648 The *wc* argument is a **wint\_t**, the value of which the application shall ensure is a wide-character  
21649 code corresponding to a valid character in the current locale, or equal to the value of the macro  
21650 WEOF. If the argument has any other value, the behavior is undefined.

21651 **RETURN VALUE**

21652 The *iswdigit()* function shall return non-zero if *wc* is a decimal digit wide-character code;  
21653 otherwise, it shall return 0.

21654 **ERRORS**

21655 No errors are defined.

21656 **EXAMPLES**

21657 None.

21658 **APPLICATION USAGE**

21659 To ensure applications portability, especially across natural languages, only this function and  
21660 those listed in the SEE ALSO section should be used for classification of wide-character codes.

21661 **RATIONALE**

21662 None.

21663 **FUTURE DIRECTIONS**

21664 None.

21665 **SEE ALSO**

21666 *iswalnum()*, *iswalpha()*, *iswcntrl()*, *iswctype()*, *iswgraph()*, *iswlower()*, *iswprint()*, *iswpunct()*,  
21667 *iswspace()*, *iswupper()*, *iswxdigit()*, *setlocale()*, the Base Definitions volume of  
21668 IEEE Std 1003.1-200x, <wctype.h>, <wchar.h>

21669 **CHANGE HISTORY**

21670 First released in Issue 4.

21671 **Issue 5**

21672 The following change has been made in this issue for alignment with  
21673 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 21674 • The SYNOPSIS has been changed to indicate that this function and associated data types are  
21675 now made visible by inclusion of the <wctype.h> header rather than <wchar.h>.

21676 **Issue 6**

21677 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

21678 **NAME**

21679 iswgraph — test for a visible wide-character code

21680 **SYNOPSIS**

21681 #include &lt;wctype.h&gt;

21682 int iswgraph(wint\_t wc);

21683 **DESCRIPTION**

21684 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
 21685 conflict between the requirements described here and the ISO C standard is unintentional. This  
 21686 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

21687 The *iswgraph()* function shall test whether *wc* is a wide-character code representing a character  
 21688 of class **graph** in the program's current locale; see the Base Definitions volume of  
 21689 IEEE Std 1003.1-200x, Chapter 7, Locale.

21690 The *wc* argument is a **wint\_t**, the value of which the application shall ensure is a wide-character  
 21691 code corresponding to a valid character in the current locale, or equal to the value of the macro  
 21692 WEOF. If the argument has any other value, the behavior is undefined.

21693 **RETURN VALUE**

21694 The *iswgraph()* function shall return non-zero if *wc* is a wide-character code with a visible  
 21695 representation; otherwise, it shall return 0.

21696 **ERRORS**

21697 No errors are defined.

21698 **EXAMPLES**

21699 None.

21700 **APPLICATION USAGE**

21701 To ensure applications portability, especially across natural languages, only this function and  
 21702 those listed in the SEE ALSO section should be used for classification of wide-character codes.

21703 **RATIONALE**

21704 None.

21705 **FUTURE DIRECTIONS**

21706 None.

21707 **SEE ALSO**

21708 *iswalnum()*, *iswalpha()*, *iswcntrl()*, *iswctype()*, *iswdigit()*, *iswlower()*, *iswprint()*, *iswpunct()*,  
 21709 *iswspace()*, *iswupper()*, *iswxdigit()*, *setlocale()*, the Base Definitions volume of  
 21710 IEEE Std 1003.1-200x, <**wctype.h**>, <**wchar.h**>, the Base Definitions volume of  
 21711 IEEE Std 1003.1-200x, Chapter 7, Locale

21712 **CHANGE HISTORY**

21713 First released in Issue 4.

21714 **Issue 5**

21715 The following change has been made in this issue for alignment with  
 21716 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 21717 • The SYNOPSIS has been changed to indicate that this function and associated data types are  
 21718 now made visible by inclusion of the <**wctype.h**> header rather than <**wchar.h**>.

21719 **Issue 6**

21720

The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

21721 **NAME**

21722 iswlower — test for a lowercase letter wide-character code

21723 **SYNOPSIS**

21724 #include &lt;wctype.h&gt;

21725 int iswlower(wint\_t wc);

21726 **DESCRIPTION**

21727 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
 21728 conflict between the requirements described here and the ISO C standard is unintentional. This  
 21729 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

21730 The *iswlower()* function shall test whether *wc* is a wide-character code representing a character  
 21731 of class **lower** in the program's current locale; see the Base Definitions volume of  
 21732 IEEE Std 1003.1-200x, Chapter 7, Locale.

21733 The *wc* argument is a **wint\_t**, the value of which the application shall ensure is a wide-character  
 21734 code corresponding to a valid character in the current locale, or equal to the value of the macro  
 21735 WEOF. If the argument has any other value, the behavior is undefined.

21736 **RETURN VALUE**

21737 The *iswlower()* function shall return non-zero if *wc* is a lowercase letter wide-character code;  
 21738 otherwise, it shall return 0.

21739 **ERRORS**

21740 No errors are defined.

21741 **EXAMPLES**

21742 None.

21743 **APPLICATION USAGE**

21744 To ensure applications portability, especially across natural languages, only this function and  
 21745 those listed in the SEE ALSO section should be used for classification of wide-character codes.

21746 **RATIONALE**

21747 None.

21748 **FUTURE DIRECTIONS**

21749 None.

21750 **SEE ALSO**

21751 *iswalnum()*, *iswalpha()*, *iswcntrl()*, *iswctype()*, *iswdigit()*, *iswgraph()*, *iswprint()*, *iswpunct()*,  
 21752 *iswspace()*, *iswupper()*, *iswxdigit()*, *setlocale()*, the Base Definitions volume of  
 21753 IEEE Std 1003.1-200x, <wctype.h>, <wchar.h>, the Base Definitions volume of  
 21754 IEEE Std 1003.1-200x, Chapter 7, Locale

21755 **CHANGE HISTORY**

21756 First released in Issue 4.

21757 **Issue 5**

21758 The following change has been made in this issue for alignment with  
 21759 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 21760 • The SYNOPSIS has been changed to indicate that this function and associated data types are  
 21761 now made visible by inclusion of the <wctype.h> header rather than <wchar.h>.

21762 **Issue 6**

21763

The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

21764 **NAME**

21765 iswprint — test for a printable wide-character code |

21766 **SYNOPSIS**

21767 #include &lt;wctype.h&gt;

21768 int iswprint(wint\_t wc);

21769 **DESCRIPTION**

21770 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
 21771 conflict between the requirements described here and the ISO C standard is unintentional. This  
 21772 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

21773 The *iswprint()* function shall test whether *wc* is a wide-character code representing a character of  
 21774 class **print** in the program's current locale; see the Base Definitions volume of  
 21775 IEEE Std 1003.1-200x, Chapter 7, Locale.

21776 The *wc* argument is a **wint\_t**, the value of which the application shall ensure is a wide-character |  
 21777 code corresponding to a valid character in the current locale, or equal to the value of the macro  
 21778 WEOF. If the argument has any other value, the behavior is undefined.

21779 **RETURN VALUE**

21780 The *iswprint()* function shall return non-zero if *wc* is a printable wide-character code; otherwise, |  
 21781 it shall return 0. |

21782 **ERRORS**

21783 No errors are defined.

21784 **EXAMPLES**

21785 None.

21786 **APPLICATION USAGE**

21787 To ensure applications portability, especially across natural languages, only this function and  
 21788 those listed in the SEE ALSO section should be used for classification of wide-character codes.

21789 **RATIONALE**

21790 None.

21791 **FUTURE DIRECTIONS**

21792 None.

21793 **SEE ALSO**

21794 *iswalnum()*, *iswalpha()*, *iswcntrl()*, *iswctype()*, *iswdigit()*, *iswgraph()*, *iswlower()*, *iswpunct()*,  
 21795 *iswspace()*, *iswupper()*, *iswxdigit()*, *setlocale()*, the Base Definitions volume of  
 21796 IEEE Std 1003.1-200x, <wctype.h>, <wchar.h>, the Base Definitions volume of  
 21797 IEEE Std 1003.1-200x, Chapter 7, Locale

21798 **CHANGE HISTORY**

21799 First released in Issue 4.

21800 **Issue 5**

21801 The following change has been made in this issue for alignment with  
 21802 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 21803 • The SYNOPSIS has been changed to indicate that this function and associated data types are  
 21804 now made visible by inclusion of the <wctype.h> header rather than <wchar.h>.

21805 **Issue 6**

21806

The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

21807 **NAME**

21808           iswpunct — test for a punctuation wide-character code

21809 **SYNOPSIS**

21810           #include &lt;wctype.h&gt;

21811           int iswpunct(wint\_t wc);

21812 **DESCRIPTION**

21813 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
 21814           conflict between the requirements described here and the ISO C standard is unintentional. This  
 21815           volume of IEEE Std 1003.1-200x defers to the ISO C standard.

21816       The *iswpunct()* function shall test whether *wc* is a wide-character code representing a character  
 21817       of class **punct** in the program's current locale; see the Base Definitions volume of  
 21818       IEEE Std 1003.1-200x, Chapter 7, Locale.

21819       The *wc* argument is a **wint\_t**, the value of which the application shall ensure is a wide-character  
 21820       code corresponding to a valid character in the current locale, or equal to the value of the macro  
 21821       WEOF. If the argument has any other value, the behavior is undefined.

21822 **RETURN VALUE**

21823       The *iswpunct()* function shall return non-zero if *wc* is a punctuation wide-character code;  
 21824       otherwise, it shall return 0.

21825 **ERRORS**

21826       No errors are defined.

21827 **EXAMPLES**

21828       None.

21829 **APPLICATION USAGE**

21830       To ensure applications portability, especially across natural languages, only this function and  
 21831       those listed in the SEE ALSO section should be used for classification of wide-character codes.

21832 **RATIONALE**

21833       None.

21834 **FUTURE DIRECTIONS**

21835       None.

21836 **SEE ALSO**

21837       *iswalnum()*, *iswalpha()*, *iswcntrl()*, *iswctype()*, *iswdigit()*, *iswgraph()*, *iswlower()*, *iswprint()*,  
 21838       *iswspace()*, *iswupper()*, *iswxdigit()*, *setlocale()*, the Base Definitions volume of  
 21839       IEEE Std 1003.1-200x, <wctype.h>, <wchar.h>, the Base Definitions volume of  
 21840       IEEE Std 1003.1-200x, Chapter 7, Locale

21841 **CHANGE HISTORY**

21842       First released in Issue 4.

21843 **Issue 5**

21844       The following change has been made in this issue for alignment with  
 21845       ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 21846       • The SYNOPSIS has been changed to indicate that this function and associated data types are  
 21847       now made visible by inclusion of the <wctype.h> header rather than <wchar.h>.

21848 **Issue 6**

21849 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

21850 **NAME**

21851           iswspace — test for a white-space wide-character code

21852 **SYNOPSIS**

21853           #include &lt;wctype.h&gt;

21854           int iswspace(wint\_t wc);

21855 **DESCRIPTION**

21856 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
 21857 conflict between the requirements described here and the ISO C standard is unintentional. This  
 21858 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

21859       The *iswspace()* function shall test whether *wc* is a wide-character code representing a character of  
 21860 class **space** in the program's current locale; see the Base Definitions volume of  
 21861 IEEE Std 1003.1-200x, Chapter 7, Locale.

21862       The *wc* argument is a **wint\_t**, the value of which the application shall ensure is a wide-character  
 21863 code corresponding to a valid character in the current locale, or equal to the value of the macro  
 21864 WEOF. If the argument has any other value, the behavior is undefined.

21865 **RETURN VALUE**

21866       The *iswspace()* function shall return non-zero if *wc* is a white-space wide-character code;  
 21867 otherwise, it shall return 0.

21868 **ERRORS**

21869       No errors are defined.

21870 **EXAMPLES**

21871       None.

21872 **APPLICATION USAGE**

21873       To ensure applications portability, especially across natural languages, only this function and  
 21874 those listed in the SEE ALSO section should be used for classification of wide-character codes.

21875 **RATIONALE**

21876       None.

21877 **FUTURE DIRECTIONS**

21878       None.

21879 **SEE ALSO**

21880       *iswalnum()*, *iswalpha()*, *iswcntrl()*, *iswctype()*, *iswdigit()*, *iswgraph()*, *iswlower()*, *iswprint()*,  
 21881 *iswpunct()*, *iswupper()*, *iswxdigit()*, *setlocale()*, the Base Definitions volume of  
 21882 IEEE Std 1003.1-200x, <wctype.h>, <wchar.h>, the Base Definitions volume of  
 21883 IEEE Std 1003.1-200x, Chapter 7, Locale

21884 **CHANGE HISTORY**

21885       First released in Issue 4.

21886 **Issue 5**

21887       The following change has been made in this issue for alignment with  
 21888 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 21889       • The SYNOPSIS has been changed to indicate that this function and associated data types are  
 21890       now made visible by inclusion of the <wctype.h> header rather than <wchar.h>.

21891 **Issue 6**

21892 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

21893 **NAME**

21894           iswupper — test for an uppercase letter wide-character code

21895 **SYNOPSIS**

21896           #include &lt;wctype.h&gt;

21897           int iswupper(wint\_t wc);

21898 **DESCRIPTION**

21899 cx       The functionality described on this reference page is aligned with the ISO C standard. Any  
 21900       conflict between the requirements described here and the ISO C standard is unintentional. This  
 21901       volume of IEEE Std 1003.1-200x defers to the ISO C standard.

21902       The *iswupper()* function shall test whether *wc* is a wide-character code representing a character  
 21903       of class **upper** in the program's current locale; see the Base Definitions volume of  
 21904       IEEE Std 1003.1-200x, Chapter 7, Locale.

21905       The *wc* argument is a **wint\_t**, the value of which the application shall ensure is a wide-character  
 21906       code corresponding to a valid character in the current locale, or equal to the value of the macro  
 21907       WEOF. If the argument has any other value, the behavior is undefined.

21908 **RETURN VALUE**

21909       The *iswupper()* function shall return non-zero if *wc* is an uppercase letter wide-character code;  
 21910       otherwise, it shall return 0.

21911 **ERRORS**

21912           No errors are defined.

21913 **EXAMPLES**

21914           None.

21915 **APPLICATION USAGE**

21916       To ensure applications portability, especially across natural languages, only this function and  
 21917       those listed in the SEE ALSO section should be used for classification of wide-character codes.

21918 **RATIONALE**

21919           None.

21920 **FUTURE DIRECTIONS**

21921           None.

21922 **SEE ALSO**

21923       *iswalnum()*, *iswalpha()*, *iswcntrl()*, *iswctype()*, *iswdigit()*, *iswgraph()*, *iswlower()*, *iswprint()*,  
 21924       *iswpunct()*, *iswspace()*, *iswxdigit()*, *setlocale()*, the Base Definitions volume of  
 21925       IEEE Std 1003.1-200x, <wctype.h>, <wchar.h>, the Base Definitions volume of  
 21926       IEEE Std 1003.1-200x, Chapter 7, Locale

21927 **CHANGE HISTORY**

21928           First released in Issue 4.

21929 **Issue 5**

21930       The following change has been made in this issue for alignment with  
 21931       ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 21932           • The SYNOPSIS has been changed to indicate that this function and associated data types are  
 21933           now made visible by inclusion of the <wctype.h> header rather than <wchar.h>.

21934 **Issue 6**

21935

The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

21936 **NAME**

21937 iswxdigit — test for a hexadecimal digit wide-character code

21938 **SYNOPSIS**

21939 #include &lt;wctype.h&gt;

21940 int iswxdigit(wint\_t wc);

21941 **DESCRIPTION**

21942 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
 21943 conflict between the requirements described here and the ISO C standard is unintentional. This  
 21944 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

21945 The *iswxdigit()* function shall test whether *wc* is a wide-character code representing a character  
 21946 of class **xdigit** in the program's current locale; see the Base Definitions volume of  
 21947 IEEE Std 1003.1-200x, Chapter 7, Locale.

21948 The *wc* argument is a **wint\_t**, the value of which the application shall ensure is a wide-character  
 21949 code corresponding to a valid character in the current locale, or equal to the value of the macro  
 21950 WEOF. If the argument has any other value, the behavior is undefined.

21951 **RETURN VALUE**

21952 The *iswxdigit()* function shall return non-zero if *wc* is a hexadecimal digit wide-character code;  
 21953 otherwise, it shall return 0.

21954 **ERRORS**

21955 No errors are defined.

21956 **EXAMPLES**

21957 None.

21958 **APPLICATION USAGE**

21959 To ensure applications portability, especially across natural languages, only this function and  
 21960 those listed in the SEE ALSO section should be used for classification of wide-character codes.

21961 **RATIONALE**

21962 None.

21963 **FUTURE DIRECTIONS**

21964 None.

21965 **SEE ALSO**

21966 *iswalnum()*, *iswalpha()*, *iswcntrl()*, *iswctype()*, *iswdigit()*, *iswgraph()*, *iswlower()*, *iswprint()*,  
 21967 *iswpunct()*, *iswspace()*, *iswupper()*, *setlocale()*, the Base Definitions volume of  
 21968 IEEE Std 1003.1-200x, <wctype.h>, <wchar.h>

21969 **CHANGE HISTORY**

21970 First released in Issue 4.

21971 **Issue 5**

21972 The following change has been made in this issue for alignment with  
 21973 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 21974 • The SYNOPSIS has been changed to indicate that this function and associated data types are  
 21975 now made visible by inclusion of the <wctype.h> header rather than <wchar.h>.

21976 **Issue 6**

21977 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

21978 **NAME**

21979 isxdigit — test for a hexadecimal digit

21980 **SYNOPSIS**

21981 #include &lt;ctype.h&gt;

21982 int isxdigit(int c);

21983 **DESCRIPTION**

21984 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
21985 conflict between the requirements described here and the ISO C standard is unintentional. This  
21986 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

21987 The *isxdigit()* function shall test whether *c* is a character of class **xdigit** in the program's current  
21988 locale; see the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 7, Locale.

21989 The *c* argument is an **int**, the value of which the application shall ensure is a character  
21990 representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has  
21991 any other value, the behavior is undefined.

21992 **RETURN VALUE**

21993 The *isxdigit()* function shall return non-zero if *c* is a hexadecimal digit; otherwise, it shall return  
21994 0.

21995 **ERRORS**

21996 No errors are defined.

21997 **EXAMPLES**

21998 None.

21999 **APPLICATION USAGE**

22000 To ensure applications portability, especially across natural languages, only this function and  
22001 those listed in the SEE ALSO section should be used for character classification.

22002 **RATIONALE**

22003 None.

22004 **FUTURE DIRECTIONS**

22005 None.

22006 **SEE ALSO**

22007 *isalnum()*, *isalpha()*, *iscntrl()*, *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isspace()*, *isupper()*,  
22008 the Base Definitions volume of IEEE Std 1003.1-200x, <ctype.h>

22009 **CHANGE HISTORY**

22010 First released in Issue 1. Derived from Issue 1 of the SVID.

22011 **Issue 6**

22012 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

22013 **NAME**

22014 j0, j1, jn — Bessel functions of the first kind

22015 **SYNOPSIS**

```
22016 xSI #include <math.h>
22017 double j0(double x);
22018 double j1(double x);
22019 double jn(int n, double x);
22020
```

22021 **DESCRIPTION**

22022 The *j0()*, *j1()*, and *jn()* functions shall compute Bessel functions of *x* of the first kind of orders 0,  
 22023 1, and *n*, respectively.

22024 An application wishing to check for error situations should set *errno* to zero and call  
 22025 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 22026 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 22027 zero, an error has occurred.

22028 **RETURN VALUE**

22029 Upon successful completion, these functions shall return the relevant Bessel value of *x* of the  
 22030 first kind.

22031 If the *x* argument is too large in magnitude, or the correct result would cause underflow, 0 shall  
 22032 be returned and a range error may occur.

22033 If *x* is NaN, a NaN shall be returned.

22034 **ERRORS**

22035 These functions may fail if:

22036 Range Error The value of *x* was too large in magnitude, or an underflow occurred.

22037 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero, |  
 22038 then *errno* shall be set to [ERANGE]. If the integer expression |  
 22039 (math\_errhandling & MATH\_ERREXCEPT) is non-zero, then the underflow |  
 22040 floating-point exception shall be raised. |

22041 No other errors shall occur.

22042 **EXAMPLES**

22043 None.

22044 **APPLICATION USAGE**

22045 On error, the expressions (math\_errhandling & MATH\_ERRNO) and (math\_errhandling &  
 22046 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

22047 **RATIONALE**

22048 None.

22049 **FUTURE DIRECTIONS**

22050 None.

22051 **SEE ALSO**

22052 *feclearexcept()*, *fetestexcept()*, *isnan()*, *y0()*, the Base Definitions volume of IEEE Std 1003.1-200x, |  
 22053 Section 4.18, Treatment of Error Conditions for Mathematical Functions, <math.h> |

22054 **CHANGE HISTORY**

22055 First released in Issue 1. Derived from Issue 1 of the SVID.

22056 **Issue 5**

22057 The DESCRIPTION is updated to indicate how an application should check for an error. This  
22058 text was previously published in the APPLICATION USAGE section.

22059 **Issue 6**

22060 The may fail [EDOM] error is removed for the case for NaN.

22061 The RETURN VALUE and ERRORS sections are reworked for alignment of the error handling |  
22062 with the ISO/IEC 9899:1999 standard.

22063 **NAME**

22064           jrand48 — generate a uniformly distributed pseudo-random long signed integer

22065 **SYNOPSIS**

22066 xSI       #include &lt;stdlib.h&gt;

22067           long jrand48(unsigned short xsubi[3]);

22068

22069 **DESCRIPTION**22070           Refer to *drand48()*.

## 22071 NAME

22072 kill — send a signal to a process or a group of processes

## 22073 SYNOPSIS

22074 cx #include &lt;signal.h&gt;

22075 int kill(pid\_t pid, int sig);

22076

## 22077 DESCRIPTION

22078 The *kill()* function shall send a signal to a process or a group of processes specified by *pid*. The  
 22079 signal to be sent is specified by *sig* and is either one from the list given in <signal.h> or 0. If *sig* is  
 22080 0 (the null signal), error checking is performed but no signal is actually sent. The null signal can  
 22081 be used to check the validity of *pid*.

22082 For a process to have permission to send a signal to a process designated by *pid*, unless the  
 22083 sending process has appropriate privileges, the real or effective user ID of the sending process  
 22084 shall match the real or saved set-user-ID of the receiving process.

22085 If *pid* is greater than 0, *sig* shall be sent to the process whose process ID is equal to *pid*.

22086 If *pid* is 0, *sig* shall be sent to all processes (excluding an unspecified set of system processes)  
 22087 whose process group ID is equal to the process group ID of the sender, and for which the  
 22088 process has permission to send a signal.

22089 If *pid* is  $-1$ , *sig* shall be sent to all processes (excluding an unspecified set of system processes) for  
 22090 which the process has permission to send that signal.

22091 If *pid* is negative, but not  $-1$ , *sig* shall be sent to all processes (excluding an unspecified set of  
 22092 system processes) whose process group ID is equal to the absolute value of *pid*, and for which  
 22093 the process has permission to send a signal.

22094 If the value of *pid* causes *sig* to be generated for the sending process, and if *sig* is not blocked for  
 22095 the calling thread and if no other thread has *sig* unblocked or is waiting in a *sigwait()* function  
 22096 for *sig*, either *sig* or at least one pending unblocked signal shall be delivered to the sending  
 22097 thread before *kill()* returns.

22098 The user ID tests described above shall not be applied when sending SIGCONT to a process that  
 22099 is a member of the same session as the sending process.

22100 An implementation that provides extended security controls may impose further  
 22101 implementation-defined restrictions on the sending of signals, including the null signal. In  
 22102 particular, the system may deny the existence of some or all of the processes specified by *pid*.

22103 The *kill()* function is successful if the process has permission to send *sig* to any of the processes  
 22104 specified by *pid*. If *kill()* fails, no signal shall be sent.

## 22105 RETURN VALUE

22106 Upon successful completion, 0 shall be returned. Otherwise,  $-1$  shall be returned and *errno* set to  
 22107 indicate the error.

## 22108 ERRORS

22109 The *kill()* function shall fail if:

22110 [EINVAL] The value of the *sig* argument is an invalid or unsupported signal number.

22111 [EPERM] The process does not have permission to send the signal to any receiving  
 22112 process.

22113 [ESRCH] No process or process group can be found corresponding to that specified by  
 22114 *pid*.

22115 **EXAMPLES**

22116 None.

22117 **APPLICATION USAGE**

22118 None.

22119 **RATIONALE**

22120 The semantics for permission checking for *kill()* differed between System V and most other  
 22121 implementations, such as Version 7 or 4.3 BSD. The semantics chosen for this volume of  
 22122 IEEE Std 1003.1-200x agree with System V. Specifically, a set-user-ID process cannot protect  
 22123 itself against signals (or at least not against SIGKILL) unless it changes its real user ID. This  
 22124 choice allows the user who starts an application to send it signals even if it changes its effective  
 22125 user ID. The other semantics give more power to an application that wants to protect itself from  
 22126 the user who ran it.

22127 Some implementations provide semantic extensions to the *kill()* function when the absolute  
 22128 value of *pid* is greater than some maximum, or otherwise special, value. Negative values are a  
 22129 flag to *kill()*. Since most implementations return [ESRCH] in this case, this behavior is not  
 22130 included in this volume of IEEE Std 1003.1-200x, although a conforming implementation could  
 22131 provide such an extension.

22132 The implementation-defined processes to which a signal cannot be sent may include the  
 22133 scheduler or *init*.

22134 There was initially strong sentiment to specify that, if *pid* specifies that a signal be sent to the  
 22135 calling process and that signal is not blocked, that signal would be delivered before *kill()*  
 22136 returns. This would permit a process to call *kill()* and be guaranteed that the call never return.  
 22137 However, historical implementations that provide only the *signal()* function make only the  
 22138 weaker guarantee in this volume of IEEE Std 1003.1-200x, because they only deliver one signal  
 22139 each time a process enters the kernel. Modifications to such implementations to support the  
 22140 *sigaction()* function generally require entry to the kernel following return from a signal-catching  
 22141 function, in order to restore the signal mask. Such modifications have the effect of satisfying the  
 22142 stronger requirement, at least when *sigaction()* is used, but not necessarily when *signal()* is used.  
 22143 The developers of this volume of IEEE Std 1003.1-200x considered making the stronger  
 22144 requirement except when *signal()* is used, but felt this would be unnecessarily complex.  
 22145 Implementors are encouraged to meet the stronger requirement whenever possible. In practice,  
 22146 the weaker requirement is the same, except in the rare case when two signals arrive during a  
 22147 very short window. This reasoning also applies to a similar requirement for *sigprocmask()*.

22148 In 4.2 BSD, the SIGCONT signal can be sent to any descendant process regardless of user-ID  
 22149 security checks. This allows a job control shell to continue a job even if processes in the job have  
 22150 altered their user IDs (as in the *su* command). In keeping with the addition of the concept of  
 22151 sessions, similar functionality is provided by allowing the SIGCONT signal to be sent to any  
 22152 process in the same session regardless of user ID security checks. This is less restrictive than BSD  
 22153 in the sense that ancestor processes (in the same session) can now be the recipient. It is more  
 22154 restrictive than BSD in the sense that descendant processes that form new sessions are now  
 22155 subject to the user ID checks. A similar relaxation of security is not necessary for the other job  
 22156 control signals since those signals are typically sent by the terminal driver in recognition of  
 22157 special characters being typed; the terminal driver bypasses all security checks.

22158 In secure implementations, a process may be restricted from sending a signal to a process having  
 22159 a different security label. In order to prevent the existence or nonexistence of a process from  
 22160 being used as a covert channel, such processes should appear nonexistent to the sender; that is,  
 22161 [ESRCH] should be returned, rather than [EPERM], if *pid* refers only to such processes.

22162 Existing implementations vary on the result of a *kill()* with *pid* indicating an inactive process (a  
 22163 terminated process that has not been waited for by its parent). Some indicate success on such a  
 22164 call (subject to permission checking), while others give an error of [ESRCH]. Since the definition  
 22165 of process lifetime in this volume of IEEE Std 1003.1-200x covers inactive processes, the  
 22166 [ESRCH] error as described is inappropriate in this case. In particular, this means that an  
 22167 application cannot have a parent process check for termination of a particular child with *kill()*.  
 22168 (Usually this is done with the null signal; this can be done reliably with *waitpid()*.)

22169 There is some belief that the name *kill()* is misleading, since the function is not always intended  
 22170 to cause process termination. However, the name is common to all historical implementations,  
 22171 and any change would be in conflict with the goal of minimal changes to existing application  
 22172 code.

#### 22173 FUTURE DIRECTIONS

22174 None.

#### 22175 SEE ALSO

22176 *getpid()*, *raise()*, *setsid()*, *sigaction()*, *sigqueue()*, the Base Definitions volume of  
 22177 IEEE Std 1003.1-200x, <signal.h>, <sys/types.h>

#### 22178 CHANGE HISTORY

22179 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 22180 Issue 5

22181 The DESCRIPTION is updated for alignment with POSIX Threads Extension.

#### 22182 Issue 6

22183 In the SYNOPSIS, the optional include of the <sys/types.h> header is removed.

22184 The following new requirements on POSIX implementations derive from alignment with the  
 22185 Single UNIX Specification:

- 22186 • In the DESCRIPTION, the second paragraph is reworded to indicate that the saved set-user-  
 22187 ID of the calling process is checked in place of its effective user ID. This is a FIPS  
 22188 requirement.
- 22189 • The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was  
 22190 required for conforming implementations of previous POSIX specifications, it was not  
 22191 required for UNIX applications.
- 22192 • The behavior when *pid* is  $-1$  is now specified. It was previously explicitly unspecified in the  
 22193 POSIX.1-1988 standard.

22194 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

22195 **NAME**

22196 killpg — send a signal to a process group

22197 **SYNOPSIS**

22198 XSI #include &lt;signal.h&gt;

22199 int killpg(pid\_t pgrp, int sig);

22200

22201 **DESCRIPTION**22202 The *killpg()* function shall send the signal specified by *sig* to the process group specified by *pgrp*.22203 If *pgrp* is greater than 1, *killpg(pgrp, sig)* shall be equivalent to *kill(-pgrp, sig)*. If *pgrp* is less than or  
22204 equal to 1, the behavior of *killpg()* is undefined.22205 **RETURN VALUE**22206 Refer to *kill()*.22207 **ERRORS**22208 Refer to *kill()*.22209 **EXAMPLES**

22210 None.

22211 **APPLICATION USAGE**

22212 None.

22213 **RATIONALE**

22214 None.

22215 **FUTURE DIRECTIONS**

22216 None.

22217 **SEE ALSO**22218 *getpgid()*, *getpid()*, *kill()*, *raise()*, the Base Definitions volume of IEEE Std 1003.1-200x, <signal.h>22219 **CHANGE HISTORY**

22220 First released in Issue 4, Version 2.

22221 **Issue 5**

22222 Moved from X/OPEN UNIX extension to BASE.

22223 **NAME**

22224       l64a — convert a 32-bit integer to a radix-64 ASCII string

22225 **SYNOPSIS**

22226 XSI     #include <stdlib.h>

22227       char \*l64a(long value);

22228

22229 **DESCRIPTION**

22230       Refer to *a64l()*.

22231 **NAME**

22232 labs, llabs — return a long integer absolute value

22233 **SYNOPSIS**

22234 #include <stdlib.h>

22235 long labs(long *i*);

22236 long long llabs(long long *i*);

22237 **DESCRIPTION**

22238 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
22239 conflict between the requirements described here and the ISO C standard is unintentional. This  
22240 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

22241 The *labs()* function shall compute the absolute value of the **long** integer operand *i*. The *llabs()*  
22242 function shall compute the absolute value of the **long long** integer operand *i*. If the result cannot  
22243 be represented, the behavior is undefined.

22244 **RETURN VALUE**

22245 The *labs()* function shall return the absolute value of the **long** integer operand. The *llabs()*  
22246 function shall return the absolute value of the **long long** integer operand.

22247 **ERRORS**

22248 No errors are defined.

22249 **EXAMPLES**

22250 None.

22251 **APPLICATION USAGE**

22252 None.

22253 **RATIONALE**

22254 None.

22255 **FUTURE DIRECTIONS**

22256 None.

22257 **SEE ALSO**

22258 *abs()*, the Base Definitions volume of IEEE Std 1003.1-200x, <stdlib.h>

22259 **CHANGE HISTORY**

22260 First released in Issue 4. Derived from the ISO C standard.

22261 **Issue 6**

22262 The *llabs()* function is added for alignment with the ISO/IEC 9899:1999 standard.

22263 **NAME**

22264 lchown — change the owner and group of a symbolic link

22265 **SYNOPSIS**

22266 XSI #include &lt;unistd.h&gt;

22267 int lchown(const char \*path, uid\_t owner, gid\_t group);

22268

22269 **DESCRIPTION**

22270 The *lchown()* function shall be equivalent to *chown()*, except in the case where the named file is a  
 22271 symbolic link. In this case, *lchown()* shall change the ownership of the symbolic link file itself,  
 22272 while *chown()* changes the ownership of the file or directory to which the symbolic link refers.

22273 **RETURN VALUE**

22274 Upon successful completion, *lchown()* shall return 0. Otherwise, it shall return -1 and set *errno* to  
 22275 indicate an error.

22276 **ERRORS**22277 The *lchown()* function shall fail if:22278 [EACCES] Search permission is denied on a component of the path prefix of *path*.

22279 [EINVAL] The owner or group ID is not a value supported by the implementation.

22280 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*  
 22281 argument.

22282 [ENAMETOOLONG]

22283 The length of a pathname exceeds {PATH\_MAX} or a pathname component is  
 22284 longer than {NAME\_MAX}.

22285 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.22286 [ENOTDIR] A component of the path prefix of *path* is not a directory.

22287 [EOPNOTSUPP] The *path* argument names a symbolic link and the implementation does not  
 22288 support setting the owner or group of a symbolic link.

22289 [EPERM] The effective user ID does not match the owner of the file and the process  
 22290 does not have appropriate privileges.

22291 [EROFS] The file resides on a read-only file system.

22292 The *lchown()* function may fail if:

22293 [EIO] An I/O error occurred while reading or writing to the file system.

22294 [INTR] A signal was caught during execution of the function.

22295 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
 22296 resolution of the *path* argument.

22297 [ENAMETOOLONG]

22298 Pathname resolution of a symbolic link produced an intermediate result  
 22299 whose length exceeds {PATH\_MAX}.

22300 **EXAMPLES**22301 **Changing the Current Owner of a File**

22302 The following example shows how to change the ownership of the symbolic link named  
22303 **/modules/pass1** to the user ID associated with “jones” and the group ID associated with “cnd”.

22304 The numeric value for the user ID is obtained by using the *getpwnam()* function. The numeric  
22305 value for the group ID is obtained by using the *getgrnam()* function.

```
22306 #include <sys/types.h>
22307 #include <unistd.h>
22308 #include <pwd.h>
22309 #include <grp.h>

22310 struct passwd *pwd;
22311 struct group *grp;
22312 char *path = "/modules/pass1";
22313 ...
22314 pwd = getpwnam("jones");
22315 grp = getgrnam("cnd");
22316 lchown(path, pwd->pw_uid, grp->gr_gid);
```

22317 **APPLICATION USAGE**

22318 None.

22319 **RATIONALE**

22320 None.

22321 **FUTURE DIRECTIONS**

22322 None.

22323 **SEE ALSO**

22324 *chown()*, *symlink()*, the Base Definitions volume of IEEE Std 1003.1-200x, **<unistd.h>**

22325 **CHANGE HISTORY**

22326 First released in Issue 4, Version 2.

22327 **Issue 5**

22328 Moved from X/OPEN UNIX extension to BASE.

22329 **Issue 6**

22330 The wording of the mandatory [ELOOP] error condition is updated, and a second optional  
22331 [ELOOP] error condition is added.

22332 **NAME**

22333        lcong48 — seed a uniformly distributed pseudo-random signed long integer generator

22334 **SYNOPSIS**

22335 xSI     #include <stdlib.h>

22336        void lcong48(unsigned short param[7]);

22337

22338 **DESCRIPTION**

22339        Refer to *drand48()*.

22340 **NAME**

22341 ldexp, ldexpf, ldexpl — load exponent of a floating-point number

22342 **SYNOPSIS**

22343 #include &lt;math.h&gt;

22344 double ldexp(double x, int exp);

22345 float ldexpf(float x, int exp);

22346 long double ldexpl(long double x, int exp);

22347 **DESCRIPTION**

22348 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 22349 conflict between the requirements described here and the ISO C standard is unintentional. This  
 22350 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

22351 These functions shall compute the quantity  $x * 2^{exp}$ .

22352 An application wishing to check for error situations should set *errno* to zero and call  
 22353 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 22354 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 22355 zero, an error has occurred.

22356 **RETURN VALUE**22357 Upon successful completion, these functions shall return *x* multiplied by 2, raised to the power  
22358 *exp*.

22359 If these functions would cause overflow, a range error shall occur and *ldexp*(*x*), *ldexpf*(*x*), and  
 22360 *ldexpl*(*x*) shall return ±HUGE\_VAL, ±HUGE\_VALF, and ±HUGE\_VALL (according to the sign of  
 22361 *x*), respectively.

22362 If the correct value would cause underflow, and is not representable, a range error may occur,  
 22363 **MX** and either 0.0 (if supported), or an implementation-defined value shall be returned.

22364 **MX** If *x* is NaN, a NaN shall be returned.22365 If *x* is ±0 or ±Inf, *x* shall be returned.22366 If *exp* is 0, *x* shall be returned.

22367 If the correct value would cause underflow, and is representable, a range error may occur and  
 22368 the correct value shall be returned.

22369 **ERRORS**

22370 These functions shall fail if:

22371 Range Error The result overflows.

22372 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero, |  
 22373 then *errno* shall be set to [ERANGE]. If the integer expression |  
 22374 (math\_errhandling & MATH\_ERREXCEPT) is non-zero, then the overflow |  
 22375 floating-point exception shall be raised. |

22376 These functions may fail if:

22377 Range Error The result underflows.

22378 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero, |  
 22379 then *errno* shall be set to [ERANGE]. If the integer expression |  
 22380 (math\_errhandling & MATH\_ERREXCEPT) is non-zero, then the underflow |  
 22381 floating-point exception shall be raised. |

22382 **EXAMPLES**

22383 None.

22384 **APPLICATION USAGE**

22385 On error, the expressions (math\_errhandling & MATH\_ERRNO) and (math\_errhandling &  
22386 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

22387 **RATIONALE**

22388 None.

22389 **FUTURE DIRECTIONS**

22390 None.

22391 **SEE ALSO**

22392 *feclearexcept()*, *fetestexcept()*, *frexp()*, *isnan()*, the Base Definitions volume of |  
22393 IEEE Std 1003.1-200x, Section 4.18, Treatment of Error Conditions for Mathematical Functions, |  
22394 <math.h>

22395 **CHANGE HISTORY**

22396 First released in Issue 1. Derived from Issue 1 of the SVID.

22397 **Issue 5**

22398 The DESCRIPTION is updated to indicate how an application should check for an error. This  
22399 text was previously published in the APPLICATION USAGE section.

22400 **Issue 6**

22401 The *ldexpf()* and *ldexpl()* functions are added for alignment with the ISO/IEC 9899:1999  
22402 standard.

22403 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
22404 revised to align with the ISO/IEC 9899:1999 standard.

22405 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
22406 marked.

22407 **NAME**

22408 ldiv, lldiv — compute quotient and remainder of a long division

22409 **SYNOPSIS**

22410 #include &lt;stdlib.h&gt;

22411 ldiv\_t ldiv(long *numer*, long *denom*);22412 lldiv\_t lldiv(long long *numer*, long long *denom*);22413 **DESCRIPTION**

22414 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 22415 conflict between the requirements described here and the ISO C standard is unintentional. This  
 22416 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

22417 These functions shall compute the quotient and remainder of the division of the numerator  
 22418 *numer* by the denominator *denom*. If the division is inexact, the resulting quotient is the **long**  
 22419 integer (for the *ldiv()* function) or **long long** integer (for the *lldiv()* function) of lesser magnitude  
 22420 that is the nearest to the algebraic quotient. If the result cannot be represented, the behavior is  
 22421 undefined; otherwise, *quot* \* *denom* + *rem* shall equal *numer*.

22422 **RETURN VALUE**

22423 The *ldiv()* function shall return a structure of type **ldiv\_t**, comprising both the quotient and the  
 22424 remainder. The structure shall include the following members, in any order:

22425 long quot; /\* Quotient \*/

22426 long rem; /\* Remainder \*/

22427 The *lldiv()* function shall return a structure of type **lldiv\_t**, comprising both the quotient and the  
 22428 remainder. The structure shall include the following members, in any order:

22429 long long quot; /\* Quotient \*/

22430 long long rem; /\* Remainder \*/

22431 **ERRORS**

22432 No errors are defined.

22433 **EXAMPLES**

22434 None.

22435 **APPLICATION USAGE**

22436 None.

22437 **RATIONALE**

22438 None.

22439 **FUTURE DIRECTIONS**

22440 None.

22441 **SEE ALSO**22442 *div()*, the Base Definitions volume of IEEE Std 1003.1-200x, <stdlib.h>22443 **CHANGE HISTORY**

22444 First released in Issue 4. Derived from the ISO C standard.

22445 **Issue 6**22446 The *lldiv()* function is added for alignment with the ISO/IEC 9899:1999 standard.

22447 **NAME**

22448       lfind — find entry in a linear search table

22449 **SYNOPSIS**

22450 XSI       #include <search.h>

```
22451       void *lfind(const void *key, const void *base, size_t *nelp,  
22452                   size_t width, int (*compar)(const void *, const void *));
```

22453

22454 **DESCRIPTION**

22455       Refer to *lsearch()*.

22456 **NAME**

22457 lgamma, lgammaf, lgammal — log gamma function

22458 **SYNOPSIS**

22459 #include &lt;math.h&gt;

22460 double lgamma(double x);

22461 float lgammaf(float x);

22462 long double lgammal(long double x);

22463 XSI extern int signgam;

22464

22465 **DESCRIPTION**

22466 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 22467 conflict between the requirements described here and the ISO C standard is unintentional. This  
 22468 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

22469 These functions shall compute  $\log_e|\Gamma(x)|$  where  $\Gamma(x)$  is defined as  $\int_0^{\infty} e^{-t}t^{x-1} dt$ . The argument  $x$   
 22470 need not be a non-positive integer ( $\Gamma(x)$  is defined over the reals, except the non-positive  
 22471 integers).  
 22472

22473 XSI The sign of  $\Gamma(x)$  is returned in the external integer *signgam*.

22474 CX These functions need not be reentrant. A function that is not required to be reentrant is not  
 22475 required to be thread-safe.

22476 An application wishing to check for error situations should set *errno* to zero and call  
 22477 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 22478 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 22479 zero, an error has occurred.

22480 **RETURN VALUE**22481 Upon successful completion, these functions shall return the logarithmic gamma of  $x$ .

22482 If  $x$  is a non-positive integer, a pole error shall occur and *lgamma()*, *lgammaf()*, and *lgammal()*  
 22483 shall return +HUGE\_VAL, +HUGE\_VALF, and +HUGE\_VALL, respectively.

22484 If the correct value would cause overflow, a range error shall occur and *lgamma()*, *lgammaf()*,  
 22485 and *lgammal()* shall return ±HUGE\_VAL, ±HUGE\_VALF, and ±HUGE\_VALL (having the same  
 22486 sign as the correct value), respectively.

22487 MX If  $x$  is NaN, a NaN shall be returned.22488 If  $x$  is 1 or 2, +0 shall be returned.22489 If  $x$  is ±Inf, +Inf shall be returned22490 **ERRORS**

22491 These functions shall fail if:

22492 Pole Error The  $x$  argument is a negative integer or zero.

22493 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero, |  
 22494 then *errno* shall be set to [ERANGE]. If the integer expression |  
 22495 (math\_errhandling & MATH\_ERREXCEPT) is non-zero, then the divide-by- |  
 22496 zero floating-point exception shall be raised. |

22497 Range Error The result overflows

22498 If the integer expression (`math_errhandling & MATH_ERRNO`) is non-zero, |  
22499 then *errno* shall be set to [ERANGE]. If the integer expression |  
22500 (`math_errhandling & MATH_ERREXCEPT`) is non-zero, then the overflow |  
22501 floating-point exception shall be raised. |

**22502 EXAMPLES**

22503 None.

**22504 APPLICATION USAGE**

22505 On error, the expressions (`math_errhandling & MATH_ERRNO`) and (`math_errhandling &`  
22506 `MATH_ERREXCEPT`) are independent of each other, but at least one of them must be non-zero.

**22507 RATIONALE**

22508 None.

**22509 FUTURE DIRECTIONS**

22510 None.

**22511 SEE ALSO**

22512 *exp()*, *feclearexcept()*, *fetestexcept()*, *isnan()*, the Base Definitions volume of IEEE Std 1003.1-200x, |  
22513 Section 4.18, Treatment of Error Conditions for Mathematical Functions, <**math.h**> |

**22514 CHANGE HISTORY**

22515 First released in Issue 3.

**22516 Issue 5**

22517 The DESCRIPTION is updated to indicate how an application should check for an error. This  
22518 text was previously published in the APPLICATION USAGE section.

22519 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

**22520 Issue 6**

22521 The *lgamma()* function is no longer marked as an extension.

22522 The *lgammaf()* and *lgammal()* functions are added for alignment with the ISO/IEC 9899:1999  
22523 standard.

22524 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
22525 revised to align with the ISO/IEC 9899:1999 standard.

22526 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
22527 marked.

22528 XSI extensions are marked.

22529 **NAME**

22530 link — link to a file

22531 **SYNOPSIS**

22532 #include &lt;unistd.h&gt;

22533 int link(const char \*path1, const char \*path2);

22534 **DESCRIPTION**22535 The *link()* function shall create a new link (directory entry) for the existing file, *path1*.

22536 The *path1* argument points to a pathname naming an existing file. The *path2* argument points to  
 22537 a pathname naming the new directory entry to be created. The *link()* function shall atomically  
 22538 create a new link for the existing file and the link count of the file shall be incremented by one.

22539 If *path1* names a directory, *link()* shall fail unless the process has appropriate privileges and the  
 22540 implementation supports using *link()* on directories.

22541 Upon successful completion, *link()* shall mark for update the *st\_ctime* field of the file. Also, the  
 22542 *st\_ctime* and *st\_mtime* fields of the directory that contains the new entry shall be marked for  
 22543 update.

22544 If *link()* fails, no link shall be created and the link count of the file shall remain unchanged.

22545 The implementation may require that the calling process has permission to access the existing  
 22546 file.

22547 **RETURN VALUE**

22548 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to  
 22549 indicate the error.

22550 **ERRORS**22551 The *link()* function shall fail if:

22552 [EACCES] A component of either path prefix denies search permission, or the requested  
 22553 link requires writing in a directory that denies write permission, or the calling  
 22554 process does not have permission to access the existing file and this is  
 22555 required by the implementation.

22556 [EEXIST] The *path2* argument resolves to an existing file or refers to a symbolic link.

22557 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path1* or  
 22558 *path2* argument.

22559 [EMLINK] The number of links to the file named by *path1* would exceed {LINK\_MAX}.

22560 [ENAMETOOLONG]

22561 The length of the *path1* or *path2* argument exceeds {PATH\_MAX} or a  
 22562 pathname component is longer than {NAME\_MAX}.

22563 [ENOENT] A component of either path prefix does not exist; the file named by *path1* does  
 22564 not exist; or *path1* or *path2* points to an empty string.

22565 [ENOSPC] The directory to contain the link cannot be extended.

22566 [ENOTDIR] A component of either path prefix is not a directory.

22567 [EPERM] The file named by *path1* is a directory and either the calling process does not  
 22568 have appropriate privileges or the implementation prohibits using *link()* on  
 22569 directories.

- 22570 [EROFS] The requested link requires writing in a directory on a read-only file system.
- 22571 [EXDEV] The link named by *path2* and the file named by *path1* are on different file  
22572 XSR systems and the implementation does not support links between file systems,  
22573 or *path1* refers to a named STREAM.
- 22574 The *link()* function may fail if:
- 22575 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
22576 resolution of the *path1* or *path2* argument.
- 22577 [ENAMETOOLONG]  
22578 As a result of encountering a symbolic link in resolution of the *path1* or *path2* |  
22579 argument, the length of the substituted pathname string exceeded |  
22580 {PATH\_MAX}.

22581 **EXAMPLES**22582 **Creating a Link to a File**

22583 The following example shows how to create a link to a file named **/home/cnd/mod1** by creating a  
22584 new directory entry named **/modules/pass1**.

```
22585 #include <unistd.h>
22586
22586 char *path1 = "/home/cnd/mod1";
22587 char *path2 = "/modules/pass1";
22588 int status;
22589 ...
22590 status = link (path1, path2);
```

22591 **Creating a Link to a File Within a Program**

22592 In the following program example, the *link()* function links the **/etc/passwd** file (defined as |  
22593 **PASSWDFILE**) to a file named **/etc/opasswd** (defined as **SAVEFILE**), which is used to save the  
22594 current password file. Then, after removing the current password file (defined as  
22595 **PASSWDFILE**), the new password file is saved as the current password file using the *link()*  
22596 function again.

```
22597 #include <unistd.h>
22598
22598 #define LOCKFILE "/etc/ptmp"
22599 #define PASSWDFILE "/etc/passwd"
22600 #define SAVEFILE "/etc/opasswd"
22601 ...
22602 /* Save current password file */
22603 link (PASSWDFILE, SAVEFILE);
22604
22604 /* Remove current password file. */
22605 unlink (PASSWDFILE);
22606
22606 /* Save new password file as current password file. */
22607 link (LOCKFILE, PASSWDFILE);
```

22608 **APPLICATION USAGE**

22609 Some implementations do allow links between file systems.

**22610 RATIONALE**

22611 Linking to a directory is restricted to the superuser in most historical implementations because  
22612 this capability may produce loops in the file hierarchy or otherwise corrupt the file system. This  
22613 volume of IEEE Std 1003.1-200x continues that philosophy by prohibiting *link()* and *unlink()*  
22614 from doing this. Other functions could do it if the implementor designed such an extension.

22615 Some historical implementations allow linking of files on different file systems. Wording was  
22616 added to explicitly allow this optional behavior.

22617 The exception for cross-file system links is intended to apply only to links that are  
22618 programmatically indistinguishable from “hard” links.

**22619 FUTURE DIRECTIONS**

22620 None.

**22621 SEE ALSO**

22622 *symlink()*, *unlink()*, the Base Definitions volume of IEEE Std 1003.1-200x, <**unistd.h**>

**22623 CHANGE HISTORY**

22624 First released in Issue 1. Derived from Issue 1 of the SVID.

**22625 Issue 6**

22626 The following new requirements on POSIX implementations derive from alignment with the  
22627 Single UNIX Specification:

- 22628 • The [ELOOP] mandatory error condition is added.
- 22629 • A second [ENAMETOOLONG] is added as an optional error condition.

22630 The following changes were made to align with the IEEE P1003.1a draft standard:

- 22631 • An explanation is added of action when *path2* refers to a symbolic link.
- 22632 • The [ELOOP] optional error condition is added.

## 22633 NAME

22634 lio\_listio — list directed I/O (**REALTIME**)

## 22635 SYNOPSIS

22636 AIO #include &lt;aio.h&gt;

```
22637 int lio_listio(int mode, struct aiocb *restrict const list[restrict],
22638               int nent, struct sigevent *restrict sig);
22639
```

## 22640 DESCRIPTION

22641 The *lio\_listio()* function shall initiate a list of I/O requests with a single function call.

22642 The *mode* argument takes one of the values LIO\_WAIT or LIO\_NOWAIT declared in <aio.h> and  
 22643 determines whether the function returns when the I/O operations have been completed, or as  
 22644 soon as the operations have been queued. If the *mode* argument is LIO\_WAIT, the function shall  
 22645 wait until all I/O is complete and the *sig* argument shall be ignored.

22646 If the *mode* argument is LIO\_NOWAIT, the function shall return immediately, and asynchronous  
 22647 notification shall occur, according to the *sig* argument, when all the I/O operations complete. If  
 22648 *sig* is NULL, then no asynchronous notification shall occur. If *sig* is not NULL, asynchronous  
 22649 notification occurs as specified in Section 2.4.1 (on page 478) when all the requests in *list* have  
 22650 completed.

22651 The I/O requests enumerated by *list* are submitted in an unspecified order.

22652 The *list* argument is an array of pointers to **aiocb** structures. The array contains *nent* elements.  
 22653 The array may contain NULL elements, which shall be ignored.

22654 The *aio\_lio\_opcode* field of each **aiocb** structure specifies the operation to be performed. The  
 22655 supported operations are LIO\_READ, LIO\_WRITE, and LIO\_NOP; these symbols are defined in  
 22656 <aio.h>. The LIO\_NOP operation causes the list entry to be ignored. If the *aio\_lio\_opcode*  
 22657 element is equal to LIO\_READ, then an I/O operation is submitted as if by a call to *aio\_read()*  
 22658 with the *aiocbp* equal to the address of the **aiocb** structure. If the *aio\_lio\_opcode* element is equal  
 22659 to LIO\_WRITE, then an I/O operation is submitted as if by a call to *aio\_write()* with the *aiocbp*  
 22660 equal to the address of the **aiocb** structure.

22661 The *aio\_fildes* member specifies the file descriptor on which the operation is to be performed.22662 The *aio\_buf* member specifies the address of the buffer to or from which the data is transferred.22663 The *aio\_nbytes* member specifies the number of bytes of data to be transferred.

22664 The members of the **aiocb** structure further describe the I/O operation to be performed, in a  
 22665 manner identical to that of the corresponding **aiocb** structure when used by the *aio\_read()* and  
 22666 *aio\_write()* functions.

22667 The *nent* argument specifies how many elements are members of the list; that is, the length of the  
 22668 array.

22669 The behavior of this function is altered according to the definitions of synchronized I/O data  
 22670 integrity completion and synchronized I/O file integrity completion if synchronized I/O is  
 22671 enabled on the file associated with *aio\_fildes*.

22672 For regular files, no data transfer shall occur past the offset maximum established in the open  
 22673 file description associated with *aiocbp->aio\_fildes*.

## 22674 RETURN VALUE

22675 If the *mode* argument has the value LIO\_NOWAIT, the *lio\_listio()* function shall return the value  
 22676 zero if the I/O operations are successfully queued; otherwise, the function shall return the value  
 22677 -1 and set *errno* to indicate the error.

22678 If the *mode* argument has the value LIO\_WAIT, the *lio\_listio()* function shall return the value  
 22679 zero when all the indicated I/O has completed successfully. Otherwise, *lio\_listio()* shall return a  
 22680 value of -1 and set *errno* to indicate the error.

22681 In either case, the return value only indicates the success or failure of the *lio\_listio()* call itself,  
 22682 not the status of the individual I/O requests. In some cases one or more of the I/O requests  
 22683 contained in the list may fail. Failure of an individual request does not prevent completion of  
 22684 any other individual request. To determine the outcome of each I/O request, the application  
 22685 shall examine the error status associated with each **aiocb** control block. The error statuses so  
 22686 returned are identical to those returned as the result of an *aio\_read()* or *aio\_write()* function.

## 22687 ERRORS

22688 The *lio\_listio()* function shall fail if:

22689 [EAGAIN] The resources necessary to queue all the I/O requests were not available. The  
 22690 application may check the error status for each **aiocb** to determine the  
 22691 individual request(s) that failed.

22692 [EAGAIN] The number of entries indicated by *nent* would cause the system-wide limit  
 22693 {AIO\_MAX} to be exceeded.

22694 [EINVAL] The *mode* argument is not a proper value, or the value of *nent* was greater than  
 22695 {AIO\_LISTIO\_MAX}.

22696 [EINTR] A signal was delivered while waiting for all I/O requests to complete during  
 22697 an LIO\_WAIT operation. Note that, since each I/O operation invoked by  
 22698 *lio\_listio()* may possibly provoke a signal when it completes, this error return  
 22699 may be caused by the completion of one (or more) of the very I/O operations  
 22700 being awaited. Outstanding I/O requests are not canceled, and the application  
 22701 shall examine each list element to determine whether the request was  
 22702 initiated, canceled, or completed.

22703 [EIO] One or more of the individual I/O operations failed. The application may  
 22704 check the error status for each **aiocb** structure to determine the individual  
 22705 request(s) that failed.

22706 In addition to the errors returned by the *lio\_listio()* function, if the *lio\_listio()* function succeeds  
 22707 or fails with errors of [EAGAIN], [EINTR], or [EIO], then some of the I/O specified by the list  
 22708 may have been initiated. If the *lio\_listio()* function fails with an error code other than [EAGAIN],  
 22709 [EINTR], or [EIO], no operations from the list shall have been initiated. The I/O operation  
 22710 indicated by each list element can encounter errors specific to the individual read or write  
 22711 function being performed. In this event, the error status for each **aiocb** control block contains the  
 22712 associated error code. The error codes that can be set are the same as would be set by a *read()* or  
 22713 *write()* function, with the following additional error codes possible:

22714 [EAGAIN] The requested I/O operation was not queued due to resource limitations.

22715 [ECANCELED] The requested I/O was canceled before the I/O completed due to an explicit  
 22716 *aio\_cancel()* request.

22717 [EFBIG] The *aiocbp->aio\_lio\_opcode* is LIO\_WRITE, the file is a regular file, *aiocbp->*  
 22718 *aio\_nbytes* is greater than 0, and the *aiocbp->aio\_offset* is greater than or equal  
 22719 to the offset maximum in the open file description associated with *aiocbp-*

22720 >*aio\_fildes*.

22721 [EINPROGRESS] The requested I/O is in progress.

22722 [EOVERFLOW] The *aiochp->aio\_lio\_opcode* is LIO\_READ, the file is a regular file, *aiochp->aio\_nbytes* is greater than 0, and the *aiochp->aio\_offset* is before the end-of-file and is greater than or equal to the offset maximum in the open file description associated with *aiochp->aio\_fildes*.

#### 22726 EXAMPLES

22727 None.

#### 22728 APPLICATION USAGE

22729 None.

#### 22730 RATIONALE

22731 Although it may appear that there are inconsistencies in the specified circumstances for error codes, the [EIO] error condition applies when any circumstance relating to an individual operation makes that operation fail. This might be due to a badly formulated request (for example, the *aio\_lio\_opcode* field is invalid, and *aio\_error()* returns [EINVAL]) or might arise from application behavior (for example, the file descriptor is closed before the operation is initiated, and *aio\_error()* returns [EBADF]).

22737 The limitation on the set of error codes returned when operations from the list shall have been initiated enables applications to know when operations have been started and whether *aio\_error()* is valid for a specific operation.

#### 22740 FUTURE DIRECTIONS

22741 None.

#### 22742 SEE ALSO

22743 *aio\_read()*, *aio\_write()*, *aio\_error()*, *aio\_return()*, *aio\_cancel()*, *close()*, *exec*, *exit()*, *fork()*, *lseek()*, *read()*, the Base Definitions volume of IEEE Std 1003.1-200x, <**aio.h**>

#### 22745 CHANGE HISTORY

22746 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

22747 Large File Summit extensions are added.

#### 22748 Issue 6

22749 The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Asynchronous Input and Output option.

22751 The *lio\_listio()* function is marked as part of the Asynchronous Input and Output option.

22752 The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- 22754 • In the DESCRIPTION, text is added to indicate that for regular files no data transfer occurs past the offset maximum established in the open file description associated with *aiochp->aio\_fildes*. This change is to support large files.
- 22755 • The [EBIG] and [EOVERFLOW] error conditions are defined. This change is to support large files.

22759 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

22760 The **restrict** keyword is added to the *lio\_listio()* prototype for alignment with the ISO/IEC 9899:1999 standard.

22761

22762 **NAME**

22763 listen — listen for socket connections and limit the queue of incoming connections

22764 **SYNOPSIS**

22765 #include <sys/socket.h>

22766 int listen(int *socket*, int *backlog*);

22767 **DESCRIPTION**

22768 The *listen()* function shall mark a connection-mode socket, specified by the *socket* argument, as  
 22769 accepting connections.

22770 The *backlog* argument provides a hint to the implementation which the implementation shall use  
 22771 to limit the number of outstanding connections in the socket's listen queue. Implementations  
 22772 may impose a limit on *backlog* and silently reduce the specified value. Normally, a larger *backlog*  
 22773 argument value shall result in a larger or equal length of the listen queue. Implementations shall  
 22774 support values of *backlog* up to SOMAXCONN, defined in <sys/socket.h>.

22775 The implementation may include incomplete connections in its listen queue. The limits on the  
 22776 number of incomplete connections and completed connections queued may be different.

22777 The implementation may have an upper limit on the length of the listen queue—either global or  
 22778 per accepting socket. If *backlog* exceeds this limit, the length of the listen queue is set to the limit.

22779 If *listen()* is called with a *backlog* argument value that is less than 0, the function behaves as if it  
 22780 had been called with a *backlog* argument value of 0.

22781 A *backlog* argument of 0 may allow the socket to accept connections, in which case the length of  
 22782 the listen queue may be set to an implementation-defined minimum value.

22783 The socket in use may require the process to have appropriate privileges to use the *listen()*  
 22784 function.

22785 **RETURN VALUE**

22786 Upon successful completions, *listen()* shall return 0; otherwise, -1 shall be returned and *errno* set  
 22787 to indicate the error.

22788 **ERRORS**

22789 The *listen()* function shall fail if:

22790 [EBADF] The *socket* argument is not a valid file descriptor.

22791 [EDESTADDRREQ]

22792 The socket is not bound to a local address, and the protocol does not support  
 22793 listening on an unbound socket.

22794 [EINVAL] The *socket* is already connected.

22795 [ENOTSOCK] The *socket* argument does not refer to a socket.

22796 [EOPNOTSUPP] The socket protocol does not support *listen()*.

22797 The *listen()* function may fail if:

22798 [EACCES] The calling process does not have the appropriate privileges.

22799 [EINVAL] The *socket* has been shut down.

22800 [ENOBUFS] Insufficient resources are available in the system to complete the call.

22801 **EXAMPLES**

22802           None.

22803 **APPLICATION USAGE**

22804           None.

22805 **RATIONALE**

22806           None.

22807 **FUTURE DIRECTIONS**

22808           None.

22809 **SEE ALSO**

22810           *accept()*, *connect()*, *socket()*, the Base Definitions volume of IEEE Std 1003.1-200x, <**sys/socket.h**>

22811 **CHANGE HISTORY**

22812           First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

22813           The DESCRIPTION is updated to describe the relationship of SOMAXCONN and the *backlog*

22814           argument.

22815 **NAME**

22816       llabs — return a long integer absolute value

22817 **SYNOPSIS**

22818       #include <stdlib.h>

22819       long long llabs(long long *i*);

22820 **DESCRIPTION**

22821       Refer to *labs()*.

22822 **NAME**

22823        `lldiv` — compute quotient and remainder of a long division

22824 **SYNOPSIS**

22825        `#include <stdlib.h>`

22826        `lldiv_t lldiv(long long numer, long long denom);`

22827 **DESCRIPTION**

22828        Refer to *ldiv()*.

22829 **NAME**

22830 llrint, llrintf, llrintl, — round to nearest integer value using current rounding direction

22831 **SYNOPSIS**

22832 #include <math.h>

22833 long long llrint(double x);

22834 long long llrintf(float x);

22835 long long llrintl(long double x);

22836 **DESCRIPTION**

22837 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 22838 conflict between the requirements described here and the ISO C standard is unintentional. This  
 22839 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

22840 These functions shall round their argument to the nearest integer value, rounding according to  
 22841 the current rounding direction.

22842 An application wishing to check for error situations should set *errno* to zero and call  
 22843 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 22844 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 22845 zero, an error has occurred.

22846 **RETURN VALUE**

22847 Upon successful completion, these functions shall return the rounded integer value.

22848 **MX** If *x* is NaN, a domain error shall occur, and an unspecified value is returned.

22849 If *x* is +Inf, a domain error shall occur and an unspecified value is returned.

22850 If *x* is -Inf, a domain error shall occur and an unspecified value is returned.

22851 If the correct value is positive and too large to represent as a **long long**, a domain error shall  
 22852 occur and an unspecified value is returned.

22853 If the correct value is negative and too large to represent as a **long long**, a domain error shall  
 22854 occur and an unspecified value is returned.

22855 **ERRORS**

22856 These functions shall fail if:

22857 **MX** Domain Error The *x* argument is NaN or ±Inf, or the correct value is not representable as an  
 22858 integer.

22859 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero,  
 22860 then *errno* shall be set to [EDOM]. If the integer expression (math\_errhandling  
 22861 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception  
 22862 shall be raised.

22863 **EXAMPLES**

22864 None.

22865 **APPLICATION USAGE**

22866 On error, the expressions (math\_errhandling & MATH\_ERRNO) and (math\_errhandling &  
 22867 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

22868 **RATIONALE**

22869 These functions provide floating-to-integer conversions. They round according to the current  
 22870 rounding direction. If the rounded value is outside the range of the return type, the numeric  
 22871 result is unspecified and the invalid floating-point exception is raised. When they raise no other  
 22872 floating-point exception and the result differs from the argument, they raise the inexact

22873 floating-point exception.

22874 **FUTURE DIRECTIONS**

22875 None.

22876 **SEE ALSO**

22877 *feclearexcept()*, *fetestexcept()*, *lrint()*, the Base Definitions volume of IEEE Std 1003.1-200x, Section |  
22878 4.18, Treatment of Error Conditions for Mathematical Functions, <math.h> |

22879 **CHANGE HISTORY**

22880 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

## 22881 NAME

22882 llround, llroundf, llroundl, — round to nearest integer value

## 22883 SYNOPSIS

22884 #include &lt;math.h&gt;

22885 long long llround(double x);

22886 long long llroundf(float x);

22887 long long llroundl(long double x);

## 22888 DESCRIPTION

22889 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 22890 conflict between the requirements described here and the ISO C standard is unintentional. This  
 22891 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

22892 These functions shall round their argument to the nearest integer value, rounding halfway cases  
 22893 away from zero, regardless of the current rounding direction.

22894 An application wishing to check for error situations should set *errno* to zero and call  
 22895 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 22896 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 22897 zero, an error has occurred.

## 22898 RETURN VALUE

22899 Upon successful completion, these functions shall return the rounded integer value.

22900 MX If *x* is NaN, a domain error shall occur, and an unspecified value is returned.22901 If *x* is +Inf, a domain error shall occur and an unspecified value is returned.22902 If *x* is -Inf, a domain error shall occur and an unspecified value is returned.

22903 If the correct value is positive and too large to represent as a **long long**, a domain error shall  
 22904 occur and an unspecified value is returned.

22905 If the correct value is negative and too large to represent as a **long long**, a domain error shall  
 22906 occur and an unspecified value is returned.

## 22907 ERRORS

22908 These functions shall fail if:

22909 MX	Domain Error	The <i>x</i> argument is NaN or ±Inf, or the correct value is not representable as an integer.
----------	--------------	--

22911	22912	22913	22914	If the integer expression (math_errhandling & MATH_ERRNO) is non-zero, then <i>errno</i> shall be set to [EDOM]. If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.
-------	-------	-------	-------	--

## 22915 EXAMPLES

22916 None.

## 22917 APPLICATION USAGE

22918 On error, the expressions (math\_errhandling & MATH\_ERRNO) and (math\_errhandling &  
 22919 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

## 22920 RATIONALE

22921 These functions provide floating-to-integer conversions. They round according to the current  
 22922 rounding direction. If the rounded value is outside the range of the return type, the numeric  
 22923 result is unspecified and the invalid floating-point exception is raised. When they raise no other  
 22924 floating-point exception and the result differs from the argument, they raise the inexact

22925 floating-point exception.

22926 These functions differ from the *llrint()* functions in that the default rounding direction for the  
22927 *lround()* functions round halfway cases away from zero and need not raise the inexact floating-  
22928 point exception for non-integer arguments that round to within the range of the return type.

22929 **FUTURE DIRECTIONS**

22930 None.

22931 **SEE ALSO**

22932 *feclearexcept()*, *fetestexcept()*, *lround()*, the Base Definitions volume of IEEE Std 1003.1-200x, |  
22933 Section 4.18, Treatment of Error Conditions for Mathematical Functions, <math.h> |

22934 **CHANGE HISTORY**

22935 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

22936 **NAME**

22937 localeconv — return locale-specific information

22938 **SYNOPSIS**

22939 #include &lt;locale.h&gt;

22940 struct lconv \*localeconv(void);

22941 **DESCRIPTION**

22942 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 22943 conflict between the requirements described here and the ISO C standard is unintentional. This  
 22944 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

22945 The *localeconv()* function shall set the components of an object with the type **struct lconv** with  
 22946 the values appropriate for the formatting of numeric quantities (monetary and otherwise)  
 22947 according to the rules of the current locale.

22948 The members of the structure with type **char \*** are pointers to strings, any of which (except  
 22949 **decimal\_point**) can point to " ", to indicate that the value is not available in the current locale or  
 22950 is of zero length. The members with type **char** are non-negative numbers, any of which can be  
 22951 {CHAR\_MAX} to indicate that the value is not available in the current locale.

22952 The members include the following:

22953 **char \*decimal\_point**

22954 The radix character used to format non-monetary quantities.

22955 **char \*thousands\_sep**

22956 The character used to separate groups of digits before the decimal-point character in  
 22957 formatted non-monetary quantities.

22958 **char \*grouping**

22959 A string whose elements taken as one-byte integer values indicate the size of each group of  
 22960 digits in formatted non-monetary quantities.

22961 **char \*int\_curr\_symbol**

22962 The international currency symbol applicable to the current locale. The first three  
 22963 characters contain the alphabetic international currency symbol in accordance with those  
 22964 specified in the ISO 4217:1995 standard. The fourth character (immediately preceding the  
 22965 null byte) is the character used to separate the international currency symbol from the  
 22966 monetary quantity.

22967 **char \*currency\_symbol**

22968 The local currency symbol applicable to the current locale.

22969 **char \*mon\_decimal\_point**

22970 The radix character used to format monetary quantities.

22971 **char \*mon\_thousands\_sep**

22972 The separator for groups of digits before the decimal-point in formatted monetary  
 22973 quantities.

22974 **char \*mon\_grouping**

22975 A string whose elements taken as one-byte integer values indicate the size of each group of  
 22976 digits in formatted monetary quantities.

22977 **char \*positive\_sign**

22978 The string used to indicate a non-negative valued formatted monetary quantity.

- 22979 **char \*negative\_sign**  
 22980 The string used to indicate a negative valued formatted monetary quantity.
- 22981 **char int\_frac\_digits**  
 22982 The number of fractional digits (those after the decimal-point) to be displayed in an  
 22983 internationally formatted monetary quantity.
- 22984 **char frac\_digits**  
 22985 The number of fractional digits (those after the decimal-point) to be displayed in a  
 22986 formatted monetary quantity.
- 22987 **char p\_cs\_precedes**  
 22988 Set to 1 if the **currency\_symbol** or **int\_curr\_symbol** precedes the value for a non-negative  
 22989 formatted monetary quantity. Set to 0 if the symbol succeeds the value.
- 22990 **char p\_sep\_by\_space**  
 22991 Set to 0 if no space separates the **currency\_symbol** or **int\_curr\_symbol** from the value for a  
 22992 non-negative formatted monetary quantity. Set to 1 if a space separates the symbol from the  
 22993 value; and set to 2 if a space separates the symbol and the sign string, if adjacent. XSI
- 22994 **char n\_cs\_precedes**  
 22995 Set to 1 if the **currency\_symbol** or **int\_curr\_symbol** precedes the value for a negative  
 22996 formatted monetary quantity. Set to 0 if the symbol succeeds the value.
- 22997 **char n\_sep\_by\_space**  
 22998 Set to 0 if no space separates the **currency\_symbol** or **int\_curr\_symbol** from the value for a  
 22999 negative formatted monetary quantity. Set to 1 if a space separates the symbol from the  
 23000 value; and set to 2 if a space separates the symbol and the sign string, if adjacent. XSI
- 23001 **char p\_sign\_posn**  
 23002 Set to a value indicating the positioning of the **positive\_sign** for a non-negative formatted  
 23003 monetary quantity.
- 23004 **char n\_sign\_posn**  
 23005 Set to a value indicating the positioning of the **negative\_sign** for a negative formatted  
 23006 monetary quantity.
- 23007 **char int\_p\_cs\_precedes**  
 23008 Set to 1 or 0 if the **int\_curr\_symbol** respectively precedes or succeeds the value for a non- |  
 23009 negative internationally formatted monetary quantity.
- 23010 **char int\_n\_cs\_precedes**  
 23011 Set to 1 or 0 if the **int\_curr\_symbol** respectively precedes or succeeds the value for a |  
 23012 negative internationally formatted monetary quantity.
- 23013 **char int\_p\_sep\_by\_space**  
 23014 Set to a value indicating the separation of the **int\_curr\_symbol**, the sign string, and the |  
 23015 value for a non-negative internationally formatted monetary quantity.
- 23016 **char int\_n\_sep\_by\_space**  
 23017 Set to a value indicating the separation of the **int\_curr\_symbol**, the sign string, and the |  
 23018 value for a negative internationally formatted monetary quantity.
- 23019 **char int\_p\_sign\_posn**  
 23020 Set to a value indicating the positioning of the **positive\_sign** for a non-negative  
 23021 internationally formatted monetary quantity.
- 23022 **char int\_n\_sign\_posn**  
 23023 Set to a value indicating the positioning of the **negative\_sign** for a negative internationally

- 23024 formatted monetary quantity.
- 23025 The elements of **grouping** and **mon\_grouping** are interpreted according to the following:
- 23026 {CHAR\_MAX} No further grouping is to be performed.
- 23027 0 The previous element is to be repeatedly used for the remainder of the digits.
- 23028 *other* The integer value is the number of digits that comprise the current group. The  
23029 next element is examined to determine the size of the next group of digits  
23030 before the current group.
- 23031 The values of **p\_sep\_by\_space**, **n\_sep\_by\_space**, **int\_p\_sep\_by\_space**, and **int\_n\_sep\_by\_space**  
23032 are interpreted according to the following:
- 23033 0 No space separates the currency symbol and value.
- 23034 1 If the currency symbol and sign string are adjacent, a space separates them from the value;  
23035 otherwise, a space separates the currency symbol from the value.
- 23036 2 If the currency symbol and sign string are adjacent, a space separates them; otherwise, a  
23037 space separates the sign string from the value.
- 23038 For **int\_p\_sep\_by\_space** and **int\_n\_sep\_by\_space**, the fourth character of **int\_curr\_symbol** is |  
23039 used instead of a space. |
- 23040 The values of **p\_sign\_posn**, **n\_sign\_posn**, **int\_p\_sign\_posn**, and **int\_n\_sign\_posn** are |  
23041 interpreted according to the following: |
- 23042 0 Parentheses surround the quantity and **currency\_symbol** or **int\_curr\_symbol**.
- 23043 1 The sign string precedes the quantity and **currency\_symbol** or **int\_curr\_symbol**.
- 23044 2 The sign string succeeds the quantity and **currency\_symbol** or **int\_curr\_symbol**.
- 23045 3 The sign string immediately precedes the **currency\_symbol** or **int\_curr\_symbol**.
- 23046 4 The sign string immediately succeeds the **currency\_symbol** or **int\_curr\_symbol**.
- 23047 The implementation shall behave as if no function in this volume of IEEE Std 1003.1-200x calls  
23048 *localeconv()*.
- 23049 cx The *localeconv()* function need not be reentrant. A function that is not required to be reentrant is  
23050 not required to be thread-safe.
- 23051 **RETURN VALUE**
- 23052 The *localeconv()* function shall return a pointer to the filled-in object. The application shall not  
23053 modify the structure pointed to by the return value which may be overwritten by a subsequent  
23054 call to *localeconv()*. In addition, calls to *setlocale()* with the categories *LC\_ALL*, *LC\_MONETARY*,  
23055 or *LC\_NUMERIC* may overwrite the contents of the structure.
- 23056 **ERRORS**
- 23057 No errors are defined.

23058 **EXAMPLES**

23059 None.

23060 **APPLICATION USAGE**

23061 The following table illustrates the rules which may be used by four countries to format monetary  
 23062 quantities.

Country	Positive Format	Negative Format	International Format
Italy	L.1.230	-L.1.230	ITL.1.230
Netherlands	F 1.234,56	F -1.234,56	NLG 1.234,56
Norway	kr1.234,56	kr1.234,56-	NOK 1.234,56
Switzerland	SFrS.1,234.56	SFrS.1,234.56C	CHF 1,234.56

23068 For these four countries, the respective values for the monetary members of the structure  
 23069 returned by *localeconv()* are:

	Italy	Netherlands	Norway	Switzerland
23070 <b>int_curr_symbol</b>	"ITL."	"NLG "	"NOK "	"CHF "
23071 <b>currency_symbol</b>	"L."	"F"	"kr"	"SFrS."
23072 <b>mon_decimal_point</b>	" "	","	","	."
23073 <b>mon_thousands_sep</b>	."	."	."	,"
23074 <b>mon_grouping</b>	"\3"	"\3"	"\3"	"\3"
23075 <b>positive_sign</b>	" "	" "	" "	" "
23076 <b>negative_sign</b>	"-"	"-"	"-"	"C"
23077 <b>int_frac_digits</b>	0	2	2	2
23078 <b>frac_digits</b>	0	2	2	2
23079 <b>p_cs_precedes</b>	1	1	1	1
23080 <b>p_sep_by_space</b>	0	1	0	0
23081 <b>n_cs_precedes</b>	1	1	1	1
23082 <b>n_sep_by_space</b>	0	1	0	0
23083 <b>p_sign_posn</b>	1	1	1	1
23084 <b>n_sign_posn</b>	1	4	2	2
23085 <b>int_p_cs_precedes</b>	1	1	1	1
23086 <b>int_n_cs_precedes</b>	1	1	1	1
23087 <b>int_p_sep_by_space</b>	0	0	0	0
23088 <b>int_n_sep_by_space</b>	0	0	0	0
23089 <b>int_p_sign_posn</b>	1	1	1	1
23090 <b>int_n_sign_posn</b>	1	4	4	2

23092 **RATIONALE**

23093 None.

23094 **FUTURE DIRECTIONS**

23095 None.

23096 **SEE ALSO**

23097 *isalpha()*, *isascii()*, *nl\_langinfo()*, *printf()*, *scanf()*, *setlocale()*, *strcat()*, *strchr()*, *strcmp()*, *strcoll()*,  
 23098 *strcpy()*, *strftime()*, *strlen()*, *strpbrk()*, *strspn()*, *strtok()*, *strxfrm()*, *strtod()*, the Base Definitions  
 23099 volume of IEEE Std 1003.1-200x, <langinfo.h>, <locale.h>

23100 **CHANGE HISTORY**

23101 First released in Issue 4. Derived from the ANSI C standard.

23102 **Issue 6**

23103 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

23104 The RETURN VALUE section is rewritten to avoid use of the term “must”.

23105 This reference page is updated for alignment with the ISO/IEC 9899: 1999 standard. |

23106 ISO/IEC 9899: 1999 standard, Technical Corrigendum No. 1 is incorporated. |

23107 **NAME**

23108 localtime, localtime\_r — convert a time value to a broken-down local time

23109 **SYNOPSIS**

23110 #include &lt;time.h&gt;

23111 struct tm \*localtime(const time\_t \*timer);

23112 TSF struct tm \*localtime\_r(const time\_t \*restrict timer,

23113 struct tm \*restrict result);

23114

23115 **DESCRIPTION**

23116 CX For *localtime()*: The functionality described on this reference page is aligned with the ISO C  
 23117 standard. Any conflict between the requirements described here and the ISO C standard is  
 23118 unintentional. This volume of IEEE Std 1003.1-200x defers to the ISO C standard.

23119 The *localtime()* function shall convert the time in seconds since the Epoch pointed to by *timer*  
 23120 into a broken-down time, expressed as a local time. The function corrects for the timezone and  
 23121 CX any seasonal time adjustments. Local timezone information is used as though *localtime()* calls  
 23122 *tzset()*.

23123 The relationship between a time in seconds since the Epoch used as an argument to *localtime()* |  
 23124 and the **tm** structure (defined in the <time.h> header) is that the result shall be as specified in the |  
 23125 expression given in the definition of seconds since the Epoch (see the Base Definitions volume of |  
 23126 IEEE Std 1003.1-200x, Section 4.14, Seconds Since the Epoch) corrected for timezone and any |  
 23127 seasonal time adjustments, where the names in the structure and in the expression correspond. |

23128 TSF The same relationship shall apply for *localtime\_r()*. |

23129 CX The *localtime()* function need not be reentrant. A function that is not required to be reentrant is |  
 23130 not required to be thread-safe. |

23131 The *asctime()*, *ctime()*, *gmtime()*, and *localtime()* functions shall return values in one of two static |  
 23132 objects: a broken-down time structure and an array of type **char**. Execution of any of the |  
 23133 functions may overwrite the information returned in either of these objects by any of the other |  
 23134 functions. |

23135 TSF The *localtime\_r()* function shall convert the time in seconds since the Epoch pointed to by *timer*  
 23136 into a broken-down time stored in the structure to which *result* points. The *localtime\_r()* function  
 23137 shall also return a pointer to that same structure.

23138 Unlike *localtime()*, the reentrant version is not required to set *tzname*.

23139 **RETURN VALUE**

23140 The *localtime()* function shall return a pointer to the broken-down time structure.

23141 TSF Upon successful completion, *localtime\_r()* shall return a pointer to the structure pointed to by  
 23142 the argument *result*.

23143 **ERRORS**

23144 No errors are defined.

23145 **EXAMPLES**23146 **Getting the Local Date and Time**

23147 The following example uses the *time()* function to calculate the time elapsed, in seconds, since  
 23148 January 1, 1970 0:00 UTC (the Epoch), *localtime()* to convert that value to a broken-down time,  
 23149 and *asctime()* to convert the broken-down time values into a printable string.

```
23150 #include <stdio.h>
23151 #include <time.h>
23152 main()
23153 {
23154     time_t result;
23155     result = time(NULL);
23156     printf("%s%ld secs since the Epoch\n",
23157           asctime(localtime(&result)),
23158           (long)result);
23159     return(0);
23160 }
```

23161 This example writes the current time to *stdout* in a form like this:

```
23162 Wed Jun 26 10:32:15 1996
23163 835810335 secs since the Epoch
```

23164 **Getting the Modification Time for a File**

23165 The following example gets the modification time for a file. The *localtime()* function converts the  
 23166 **time\_t** value of the last modification date, obtained by a previous call to *stat()*, into a **tm**  
 23167 structure that contains the year, month, day, and so on.

```
23168 #include <time.h>
23169 ...
23170 struct stat statbuf;
23171 ...
23172 tm = localtime(&statbuf.st_mtime);
23173 ...
```

23174 **Timing an Event**

23175 The following example gets the current time, converts it to a string using *localtime()* and  
 23176 *asctime()*, and prints it to standard output using *fputs()*. It then prints the number of minutes to  
 23177 an event being timed.

```
23178 #include <time.h>
23179 #include <stdio.h>
23180 ...
23181 time_t now;
23182 int minutes_to_event;
23183 ...
23184 time(&now);
23185 printf("The time is ");
23186 fputs(asctime(localtime(&now)), stdout);
23187 printf("There are still %d minutes to the event.\n",
```

23188                   minutes\_to\_event);  
23189                   ...

23190 **APPLICATION USAGE**

23191                   The *localtime\_r()* function is thread-safe and returns values in a user-supplied buffer instead of  
23192                   possibly using a static data area that may be overwritten by each call.

23193 **RATIONALE**

23194                   None.

23195 **FUTURE DIRECTIONS**

23196                   None.

23197 **SEE ALSO**

23198                   *asctime()*, *clock()*, *ctime()*, *difftime()*, *getdate()*, *gmtime()*, *mktime()*, *strftime()*, *strptime()*, *time()*,  
23199                   *utime()*, the Base Definitions volume of IEEE Std 1003.1-200x, <**time.h**>

23200 **CHANGE HISTORY**

23201                   First released in Issue 1. Derived from Issue 1 of the SVID.

23202 **Issue 5**

23203                   A note indicating that the *localtime()* function need not be reentrant is added to the  
23204                   DESCRIPTION.

23205                   The *localtime\_r()* function is included for alignment with the POSIX Threads Extension.

23206 **Issue 6**

23207                   The *localtime\_r()* function is marked as part of the Thread-Safe Functions option.

23208                   Extensions beyond the ISO C standard are now marked.

23209                   The APPLICATION USAGE section is updated to include a note on the thread-safe function and  
23210                   its avoidance of possibly using a static data area.

23211                   The **restrict** keyword is added to the *localtime\_r()* prototype for alignment with the  
23212                   ISO/IEC 9899:1999 standard.

23213                   Examples are added.

23214 **NAME**

23215 lockf — record locking on files

23216 **SYNOPSIS**23217 XSI `#include <unistd.h>`23218 `int lockf(int fildes, int function, off_t size);`

23219

23220 **DESCRIPTION**

23221 The *lockf()* function shall lock sections of a file with advisory-mode locks. Calls to *lockf()* from  
 23222 other threads which attempt to lock the locked file section shall either return an error value or  
 23223 block until the section becomes unlocked. All the locks for a process are removed when the  
 23224 process terminates. Record locking with *lockf()* shall be supported for regular files and may be  
 23225 supported for other files.

23226 The *fildes* argument is an open file descriptor. To establish a lock with this function, the file  
 23227 descriptor shall be opened with write-only permission (O\_WRONLY) or with read/write  
 23228 permission (O\_RDWR).

23229 The *function* argument is a control value which specifies the action to be taken. The permissible  
 23230 values for *function* are defined in <unistd.h> as follows:

23231

23232

23233

23234

23235

23236

Function	Description
F_ULOCK	Unlock locked sections.
F_LOCK	Lock a section for exclusive use.
F_TLOCK	Test and lock a section for exclusive use.
F_TEST	Test a section for locks by other processes.

23237 F\_TEST shall detect if a lock by another process is present on the specified section.

23238 F\_LOCK and F\_TLOCK shall both lock a section of a file if the section is available.

23239 F\_ULOCK shall remove locks from a section of the file.

23240 The *size* argument is the number of contiguous bytes to be locked or unlocked. The section to be  
 23241 locked or unlocked starts at the current offset in the file and extends forward for a positive *size*  
 23242 or backward for a negative *size* (the preceding bytes up to but not including the current offset).  
 23243 If *size* is 0, the section from the current offset through the largest possible file offset shall be  
 23244 locked (that is, from the current offset through the present or any future end-of-file). An area  
 23245 need not be allocated to the file to be locked because locks may exist past the end-of-file.

23246 The sections locked with F\_LOCK or F\_TLOCK may, in whole or in part, contain or be contained  
 23247 by a previously locked section for the same process. When this occurs, or if adjacent locked  
 23248 sections would occur, the sections shall be combined into a single locked section. If the request  
 23249 would cause the number of locks to exceed a system-imposed limit, the request shall fail.

23250 F\_LOCK and F\_TLOCK requests differ only by the action taken if the section is not available.

23251 F\_LOCK shall block the calling thread until the section is available. F\_TLOCK shall cause the  
 23252 function to fail if the section is already locked by another process.

23253 File locks shall be released on first close by the locking process of any file descriptor for the file.

23254 F\_ULOCK requests may release (wholly or in part) one or more locked sections controlled by the  
 23255 process. Locked sections shall be unlocked starting at the current file offset through *size* bytes or  
 23256 to the end-of-file if *size* is (off\_t)0. When all of a locked section is not released (that is, when the  
 23257 beginning or end of the area to be unlocked falls within a locked section), the remaining portions  
 23258 of that section shall remain locked by the process. Releasing the center portion of a locked

- 23259 section shall cause the remaining locked beginning and end portions to become two separate  
 23260 locked sections. If the request would cause the number of locks in the system to exceed a  
 23261 system-imposed limit, the request shall fail.
- 23262 A potential for deadlock occurs if the threads of a process controlling a locked section are  
 23263 blocked by accessing another process' locked section. If the system detects that deadlock would  
 23264 occur, *lockf()* shall fail with an [EDEADLK] error.
- 23265 The interaction between *fcntl()* and *lockf()* locks is unspecified.
- 23266 Blocking on a section shall be interrupted by any signal.
- 23267 An F\_ULOCK request in which *size* is non-zero and the offset of the last byte of the requested  
 23268 section is the maximum value for an object of type *off\_t*, when the process has an existing lock  
 23269 in which *size* is 0 and which includes the last byte of the requested section, shall be treated as a  
 23270 request to unlock from the start of the requested section with a size equal to 0. Otherwise, an  
 23271 F\_ULOCK request shall attempt to unlock only the requested section.
- 23272 Attempting to lock a section of a file that is associated with a buffered stream produces  
 23273 unspecified results.
- 23274 **RETURN VALUE**
- 23275 Upon successful completion, *lockf()* shall return 0. Otherwise, it shall return -1, set *errno* to  
 23276 indicate an error, and existing locks shall not be changed.
- 23277 **ERRORS**
- 23278 The *lockf()* function shall fail if:
- 23279 [EBADF] The *fildev* argument is not a valid open file descriptor; or *function* is F\_LOCK  
 23280 or F\_TLOCK and *fildev* is not a valid file descriptor open for writing.
- 23281 [EACCES] or [EAGAIN]  
 23282 The *function* argument is F\_TLOCK or F\_TEST and the section is already  
 23283 locked by another process.
- 23284 [EDEADLK] The *function* argument is F\_LOCK and a deadlock is detected.
- 23285 [EINTR] A signal was caught during execution of the function.
- 23286 [EINVAL] The *function* argument is not one of F\_LOCK, F\_TLOCK, F\_TEST, or  
 23287 F\_ULOCK; or *size* plus the current file offset is less than 0.
- 23288 [EOVERFLOW] The offset of the first, or if *size* is not 0 then the last, byte in the requested  
 23289 section cannot be represented correctly in an object of type *off\_t*.
- 23290 The *lockf()* function may fail if:
- 23291 [EAGAIN] The *function* argument is F\_LOCK or F\_TLOCK and the file is mapped with  
 23292 *mmap()*.
- 23293 [EDEADLK] or [ENOLCK]  
 23294 The *function* argument is F\_LOCK, F\_TLOCK, or F\_ULOCK, and the request  
 23295 would cause the number of locks to exceed a system-imposed limit.
- 23296 [EOPNOTSUPP] or [EINVAL]  
 23297 The implementation does not support the locking of files of the type indicated  
 23298 by the *fildev* argument.

23299 **EXAMPLES**23300 **Locking a Portion of a File**

23301 In the following example, a file named `/home/cnd/mod1` is being modified. Other processes that  
 23302 use locking are prevented from changing it during this process. Only the first 10,000 bytes are  
 23303 locked, and the lock call fails if another process has any part of this area locked already.

```
23304 #include <fcntl.h>
23305 #include <unistd.h>

23306 int fildes;
23307 int status;
23308 ...
23309 fildes = open("/home/cnd/mod1", O_RDWR);
23310 status = lockf(fildes, F_TLOCK, (off_t)10000);
```

23311 **APPLICATION USAGE**

23312 Record-locking should not be used in combination with the `fopen()`, `fread()`, `fwrite()`, and other  
 23313 `stdio` functions. Instead, the more primitive, non-buffered functions (such as `open()`) should be  
 23314 used. Unexpected results may occur in processes that do buffering in the user address space. The  
 23315 process may later read/write data which is/was locked. The `stdio` functions are the most  
 23316 common source of unexpected buffering.

23317 The `alarm()` function may be used to provide a timeout facility in applications requiring it.

23318 **RATIONALE**

23319 None.

23320 **FUTURE DIRECTIONS**

23321 None.

23322 **SEE ALSO**

23323 `alarm()`, `chmod()`, `close()`, `creat()`, `fcntl()`, `fopen()`, `mmap()`, `open()`, `read()`, `write()`, the Base  
 23324 Definitions volume of IEEE Std 1003.1-200x, `<unistd.h>`

23325 **CHANGE HISTORY**

23326 First released in Issue 4, Version 2.

23327 **Issue 5**

23328 Moved from X/OPEN UNIX extension to BASE.

23329 Large File Summit extensions are added. In particular, the description of [EINVAL] is clarified  
 23330 and moved from optional to mandatory status.

23331 A note is added to the DESCRIPTION indicating the effects of attempting to lock a section of a  
 23332 file that is associated with a buffered stream.

23333 **Issue 6**

23334 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

## 23335 NAME

23336 log, logf, logl — natural logarithm function

## 23337 SYNOPSIS

23338 #include &lt;math.h&gt;

23339 double log(double x);

23340 float logf(float x);

23341 long double logl(long double x);

## 23342 DESCRIPTION

23343 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 23344 conflict between the requirements described here and the ISO C standard is unintentional. This  
 23345 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

23346 These functions shall compute the natural logarithm of their argument  $x$ ,  $\log_e(x)$ .

23347 An application wishing to check for error situations should set *errno* to zero and call  
 23348 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 23349 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 23350 zero, an error has occurred.

## 23351 RETURN VALUE

23352 Upon successful completion, these functions shall return the natural logarithm of  $x$ .

23353 If  $x$  is  $\pm 0$ , a pole error shall occur and *log()*, *logf()*, and *logl()* shall return  $-\text{HUGE\_VAL}$ ,  
 23354  $-\text{HUGE\_VALF}$ , and  $-\text{HUGE\_VALL}$ , respectively.

23355 MX For finite values of  $x$  that are less than 0, or if  $x$  is  $-\text{Inf}$ , a domain error shall occur, and either a  
 23356 NaN (if supported), or an implementation-defined value shall be returned.

23357 MX If  $x$  is NaN, a NaN shall be returned.23358 If  $x$  is 1,  $+0$  shall be returned.23359 If  $x$  is  $+\text{Inf}$ ,  $x$  shall be returned.

## 23360 ERRORS

23361 These functions shall fail if:

23362 MX Domain Error The finite value of  $x$  is negative, or  $x$  is  $-\text{Inf}$ .

23363 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero, |  
 23364 then *errno* shall be set to [EDOM]. If the integer expression (math\_errhandling |  
 23365 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception |  
 23366 shall be raised. |

23367 Pole Error The value of  $x$  is zero.

23368 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero, |  
 23369 then *errno* shall be set to [ERANGE]. If the integer expression |  
 23370 (math\_errhandling & MATH\_ERREXCEPT) is non-zero, then the divide-by- |  
 23371 zero floating-point exception shall be raised. |

23372 **EXAMPLES**

23373 None.

23374 **APPLICATION USAGE**

23375 On error, the expressions (math\_errhandling & MATH\_ERRNO) and (math\_errhandling &  
23376 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

23377 **RATIONALE**

23378 None.

23379 **FUTURE DIRECTIONS**

23380 None.

23381 **SEE ALSO**

23382 *exp()*, *feclearexcept()*, *fetetestexcept()*, *isnan()*, *log10()*, *log1p()*, the Base Definitions volume of |  
23383 IEEE Std 1003.1-200x, Section 4.18, Treatment of Error Conditions for Mathematical Functions, |  
23384 <math.h>

23385 **CHANGE HISTORY**

23386 First released in Issue 1. Derived from Issue 1 of the SVID.

23387 **Issue 5**

23388 The DESCRIPTION is updated to indicate how an application should check for an error. This  
23389 text was previously published in the APPLICATION USAGE section.

23390 **Issue 6**

23391 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

23392 The *logf()* and *logl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

23393 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
23394 revised to align with the ISO/IEC 9899:1999 standard.

23395 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
23396 marked.

## 23397 NAME

23398 log10, log10f, log10l — base 10 logarithm function

## 23399 SYNOPSIS

23400 #include &lt;math.h&gt;

23401 double log10(double x);

23402 float log10f(float x);

23403 long double log10l(long double x);

## 23404 DESCRIPTION

23405 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
 23406 conflict between the requirements described here and the ISO C standard is unintentional. This  
 23407 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

23408 These functions shall compute the base 10 logarithm of their argument  $x$ ,  $\log_{10}(x)$ .

23409 An application wishing to check for error situations should set *errno* to zero and call  
 23410 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 23411 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 23412 zero, an error has occurred.

## 23413 RETURN VALUE

23414 Upon successful completion, these functions shall return the base 10 logarithm of  $x$ .

23415 If  $x$  is  $\pm 0$ , a pole error shall occur and *log10()*, *log10f()*, and *log10l()* shall return  $-\text{HUGE\_VAL}$ ,  
 23416  $-\text{HUGE\_VALF}$ , and  $-\text{HUGE\_VALL}$ , respectively.

23417 MX For finite values of  $x$  that are less than 0, or if  $x$  is  $-\text{Inf}$ , a domain error shall occur, and either a  
 23418 NaN (if supported), or an implementation-defined value shall be returned.

23419 MX If  $x$  is NaN, a NaN shall be returned.23420 If  $x$  is 1,  $+0$  shall be returned.23421 If  $x$  is  $+\text{Inf}$ ,  $+\text{Inf}$  shall be returned.

## 23422 ERRORS

23423 These functions shall fail if:

23424 MX Domain Error The finite value of  $x$  is negative, or  $x$  is  $-\text{Inf}$ .

23425 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero, |  
 23426 then *errno* shall be set to [EDOM]. If the integer expression (math\_errhandling |  
 23427 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception |  
 23428 shall be raised. |

23429 Pole Error The value of  $x$  is zero.

23430 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero, |  
 23431 then *errno* shall be set to [ERANGE]. If the integer expression |  
 23432 (math\_errhandling & MATH\_ERREXCEPT) is non-zero, then the divide-by- |  
 23433 zero floating-point exception shall be raised. |

23434 **EXAMPLES**

23435 None.

23436 **APPLICATION USAGE**

23437 On error, the expressions (math\_errhandling & MATH\_ERRNO) and (math\_errhandling &  
23438 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

23439 **RATIONALE**

23440 None.

23441 **FUTURE DIRECTIONS**

23442 None.

23443 **SEE ALSO**

23444 *feclearexcept()*, *fetetestexcept()*, *isnan()*, *log()*, *pow()*, the Base Definitions volume of |  
23445 IEEE Std 1003.1-200x, Section 4.18, Treatment of Error Conditions for Mathematical Functions, |  
23446 <math.h>

23447 **CHANGE HISTORY**

23448 First released in Issue 1. Derived from Issue 1 of the SVID.

23449 **Issue 5**

23450 The DESCRIPTION is updated to indicate how an application should check for an error. This  
23451 text was previously published in the APPLICATION USAGE section.

23452 **Issue 6**

23453 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

23454 The *log10f()* and *log10l()* functions are added for alignment with the ISO/IEC 9899:1999  
23455 standard.

23456 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
23457 revised to align with the ISO/IEC 9899:1999 standard.

23458 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
23459 marked.

## 23460 NAME

23461 log1p, log1pf, log1pl — compute a natural logarithm

## 23462 SYNOPSIS

23463 #include &lt;math.h&gt;

23464 double log1p(double x);

23465 float log1pf(float x);

23466 long double log1pl(long double x);

## 23467 DESCRIPTION

23468 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 23469 conflict between the requirements described here and the ISO C standard is unintentional. This  
 23470 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

23471 These functions shall compute  $\log_e(1.0 + x)$ .

23472 An application wishing to check for error situations should set *errno* to zero and call  
 23473 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 23474 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 23475 zero, an error has occurred.

## 23476 RETURN VALUE

23477 Upon successful completion, these functions shall return the natural logarithm of  $1.0 + x$ .

23478 If  $x$  is  $-1$ , a pole error shall occur and *log1p()*, *log1pf()*, and *log1pl()* shall return  $-\text{HUGE\_VAL}$ ,  
 23479  $-\text{HUGE\_VALF}$ , and  $-\text{HUGE\_VALL}$ , respectively.

23480 MX For finite values of  $x$  that are less than  $-1$ , or if  $x$  is  $-\text{Inf}$ , a domain error shall occur, and either a  
 23481 NaN (if supported), or an implementation-defined value shall be returned.

23482 MX If  $x$  is NaN, a NaN shall be returned.23483 If  $x$  is  $\pm 0$ , or  $+\text{Inf}$ ,  $x$  shall be returned.23484 If  $x$  is subnormal, a range error may occur and  $x$  should be returned.

## 23485 ERRORS

23486 These functions shall fail if:

23487 MX Domain Error The finite value of  $x$  is less than  $-1$ , or  $x$  is  $-\text{Inf}$ .

23488 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero, |  
 23489 then *errno* shall be set to [EDOM]. If the integer expression (math\_errhandling |  
 23490 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception |  
 23491 shall be raised. |

23492 Pole Error The value of  $x$  is  $-1$ .

23493 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero, |  
 23494 then *errno* shall be set to [ERANGE]. If the integer expression |  
 23495 (math\_errhandling & MATH\_ERREXCEPT) is non-zero, then the divide-by- |  
 23496 zero floating-point exception shall be raised. |

23497 These functions may fail if:

23498 MX Range Error The value of  $x$  is subnormal.

23499 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero, |  
 23500 then *errno* shall be set to [ERANGE]. If the integer expression |  
 23501 (math\_errhandling & MATH\_ERREXCEPT) is non-zero, then the underflow |  
 23502 floating-point exception shall be raised. |

23503 **EXAMPLES**

23504 None.

23505 **APPLICATION USAGE**

23506 On error, the expressions (math\_errhandling & MATH\_ERRNO) and (math\_errhandling &  
23507 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

23508 **RATIONALE**

23509 None.

23510 **FUTURE DIRECTIONS**

23511 None.

23512 **SEE ALSO**

23513 *feclearexcept()*, *fetestexcept()*, *log()*, the Base Definitions volume of IEEE Std 1003.1-200x, Section |  
23514 4.18, Treatment of Error Conditions for Mathematical Functions, <math.h> |

23515 **CHANGE HISTORY**

23516 First released in Issue 4, Version 2.

23517 **Issue 5**

23518 Moved from X/OPEN UNIX extension to BASE.

23519 **Issue 6**

23520 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

23521 The *log1p()* function is no longer marked as an extension.

23522 The *log1pf()* and *log1pl()* functions are added for alignment with the ISO/IEC 9899:1999  
23523 standard.

23524 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
23525 revised to align with the ISO/IEC 9899:1999 standard.

23526 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
23527 marked.

23528 **NAME**

23529 log2, log2f, log2l — compute base 2 logarithm functions

23530 **SYNOPSIS**

23531 #include &lt;math.h&gt;

23532 double log2(double x);

23533 float log2f(float x);

23534 long double log2l(long double x);

23535 **DESCRIPTION**

23536 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 23537 conflict between the requirements described here and the ISO C standard is unintentional. This  
 23538 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

23539 These functions shall compute the base 2 logarithm of their argument  $x$ ,  $\log_2(x)$ .

23540 An application wishing to check for error situations should set *errno* to zero and call  
 23541 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 23542 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 23543 zero, an error has occurred.

23544 **RETURN VALUE**23545 Upon successful completion, these functions shall return the base 2 logarithm of  $x$ .

23546 If  $x$  is  $\pm 0$ , a pole error shall occur and *log2()*, *log2f()*, and *log2l()* shall return `-HUGE_VAL`,  
 23547 `-HUGE_VALF`, and `-HUGE_VALL`, respectively.

23548 **MX** For finite values of  $x$  that are less than 0, or if  $x$  is `-Inf` a domain error shall occur, and either a  
 23549 NaN (if supported), or an implementation-defined value shall be returned.

23550 **MX** If  $x$  is NaN, a NaN shall be returned.23551 If  $x$  is 1, `+0` shall be returned.23552 If  $x$  is `+Inf`,  $x$  shall be returned.23553 **ERRORS**

23554 These functions shall fail if:

23555 **MX** Domain Error The finite value of  $x$  is less than zero, or  $x$  is `-Inf`.

23556 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero, |  
 23557 then *errno* shall be set to [EDOM]. If the integer expression (math\_errhandling |  
 23558 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception |  
 23559 shall be raised. |

23560 Pole Error The value of  $x$  is zero.

23561 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero, |  
 23562 then *errno* shall be set to [ERANGE]. If the integer expression |  
 23563 (math\_errhandling & MATH\_ERREXCEPT) is non-zero, then the divide-by- |  
 23564 zero floating-point exception shall be raised. |

23565 **EXAMPLES**

23566 None.

23567 **APPLICATION USAGE**

23568 On error, the expressions (math\_errhandling & MATH\_ERRNO) and (math\_errhandling &  
23569 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

23570 **RATIONALE**

23571 None.

23572 **FUTURE DIRECTIONS**

23573 None.

23574 **SEE ALSO**

23575 *feclearexcept()*, *fetestexcept()*, *log()*, the Base Definitions volume of IEEE Std 1003.1-200x, Section |  
23576 4.18, Treatment of Error Conditions for Mathematical Functions, <math.h> |

23577 **CHANGE HISTORY**

23578 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

23579 **NAME**

23580 logb, logbf, logbl — radix-independent exponent

23581 **SYNOPSIS**

23582 #include &lt;math.h&gt;

23583 double logb(double x);

23584 float logbf(float x);

23585 long double logbl(long double x);

23586 **DESCRIPTION**

23587 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 23588 conflict between the requirements described here and the ISO C standard is unintentional. This  
 23589 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

23590 These functions shall compute the exponent of  $x$ , which is the integral part of  $\log_r |x|$ , as a  
 23591 signed floating-point value, for non-zero  $x$ , where  $r$  is the radix of the machine's floating-point  
 23592 arithmetic, which is the value of FLT\_RADIX defined in the <float.h> header.

23593 If  $x$  is subnormal it is treated as though it were normalized; thus for finite positive  $x$ :

23594 
$$1 \leq x * FLT\_RADIX^{-\logb(x)} < FLT\_RADIX$$

23595 An application wishing to check for error situations should set *errno* to zero and call  
 23596 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 23597 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 23598 zero, an error has occurred.

23599 **RETURN VALUE**23600 Upon successful completion, these functions shall return the exponent of  $x$ .

23601 If  $x$  is  $\pm 0$ , a pole error shall occur and *logb*(), *logbf*(), and *logbl*() shall return `-HUGE_VAL`,  
 23602 `-HUGE_VALF`, and `-HUGE_VALL`, respectively.

23603 **MX** If  $x$  is NaN, a NaN shall be returned.23604 If  $x$  is  $\pm\text{Inf}$ , `+Inf` shall be returned.23605 **ERRORS**

23606 These functions shall fail if:

23607 Pole Error The value of  $x$  is  $\pm 0$ .

23608 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero, |  
 23609 then *errno* shall be set to [ERANGE]. If the integer expression |  
 23610 (math\_errhandling & MATH\_ERREXCEPT) is non-zero, then the divide-by- |  
 23611 zero floating-point exception shall be raised. |

23612 **EXAMPLES**

23613 None.

23614 **APPLICATION USAGE**

23615 On error, the expressions (math\_errhandling & MATH\_ERRNO) and (math\_errhandling &  
 23616 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

23617 **RATIONALE**

23618 None.

23619 **FUTURE DIRECTIONS**

23620 None.

23621 **SEE ALSO**

23622 *feclearexcept()*, *fetestexcept()*, *ilogb()*, *scalb()*, the Base Definitions volume of IEEE Std 1003.1-200x, |  
23623 Section 4.18, Treatment of Error Conditions for Mathematical Functions, <float.h>, <math.h> |

23624 **CHANGE HISTORY**

23625 First released in Issue 4, Version 2.

23626 **Issue 5**

23627 Moved from X/OPEN UNIX extension to BASE.

23628 **Issue 6**23629 The *logb()* function is no longer marked as an extension.23630 The *logbf()* and *logbl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

23631 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
23632 revised to align with the ISO/IEC 9899:1999 standard.

23633 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
23634 marked.

23635 **NAME**

23636       logf — natural logarithm function

23637 **SYNOPSIS**

23638       #include <math.h>

23639       float logf(float x);

23640 **DESCRIPTION**

23641       Refer to *log()*.

23642 **NAME**

23643       logl — natural logarithm function

23644 **SYNOPSIS**

23645       #include <math.h>

23646       long double logl(long double x);

23647 **DESCRIPTION**

23648       Refer to *log()*.

23649 **NAME**

23650 longjmp — non-local goto

23651 **SYNOPSIS**

23652 #include &lt;setjmp.h&gt;

23653 void longjmp(jmp\_buf env, int val);

23654 **DESCRIPTION**

23655 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
 23656 conflict between the requirements described here and the ISO C standard is unintentional. This  
 23657 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

23658 The *longjmp()* function shall restore the environment saved by the most recent invocation of  
 23659 *setjmp()* in the same thread, with the corresponding **jmp\_buf** argument. If there is no such  
 23660 invocation, or if the function containing the invocation of *setjmp()* has terminated execution in  
 23661 the interim, or if the invocation of *setjmp()* was within the scope of an identifier with variably  
 23662 cx modified type and execution has left that scope in the interim, the behavior is undefined. It is  
 23663 unspecified whether *longjmp()* restores the signal mask, leaves the signal mask unchanged, or  
 23664 restores it to its value at the time *setjmp()* was called.

23665 All accessible objects have values, and all other components of the abstract machine have state  
 23666 (for example, floating-point status flags and open files), as of the time *longjmp()* was called,  
 23667 except that the values of objects of automatic storage duration are unspecified if they meet all  
 23668 the following conditions:

- 23669 • They are local to the function containing the corresponding *setjmp()* invocation.
- 23670 • They do not have volatile-qualified type.
- 23671 • They are changed between the *setjmp()* invocation and *longjmp()* call.

23672 cx As it bypasses the usual function call and return mechanisms, *longjmp()* shall execute correctly  
 23673 in contexts of interrupts, signals, and any of their associated functions. However, if *longjmp()* is  
 23674 invoked from a nested signal handler (that is, from a function invoked as a result of a signal  
 23675 raised during the handling of another signal), the behavior is undefined.

23676 The effect of a call to *longjmp()* where initialization of the **jmp\_buf** structure was not performed  
 23677 in the calling thread is undefined.

23678 **RETURN VALUE**

23679 After *longjmp()* is completed, program execution continues as if the corresponding invocation of  
 23680 *setjmp()* had just returned the value specified by *val*. The *longjmp()* function shall not cause  
 23681 *setjmp()* to return 0; if *val* is 0, *setjmp()* shall return 1.

23682 **ERRORS**

23683 No errors are defined.

23684 **EXAMPLES**

23685 None.

23686 **APPLICATION USAGE**

23687 Applications whose behavior depends on the value of the signal mask should not use *longjmp()*  
 23688 and *setjmp()*, since their effect on the signal mask is unspecified, but should instead use the  
 23689 *siglongjmp()* and *sigsetjmp()* functions (which can save and restore the signal mask under  
 23690 application control).

23691 **RATIONALE**

23692 None.

23693 **FUTURE DIRECTIONS**

23694 None.

23695 **SEE ALSO**

23696 *setjmp()*, *sigaction()*, *siglongjmp()*, *sigsetjmp()*, the Base Definitions volume of  
23697 IEEE Std 1003.1-200x, <**setjmp.h**>

23698 **CHANGE HISTORY**

23699 First released in Issue 1. Derived from Issue 1 of the SVID.

23700 **Issue 5**

23701 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

23702 **Issue 6**

23703 Extensions beyond the ISO C standard are now marked.

23704 The following new requirements on POSIX implementations derive from alignment with the  
23705 Single UNIX Specification:

- 23706 • The DESCRIPTION now explicitly makes *longjmp()*'s effect on the signal mask unspecified.

23707 The DESCRIPTION is updated for alignment with the ISO/IEC 9899:1999 standard.

23708 **NAME**

23709       lrand48 — generate uniformly distributed pseudo-random non-negative long integers

23710 **SYNOPSIS**

23711 xSI     #include <stdlib.h>

23712       long lrand48(void);

23713

23714 **DESCRIPTION**

23715       Refer to *drand48()*.

23716 **NAME**

23717 lrint, lrintf, lrintl — round to nearest integer value using current rounding direction

23718 **SYNOPSIS**

```
23719 #include <math.h>

23720 long lrint(double x);
23721 long lrintf(float x);
23722 long lrintl(long double x);
```

23723 **DESCRIPTION**

23724 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 23725 conflict between the requirements described here and the ISO C standard is unintentional. This  
 23726 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

23727 These functions shall round their argument to the nearest integer value, rounding according to  
 23728 the current rounding direction.

23729 An application wishing to check for error situations should set *errno* to zero and call  
 23730 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 23731 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 23732 zero, an error has occurred.

23733 **RETURN VALUE**

23734 Upon successful completion, these functions shall return the rounded integer value.

23735 **MX** If *x* is NaN, a domain error shall occur, and an unspecified value is returned.

23736 If *x* is +Inf, a domain error shall occur and an unspecified value is returned.

23737 If *x* is -Inf, a domain error shall occur and an unspecified value is returned.

23738 If the correct value is positive and too large to represent as a **long**, a domain error shall occur  
 23739 and an unspecified value is returned.

23740 If the correct value is negative and too large to represent as a **long**, a domain error shall occur  
 23741 and an unspecified value is returned.

23742 **ERRORS**

23743 These functions shall fail if:

23744 **MX** Domain Error The *x* argument is NaN or ±Inf, or the correct value is not representable as an  
 23745 integer.

23746 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero,  
 23747 then *errno* shall be set to [EDOM]. If the integer expression (math\_errhandling  
 23748 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception  
 23749 shall be raised.

23750 **EXAMPLES**

23751 None.

23752 **APPLICATION USAGE**

23753 On error, the expressions (math\_errhandling & MATH\_ERRNO) and (math\_errhandling &  
 23754 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

23755 **RATIONALE**

23756 These functions provide floating-to-integer conversions. They round according to the current  
 23757 rounding direction. If the rounded value is outside the range of the return type, the numeric  
 23758 result is unspecified and the invalid floating-point exception is raised. When they raise no other  
 23759 floating-point exception and the result differs from the argument, they raise the inexact

23760 floating-point exception.

23761 **FUTURE DIRECTIONS**

23762 None.

23763 **SEE ALSO**

23764 *feclearexcept()*, *fetetestexcept()*, *llrint()*, the Base Definitions volume of IEEE Std 1003.1-200x, |

23765 Section 4.18, Treatment of Error Conditions for Mathematical Functions, <math.h> |

23766 **CHANGE HISTORY**

23767 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

23768 **NAME**

23769 lround, lroundf, lroundl — round to nearest integer value

23770 **SYNOPSIS**

```
23771 #include <math.h>
23772 long lround(double x);
23773 long lroundf(float x);
23774 long lroundl(long double x);
```

23775 **DESCRIPTION**

23776 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 23777 conflict between the requirements described here and the ISO C standard is unintentional. This  
 23778 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

23779 These functions shall round their argument to the nearest integer value, rounding halfway cases  
 23780 away from zero, regardless of the current rounding direction.

23781 An application wishing to check for error situations should set *errno* to zero and call  
 23782 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 23783 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 23784 zero, an error has occurred.

23785 **RETURN VALUE**

23786 Upon successful completion, these functions shall return the rounded integer value.

23787 **MX** If *x* is NaN, a domain error shall occur, and an unspecified value is returned.23788 If *x* is +Inf, a domain error shall occur and an unspecified value is returned.23789 If *x* is -Inf, a domain error shall occur and an unspecified value is returned.

23790 If the correct value is positive and too large to represent as a **long**, a domain error shall occur |  
 23791 and an unspecified value is returned.

23792 If the correct value is negative and too large to represent as a **long**, a domain error shall occur |  
 23793 and an unspecified value is returned.

23794 **ERRORS**

23795 These functions shall fail if:

23796 **MX** Domain Error The *x* argument is NaN or ±Inf, or the correct value is not representable as an  
 23797 integer.

23798 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero, |  
 23799 then *errno* shall be set to [EDOM]. If the integer expression (math\_errhandling  
 23800 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception  
 23801 shall be raised. |

23802 **EXAMPLES**

23803 None.

23804 **APPLICATION USAGE**

23805 On error, the expressions (math\_errhandling & MATH\_ERRNO) and (math\_errhandling &  
 23806 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

23807 **RATIONALE**

23808 These functions provide floating-to-integer conversions. They round according to the current  
 23809 rounding direction. If the rounded value is outside the range of the return type, the numeric  
 23810 result is unspecified and the invalid floating-point exception is raised. When they raise no other  
 23811 floating-point exception and the result differs from the argument, they raise the inexact

23812 floating-point exception.

23813 These functions differ from the *lrint()* functions in the default rounding direction, with the  
23814 *lround()* functions rounding halfway cases away from zero and needing not to raise the inexact  
23815 floating-point exception for non-integer arguments that round to within the range of the return  
23816 type.

23817 **FUTURE DIRECTIONS**

23818 None.

23819 **SEE ALSO**

23820 *feclearexcept()*, *fetestexcept()*, *llround()*, the Base Definitions volume of IEEE Std 1003.1-200x, |  
23821 Section 4.18, Treatment of Error Conditions for Mathematical Functions, <math.h> |

23822 **CHANGE HISTORY**

23823 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

23824 **NAME**

23825 lsearch, lfind — linear search and update

23826 **SYNOPSIS**

```
23827 xSI #include <search.h>
23828 void *lsearch(const void *key, void *base, size_t *nel, size_t width,
23829             int (*compar)(const void *, const void *));
23830 void *lfind(const void *key, const void *base, size_t *nel,
23831            size_t width, int (*compar)(const void *, const void *));
23832
```

23833 **DESCRIPTION**

23834 The *lsearch()* function shall linearly search the table and return a pointer into the table for the  
 23835 matching entry. If the entry does not occur, it shall be added at the end of the table. The *key*  
 23836 argument points to the entry to be sought in the table. The *base* argument points to the first  
 23837 element in the table. The *width* argument is the size of an element in bytes. The *nel* argument  
 23838 points to an integer containing the current number of elements in the table. The integer to which  
 23839 *nel* points shall be incremented if the entry is added to the table. The *compar* argument points to  
 23840 a comparison function which the application shall supply (for example, *strcmp()*). It is called  
 23841 with two arguments that point to the elements being compared. The application shall ensure  
 23842 that the function returns 0 if the elements are equal, and non-zero otherwise.

23843 The *lfind()* function shall be equivalent to *lsearch()*, except that if the entry is not found, it is not  
 23844 added to the table. Instead, a null pointer is returned.

23845 **RETURN VALUE**

23846 If the searched for entry is found, both *lsearch()* and *lfind()* shall return a pointer to it. Otherwise,  
 23847 *lfind()* shall return a null pointer and *lsearch()* shall return a pointer to the newly added element.

23848 Both functions shall return a null pointer in case of error.

23849 **ERRORS**

23850 No errors are defined.

23851 **EXAMPLES**23852 **Storing Strings in a Table**

23853 This fragment reads in less than or equal to TABSIZE strings of length less than or equal to  
 23854 ELSIZE and stores them in a table, eliminating duplicates.

```
23855 #include <stdio.h>
23856 #include <string.h>
23857 #include <search.h>
23858 #define TABSIZE 50
23859 #define ELSIZE 120
23860 ...
23861     char line[ELSIZE], tab[TABSIZE][ELSIZE];
23862     size_t nel = 0;
23863     ...
23864     while (fgets(line, ELSIZE, stdin) != NULL && nel < TABSIZE)
23865         (void) lsearch(line, tab, &nel,
23866                      ELSIZE, (int (*)(const void *, const void *)) strcmp);
23867     ...
```

23868 **Finding a Matching Entry**

23869 The following example finds any line that reads "This is a test.".

```
23870 #include <search.h>
23871 #include <string.h>
23872 ...
23873 char line[ELSIZE], tab[TABSIZE][ELSIZE];
23874 size_t nel = 0;
23875 char *findline;
23876 void *entry;

23877 findline = "This is a test.\n";

23878 entry = lfind(findline, tab, &nel, ELSIZE, (
23879     int (*)(const void *, const void *)) strcmp);
```

23880 **APPLICATION USAGE**

23881 The comparison function need not compare every byte, so arbitrary data may be contained in  
23882 the elements in addition to the values being compared.

23883 Undefined results can occur if there is not enough room in the table to add a new item.

23884 **RATIONALE**

23885 None.

23886 **FUTURE DIRECTIONS**

23887 None.

23888 **SEE ALSO**

23889 *hcreate()*, *tsearch()*, the Base Definitions volume of IEEE Std 1003.1-200x, <**search.h**>

23890 **CHANGE HISTORY**

23891 First released in Issue 1. Derived from Issue 1 of the SVID.

23892 **Issue 6**

23893 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

23894 **NAME**

23895 lseek — move the read/write file offset

23896 **SYNOPSIS**

23897 #include &lt;unistd.h&gt;

23898 off\_t lseek(int *fildev*, off\_t *offset*, int *whence*);23899 **DESCRIPTION**23900 The *lseek()* function shall set the file offset for the open file description associated with the file descriptor *fildev*, as follows:

- 23902 • If *whence* is SEEK\_SET, the file offset shall be set to *offset* bytes. |
- 23903 • If *whence* is SEEK\_CUR, the file offset shall be set to its current location plus *offset*. |
- 23904 • If *whence* is SEEK\_END, the file offset shall be set to the size of the file plus *offset*. |

23905 The symbolic constants SEEK\_SET, SEEK\_CUR, and SEEK\_END are defined in &lt;unistd.h&gt;.

23906 The behavior of *lseek()* on devices which are incapable of seeking is implementation-defined.  
23907 The value of the file offset associated with such a device is undefined.23908 The *lseek()* function shall allow the file offset to be set beyond the end of the existing data in the file. If data is later written at this point, subsequent reads of data in the gap shall return bytes with the value 0 until data is actually written into the gap.23911 The *lseek()* function shall not, by itself, extend the size of a file.23912 SHM If *fildev* refers to a shared memory object, the result of the *lseek()* function is unspecified.23913 TYM If *fildev* refers to a typed memory object, the result of the *lseek()* function is unspecified.23914 **RETURN VALUE**23915 Upon successful completion, the resulting offset, as measured in bytes from the beginning of the file, shall be returned. Otherwise, (off\_t)-1 shall be returned, *errno* shall be set to indicate the error, and the file offset shall remain unchanged.23918 **ERRORS**23919 The *lseek()* function shall fail if:

- 23920 [EBADF] The *fildev* argument is not an open file descriptor.
- 23921 [EINVAL] The *whence* argument is not a proper value, or the resulting file offset would be negative for a regular file, block special file, or directory.
- 23922
- 23923 [EOVERFLOW] The resulting file offset would be a value which cannot be represented correctly in an object of type **off\_t**.
- 23924
- 23925 [ESPIPE] The *fildev* argument is associated with a pipe, FIFO, or socket.

23926 **EXAMPLES**

23927 None.

23928 **APPLICATION USAGE**

23929 None.

23930 **RATIONALE**23931 The ISO C standard includes the functions *fgetpos()* and *fsetpos()*, which work on very large files by use of a special positioning type.23933 Although *lseek()* may position the file offset beyond the end of the file, this function does not  
23934 itself extend the size of the file. While the only function in IEEE Std 1003.1-200x that may directly |

- 23935 extend the size of the file is *write()*, *truncate()*, and *ftruncate()*, several functions originally |  
23936 derived from the ISO C standard, such as *fwrite()*, *fprintf()*, and so on, may do so (by causing |  
23937 calls on *write()*).
- 23938 An invalid file offset that would cause [EINVAL] to be returned may be both implementation-  
23939 defined and device-dependent (for example, memory may have few invalid values). A negative  
23940 file offset may be valid for some devices in some implementations.
- 23941 The POSIX.1-1990 standard did not specifically prohibit *lseek()* from returning a negative offset.  
23942 Therefore, an application was required to clear *errno* prior to the call and check *errno* upon return  
23943 to determine whether a return value of (*off\_t*)-1 is a negative offset or an indication of an error  
23944 condition. The standard developers did not wish to require this action on the part of a |  
23945 conforming application, and chose to require that *errno* be set to [EINVAL] when the resulting |  
23946 file offset would be negative for a regular file, block special file, or directory.
- 23947 **FUTURE DIRECTIONS**
- 23948 None.
- 23949 **SEE ALSO**
- 23950 *open()*, the Base Definitions volume of IEEE Std 1003.1-200x, <**sys/types.h**>, <**unistd.h**>
- 23951 **CHANGE HISTORY**
- 23952 First released in Issue 1. Derived from Issue 1 of the SVID.
- 23953 **Issue 5**
- 23954 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension.
- 23955 Large File Summit extensions are added.
- 23956 **Issue 6**
- 23957 In the SYNOPSIS, the optional include of the <**sys/types.h**> header is removed.
- 23958 The following new requirements on POSIX implementations derive from alignment with the  
23959 Single UNIX Specification:
- 23960 • The requirement to include <**sys/types.h**> has been removed. Although <**sys/types.h**> was  
23961 required for conforming implementations of previous POSIX specifications, it was not  
23962 required for UNIX applications.
  - 23963 • The [EOVERFLOW] error condition is added. This change is to support large files.
- 23964 An additional [ESPIPE] error condition is added for sockets.
- 23965 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that  
23966 *lseek()* results are unspecified for typed memory objects.

23967 **NAME**

23968           lstat — get symbolic link status

23969 **SYNOPSIS**

23970           #include &lt;sys/stat.h&gt;

23971           int lstat(const char \*restrict *path*, struct stat \*restrict *buf*);23972 **DESCRIPTION**

23973           The *lstat()* function shall be equivalent to *stat()*, except when *path* refers to a symbolic link. In  
 23974           that case *lstat()* shall return information about the link, while *stat()* shall return information  
 23975           about the file the link references.

23976           For symbolic links, the *st\_mode* member shall contain meaningful information when used with  
 23977           the file type macros, and the *st\_size* member shall contain the length of the pathname contained  
 23978           in the symbolic link. File mode bits and the contents of the remaining members of the **stat**  
 23979           structure are unspecified. The value returned in the *st\_size* member is the length of the contents  
 23980           of the symbolic link, and does not count any trailing null.

23981 **RETURN VALUE**

23982           Upon successful completion, *lstat()* shall return 0. Otherwise, it shall return  $-1$  and set *errno* to  
 23983           indicate the error.

23984 **ERRORS**23985           The *lstat()* function shall fail if:

23986           [EACCES]           A component of the path prefix denies search permission.

23987           [EIO]             An error occurred while reading from the file system.

23988           [ELOOP]           A loop exists in symbolic links encountered during resolution of the *path*  
 23989           argument.

23990           [ENAMETOOLONG]

23991                               The length of a pathname exceeds {PATH\_MAX} or a pathname component is  
 23992                               longer than {NAME\_MAX}.

23993           [ENOTDIR]         A component of the path prefix is not a directory.

23994           [ENOENT]         A component of *path* does not name an existing file or *path* is an empty string.23995           [EOVERFLOW]       The file size in bytes or the number of blocks allocated to the file or the file  
 23996           serial number cannot be represented correctly in the structure pointed to by  
 23997           *buf*.23998           The *lstat()* function may fail if:23999           [ELOOP]           More than {SYMLOOP\_MAX} symbolic links were encountered during  
 24000           resolution of the *path* argument.

24001           [ENAMETOOLONG]

24002                               As a result of encountering a symbolic link in resolution of the *path* argument,  
 24003                               the length of the substituted pathname string exceeded {PATH\_MAX}.24004           [EOVERFLOW]       One of the members is too large to store into the structure pointed to by the  
 24005           *buf* argument.

24006 **EXAMPLES**24007 **Obtaining Symbolic Link Status Information**

24008 The following example shows how to obtain status information for a symbolic link named  
24009 **/modules/pass1**. The structure variable *buffer* is defined for the **stat** structure. If the *path*  
24010 argument specified the filename for the file pointed to by the symbolic link (**/home/cnd/mod1**),  
24011 the results of calling the function would be the same as those returned by a call to the *stat()*  
24012 function.

```
24013 #include <sys/stat.h>
24014 struct stat buffer;
24015 int status;
24016 ...
24017 status = lstat("/modules/pass1", &buffer);
```

24018 **APPLICATION USAGE**

24019 None.

24020 **RATIONALE**

24021 The *lstat()* function is not required to update the time-related fields if the named file is not a  
24022 symbolic link. While the *st\_uid*, *st\_gid*, *st\_atime*, *st\_mtime*, and *st\_ctime* members of the **stat**  
24023 structure may apply to a symbolic link, they are not required to do so. No functions in  
24024 IEEE Std 1003.1-200x are required to maintain any of these time fields.

24025 **FUTURE DIRECTIONS**

24026 None.

24027 **SEE ALSO**

24028 *lstat()*, *readlink()*, *stat()*, *symlink()*, the Base Definitions volume of IEEE Std 1003.1-200x,  
24029 **<sys/stat.h>**

24030 **CHANGE HISTORY**

24031 First released in Issue 4, Version 2.

24032 **Issue 5**

24033 Moved from X/OPEN UNIX extension to BASE.

24034 Large File Summit extensions are added.

24035 **Issue 6**

24036 The following changes were made to align with the IEEE P1003.1a draft standard:

- 24037 • This function is now mandatory.
- 24038 • The [ELOOP] optional error condition is added.

24039 The **restrict** keyword is added to the *lstat()* prototype for alignment with the ISO/IEC 9899:1999  
24040 standard.

24041 **NAME**

24042       makecontext, swapcontext — manipulate user contexts

24043 **SYNOPSIS**

```
24044 xSI       #include <ucontext.h>

24045       void makecontext(ucontext_t *ucp, void (*func)(void),
24046                       int argc, ...);
24047       int swapcontext(ucontext_t *restrict oucp,
24048                       const ucontext_t *restrict ucp);
24049
```

24050 **DESCRIPTION**

24051       The *makecontext()* function shall modify the context specified by *ucp*, which has been initialized  
 24052       using *getcontext()*. When this context is resumed using *swapcontext()* or *setcontext()*, program  
 24053       execution shall continue by calling *func*, passing it the arguments that follow *argc* in the  
 24054       *makecontext()* call.

24055       Before a call is made to *makecontext()*, the application shall ensure that the context being  
 24056       modified has a stack allocated for it. The application shall ensure that the value of *argc* matches  
 24057       the number of integer arguments passed to *func*; otherwise, the behavior is undefined.

24058       The *uc\_link* member is used to determine the context that shall be resumed when the context  
 24059       being modified by *makecontext()* returns. The application shall ensure that the *uc\_link* member is  
 24060       initialized prior to the call to *makecontext()*.

24061       The *swapcontext()* function shall save the current context in the context structure pointed to by  
 24062       *oucp* and shall set the context to the context structure pointed to by *ucp*.

24063 **RETURN VALUE**

24064       Upon successful completion, *swapcontext()* shall return 0. Otherwise,  $-1$  shall be returned and  
 24065       *errno* set to indicate the error.

24066 **ERRORS**

24067       The *swapcontext()* function shall fail if:

24068       [ENOMEM]       The *ucp* argument does not have enough stack left to complete the operation.

24069 **EXAMPLES**

24070       The following example illustrates the use of *makecontext()*:

```
24071       #include <stdio.h>
24072       #include <ucontext.h>

24073       static ucontext_t ctx[3];

24074       static void
24075       f1 (void)
24076       {
24077           puts("start f1");
24078           swapcontext(&ctx[1], &ctx[2]);
24079           puts("finish f1");
24080       }

24081       static void
24082       f2 (void)
24083       {
24084           puts("start f2");
24085           swapcontext(&ctx[2], &ctx[1]);
```

```

24086         puts("finish f2");
24087     }
24088     int
24089     main (void)
24090     {
24091         char st1[8192];
24092         char st2[8192];
24093
24094         getcontext(&ctx[1]);
24095         ctx[1].uc_stack.ss_sp = st1;
24096         ctx[1].uc_stack.ss_size = sizeof st1;
24097         ctx[1].uc_link = &ctx[0];
24098         makecontext(&ctx[1], f1, 0);
24099
24100         getcontext(&ctx[2]);
24101         ctx[2].uc_stack.ss_sp = st2;
24102         ctx[2].uc_stack.ss_size = sizeof st2;
24103         ctx[2].uc_link = &ctx[1];
24104         makecontext(&ctx[2], f2, 0);
24105
24106         swapcontext(&ctx[0], &ctx[2]);
24107         return 0;
24108     }

```

#### 24106 APPLICATION USAGE

24107 None.

#### 24108 RATIONALE

24109 None.

#### 24110 FUTURE DIRECTIONS

24111 None.

#### 24112 SEE ALSO

24113 *exit()*, *getcontext()*, *sigaction()*, *sigprocmask()*, the Base Definitions volume of  
 24114 IEEE Std 1003.1-200x, <**ucontext.h**>

#### 24115 CHANGE HISTORY

24116 First released in Issue 4, Version 2.

#### 24117 Issue 5

24118 Moved from X/OPEN UNIX extension to BASE.

24119 In the ERRORS section, the description of [ENOMEM] is changed to apply to *swapcontext()* only.

#### 24120 Issue 6

24121 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

24122 The **restrict** keyword is added to the *swapcontext()* prototype for alignment with the  
 24123 ISO/IEC 9899:1999 standard.

24124 **NAME**

24125 malloc — a memory allocator

24126 **SYNOPSIS**

24127 #include &lt;stdlib.h&gt;

24128 void \*malloc(size\_t size);

24129 **DESCRIPTION**

24130 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
 24131 conflict between the requirements described here and the ISO C standard is unintentional. This  
 24132 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

24133 The *malloc()* function shall allocate unused space for an object whose size in bytes is specified by  
 24134 *size* and whose value is unspecified.

24135 The order and contiguity of storage allocated by successive calls to *malloc()* is unspecified. The  
 24136 pointer returned if the allocation succeeds shall be suitably aligned so that it may be assigned to  
 24137 a pointer to any type of object and then used to access such an object in the space allocated (until  
 24138 the space is explicitly freed or reallocated). Each such allocation shall yield a pointer to an object  
 24139 disjoint from any other object. The pointer returned points to the start (lowest byte address) of  
 24140 the allocated space. If the space cannot be allocated, a null pointer shall be returned. If the size of  
 24141 the space requested is 0, the behavior is implementation-defined: the value returned shall be  
 24142 either a null pointer or a unique pointer.

24143 **RETURN VALUE**

24144 Upon successful completion with *size* not equal to 0, *malloc()* shall return a pointer to the  
 24145 allocated space. If *size* is 0, either a null pointer or a unique pointer that can be successfully  
 24146 cx passed to *free()* shall be returned. Otherwise, it shall return a null pointer and set *errno* to  
 24147 indicate the error.

24148 **ERRORS**24149 The *malloc()* function shall fail if:

24150 cx [ENOMEM] Insufficient storage space is available.

24151 **EXAMPLES**

24152 None.

24153 **APPLICATION USAGE**

24154 None.

24155 **RATIONALE**

24156 None.

24157 **FUTURE DIRECTIONS**

24158 None.

24159 **SEE ALSO**24160 *calloc()*, *free()*, *realloc()*, the Base Definitions volume of IEEE Std 1003.1-200x, <stdlib.h>24161 **CHANGE HISTORY**

24162 First released in Issue 1. Derived from Issue 1 of the SVID.

24163 **Issue 6**

24164 Extensions beyond the ISO C standard are now marked.

24165 The following new requirements on POSIX implementations derive from alignment with the  
 24166 Single UNIX Specification:

24167

- In the RETURN VALUE section, the requirement to set *errno* to indicate an error is added.

24168

- The [ENOMEM] error condition is added.

24169 **NAME**

24170           mblen — get number of bytes in a character

24171 **SYNOPSIS**

24172           #include &lt;stdlib.h&gt;

24173           int mblen(const char \*s, size\_t n);

24174 **DESCRIPTION**

24175 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
 24176 conflict between the requirements described here and the ISO C standard is unintentional. This  
 24177 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

24178       If *s* is not a null pointer, *mblen()* shall determine the number of bytes constituting the character  
 24179 pointed to by *s*. Except that the shift state of *mbtowc()* is not affected, it shall be equivalent to:

24180       mbtowc((wchar\_t \*)0, s, n);

24181       The implementation shall behave as if no function defined in this volume of  
 24182 IEEE Std 1003.1-200x calls *mblen()*.

24183       The behavior of this function is affected by the *LC\_CTYPE* category of the current locale. For a  
 24184 state-dependent encoding, this function shall be placed into its initial state by a call for which its  
 24185 character pointer argument, *s*, is a null pointer. Subsequent calls with *s* as other than a null  
 24186 pointer shall cause the internal state of the function to be altered as necessary. A call with *s* as a  
 24187 null pointer shall cause this function to return a non-zero value if encodings have state  
 24188 dependency, and 0 otherwise. If the implementation employs special bytes to change the shift  
 24189 state, these bytes shall not produce separate wide-character codes, but shall be grouped with an  
 24190 adjacent character. Changing the *LC\_CTYPE* category causes the shift state of this function to be  
 24191 unspecified.

24192 **RETURN VALUE**

24193       If *s* is a null pointer, *mblen()* shall return a non-zero or 0 value, if character encodings,  
 24194 respectively, do or do not have state-dependent encodings. If *s* is not a null pointer, *mblen()* shall  
 24195 either return 0 (if *s* points to the null byte), or return the number of bytes that constitute the  
 24196 character (if the next *n* or fewer bytes form a valid character), or return -1 (if they do not form a  
 24197 valid character) and may set *errno* to indicate the error. In no case shall the value returned be  
 24198 greater than *n* or the value of the {MB\_CUR\_MAX} macro.

24199 **ERRORS**24200       The *mblen()* function may fail if:24201 **XSI**       [EILSEQ]       Invalid character sequence is detected.24202 **EXAMPLES**

24203       None.

24204 **APPLICATION USAGE**

24205       None.

24206 **RATIONALE**

24207       None.

24208 **FUTURE DIRECTIONS**

24209       None.

24210 **SEE ALSO**

24211 *mbtowc()*, *mbstowcs()*, *wctomb()*, *wcstombs()*, the Base Definitions volume of  
24212 IEEE Std 1003.1-200x, <stdlib.h>

24213 **CHANGE HISTORY**

24214 First released in Issue 4. Aligned with the ISO C standard.

24215 **NAME**24216 `mbrlen` — get number of bytes in a character (restartable)24217 **SYNOPSIS**24218 `#include <wchar.h>`24219 `size_t mbrlen(const char *restrict s, size_t n,`  
24220 `mbstate_t *restrict ps);`24221 **DESCRIPTION**24222 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
24223 conflict between the requirements described here and the ISO C standard is unintentional. This  
24224 volume of IEEE Std 1003.1-200x defers to the ISO C standard.24225 If *s* is not a null pointer, *mbrlen()* shall determine the number of bytes constituting the character  
24226 pointed to by *s*. It shall be equivalent to:24227 `mbstate_t internal;`24228 `mbrtowc(NULL, s, n, ps != NULL ? ps : &internal);`24229 If *ps* is a null pointer, the *mbrlen()* function shall use its own internal **mbstate\_t** object, which is  
24230 initialized at program start-up to the initial conversion state. Otherwise, the **mbstate\_t** object  
24231 pointed to by *ps* shall be used to completely describe the current conversion state of the  
24232 associated character sequence. The implementation shall behave as if no function defined in this  
24233 volume of IEEE Std 1003.1-200x calls *mbrlen()*.24234 The behavior of this function is affected by the *LC\_CTYPE* category of the current locale.24235 **RETURN VALUE**24236 The *mbrlen()* function shall return the first of the following that applies:24237 **0** If the next *n* or fewer bytes complete the character that corresponds to the null  
24238 wide character.24239 **positive** If the next *n* or fewer bytes complete a valid character; the value returned shall  
24240 be the number of bytes that complete the character.24241 **(size\_t)-2** If the next *n* bytes contribute to an incomplete but potentially valid character,  
24242 and all *n* bytes have been processed. When *n* has at least the value of the  
24243 {MB\_CUR\_MAX} macro, this case can only occur if *s* points at a sequence of  
24244 redundant shift sequences (for implementations with state-dependent  
24245 encodings).24246 **(size\_t)-1** If an encoding error occurs, in which case the next *n* or fewer bytes do not  
24247 contribute to a complete and valid character. In this case, [EILSEQ] shall be  
24248 stored in *errno* and the conversion state is undefined.24249 **ERRORS**24250 The *mbrlen()* function may fail if:24251 [EINVAL] *ps* points to an object that contains an invalid conversion state.

24252 [EILSEQ] Invalid character sequence is detected.

24253 **EXAMPLES**

24254 None.

24255 **APPLICATION USAGE**

24256 None.

24257 **RATIONALE**

24258 None.

24259 **FUTURE DIRECTIONS**

24260 None.

24261 **SEE ALSO**24262 *mbsinit()*, *mbrtowc()*, the Base Definitions volume of IEEE Std 1003.1-200x, <**wchar.h**>24263 **CHANGE HISTORY**

24264 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995

24265 (E).

24266 **Issue 6**24267 The *mbrlen()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

## 24268 NAME

24269 mbrtowc — convert a character to a wide-character code (restartable)

## 24270 SYNOPSIS

24271 #include &lt;wchar.h&gt;

24272 size\_t mbrtowc(wchar\_t \*restrict pwc, const char \*restrict s,  
24273 size\_t n, mbstate\_t \*restrict ps);

## 24274 DESCRIPTION

24275 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
24276 conflict between the requirements described here and the ISO C standard is unintentional. This  
24277 volume of IEEE Std 1003.1-200x defers to the ISO C standard.24278 If *s* is a null pointer, the *mbrtowc()* function shall be equivalent to the call:

24279 mbrtowc(NULL, "", 1, ps)

24280 In this case, the values of the arguments *pwc* and *n* are ignored.24281 If *s* is not a null pointer, the *mbrtowc()* function shall inspect at most *n* bytes beginning at the  
24282 byte pointed to by *s* to determine the number of bytes needed to complete the next character  
24283 (including any shift sequences). If the function determines that the next character is completed, it  
24284 shall determine the value of the corresponding wide character and then, if *pwc* is not a null  
24285 pointer, shall store that value in the object pointed to by *pwc*. If the corresponding wide  
24286 character is the null wide character, the resulting state described shall be the initial conversion  
24287 state.24288 If *ps* is a null pointer, the *mbrtowc()* function shall use its own internal **mbstate\_t** object, which  
24289 shall be initialized at program start-up to the initial conversion state. Otherwise, the **mbstate\_t**  
24290 object pointed to by *ps* shall be used to completely describe the current conversion state of the  
24291 associated character sequence. The implementation shall behave as if no function defined in this  
24292 volume of IEEE Std 1003.1-200x calls *mbrtowc()*.24293 The behavior of this function is affected by the *LC\_CTYPE* category of the current locale.

## 24294 RETURN VALUE

24295 The *mbrtowc()* function shall return the first of the following that applies:24296 0 If the next *n* or fewer bytes complete the character that corresponds to the null  
24297 wide character (which is the value stored).24298 between 1 and *n* inclusive24299 If the next *n* or fewer bytes complete a valid character (which is the value  
24300 stored); the value returned shall be the number of bytes that complete the  
24301 character.24302 (**size\_t**)−2 If the next *n* bytes contribute to an incomplete but potentially valid character,  
24303 and all *n* bytes have been processed (no value is stored). When *n* has at least  
24304 the value of the {**MB\_CUR\_MAX**} macro, this case can only occur if *s* points at  
24305 a sequence of redundant shift sequences (for implementations with state-  
24306 dependent encodings).24307 (**size\_t**)−1 If an encoding error occurs, in which case the next *n* or fewer bytes do not  
24308 contribute to a complete and valid character (no value is stored). In this case,  
24309 [EILSEQ] shall be stored in *errno* and the conversion state is undefined.

24310 **ERRORS**

24311 The *mbrtowc()* function may fail if:

24312 CX [EINVAL] *ps* points to an object that contains an invalid conversion state.

24313 [EILSEQ] Invalid character sequence is detected.

24314 **EXAMPLES**

24315 None.

24316 **APPLICATION USAGE**

24317 None.

24318 **RATIONALE**

24319 None.

24320 **FUTURE DIRECTIONS**

24321 None.

24322 **SEE ALSO**

24323 *mbstowc()*, the Base Definitions volume of IEEE Std 1003.1-200x, <*wchar.h*>

24324 **CHANGE HISTORY**

24325 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995  
24326 (E).

24327 **Issue 6**

24328 The *mbrtowc()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard. |

24329 The following new requirements on POSIX implementations derive from alignment with the |  
24330 Single UNIX Specification: |

24331 • The [EINVAL] error condition is added. |

24332 ISO/IEC 9899:1999 standard, Technical Corrigendum No. 1 is incorporated. |

24333 **NAME**

24334 mbsinit — determine conversion object status

24335 **SYNOPSIS**

24336 #include &lt;wchar.h&gt;

24337 int mbsinit(const mbstate\_t \*ps);

24338 **DESCRIPTION**

24339 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
 24340 conflict between the requirements described here and the ISO C standard is unintentional. This  
 24341 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

24342 If *ps* is not a null pointer, the *mbsinit()* function shall determine whether the object pointed to by  
 24343 *ps* describes an initial conversion state.

24344 **RETURN VALUE**

24345 The *mbsinit()* function shall return non-zero if *ps* is a null pointer, or if the pointed-to object  
 24346 describes an initial conversion state; otherwise, it shall return zero.

24347 If an **mbstate\_t** object is altered by any of the functions described as “restartable”, and is then  
 24348 used with a different character sequence, or in the other conversion direction, or with a different  
 24349 *LC\_CTYPE* category setting than on earlier function calls, the behavior is undefined.

24350 **ERRORS**

24351 No errors are defined.

24352 **EXAMPLES**

24353 None.

24354 **APPLICATION USAGE**

24355 The **mbstate\_t** object is used to describe the current conversion state from a particular character  
 24356 sequence to a wide-character sequence (or *vice versa*) under the rules of a particular setting of the  
 24357 *LC\_CTYPE* category of the current locale.

24358 The initial conversion state corresponds, for a conversion in either direction, to the beginning of  
 24359 a new character sequence in the initial shift state. A zero valued **mbstate\_t** object is at least one  
 24360 way to describe an initial conversion state. A zero valued **mbstate\_t** object can be used to initiate  
 24361 conversion involving any character sequence, in any *LC\_CTYPE* category setting.

24362 **RATIONALE**

24363 None.

24364 **FUTURE DIRECTIONS**

24365 None.

24366 **SEE ALSO**

24367 *mbrlen()*, *mbrtowc()*, *wcrtomb()*, *mbsrtowcs()*, *wcsrtombs()*, the Base Definitions volume of  
 24368 IEEE Std 1003.1-200x, <wchar.h>

24369 **CHANGE HISTORY**

24370 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995  
 24371 (E).

## 24372 NAME

24373 mbsrtowcs — convert a character string to a wide-character string (restartable)

## 24374 SYNOPSIS

24375 #include <wchar.h>

24376 size\_t mbsrtowcs(wchar\_t \*restrict dst, const char \*\*restrict src,  
24377 size\_t len, mbstate\_t \*restrict ps);

## 24378 DESCRIPTION

24379 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
24380 conflict between the requirements described here and the ISO C standard is unintentional. This  
24381 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

24382 The *mbsrtowcs()* function shall convert a sequence of characters, beginning in the conversion  
24383 state described by the object pointed to by *ps*, from the array indirectly pointed to by *src* into a  
24384 sequence of corresponding wide characters. If *dst* is not a null pointer, the converted characters  
24385 shall be stored into the array pointed to by *dst*. Conversion continues up to and including a  
24386 terminating null character, which shall also be stored. Conversion shall stop early in either of the  
24387 following cases:

- 24388 • A sequence of bytes is encountered that does not form a valid character.
- 24389 • *len* codes have been stored into the array pointed to by *dst* (and *dst* is not a null pointer).

24390 Each conversion shall take place as if by a call to the *mbrtowc()* function.

24391 If *dst* is not a null pointer, the pointer object pointed to by *src* shall be assigned either a null  
24392 pointer (if conversion stopped due to reaching a terminating null character) or the address just  
24393 past the last character converted (if any). If conversion stopped due to reaching a terminating  
24394 null character, and if *dst* is not a null pointer, the resulting state described shall be the initial  
24395 conversion state.

24396 If *ps* is a null pointer, the *mbsrtowcs()* function shall use its own internal **mbstate\_t** object, which  
24397 is initialized at program start-up to the initial conversion state. Otherwise, the **mbstate\_t** object  
24398 pointed to by *ps* shall be used to completely describe the current conversion state of the  
24399 associated character sequence. The implementation behaves as if no function defined in this  
24400 volume of IEEE Std 1003.1-200x calls *mbsrtowcs()*.

24401 The behavior of this function shall be affected by the *LC\_CTYPE* category of the current locale.

## 24402 RETURN VALUE

24403 If the input conversion encounters a sequence of bytes that do not form a valid character, an  
24404 encoding error occurs. In this case, the *mbsrtowcs()* function stores the value of the macro  
24405 [EILSEQ] in *errno* and shall return (**size\_t**)-1; the conversion state is undefined. Otherwise, it  
24406 shall return the number of characters successfully converted, not including the terminating null  
24407 (if any).

## 24408 ERRORS

24409 The *mbsrtowcs()* function may fail if:

- 24410 CX [EINVAL] *ps* points to an object that contains an invalid conversion state.
- 24411 [EILSEQ] Invalid character sequence is detected.

24412 **EXAMPLES**

24413 None.

24414 **APPLICATION USAGE**

24415 None.

24416 **RATIONALE**

24417 None.

24418 **FUTURE DIRECTIONS**

24419 None.

24420 **SEE ALSO**24421 *mbstowcs()*, *mbstowc()*, the Base Definitions volume of IEEE Std 1003.1-200x, <**wchar.h**>24422 **CHANGE HISTORY**24423 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995  
24424 (E).24425 **Issue 6**24426 The *mbsrtowcs()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard. |

24427 The [EINVAL] error condition is marked CX. |

24428 **NAME**

24429 mbstowcs — convert a character string to a wide-character string

24430 **SYNOPSIS**

24431 #include <stdlib.h>

24432 size\_t mbstowcs(wchar\_t \*restrict pwcs, const char \*restrict s,  
24433 size\_t n);

24434 **DESCRIPTION**

24435 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
24436 conflict between the requirements described here and the ISO C standard is unintentional. This  
24437 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

24438 The *mbstowcs()* function shall convert a sequence of characters that begins in the initial shift  
24439 state from the array pointed to by *s* into a sequence of corresponding wide-character codes and  
24440 shall store not more than *n* wide-character codes into the array pointed to by *pwcs*. No  
24441 characters that follow a null byte (which is converted into a wide-character code with value 0)  
24442 shall be examined or converted. Each character shall be converted as if by a call to *mbtowc()*,  
24443 except that the shift state of *mbtowc()* is not affected.

24444 No more than *n* elements shall be modified in the array pointed to by *pwcs*. If copying takes  
24445 place between objects that overlap, the behavior is undefined.

24446 **XSI** The behavior of this function shall be affected by the *LC\_CTYPE* category of the current locale. If  
24447 *pwcs* is a null pointer, *mbstowcs()* shall return the length required to convert the entire array  
24448 regardless of the value of *n*, but no values are stored.

24449 **RETURN VALUE**

24450 **CX** If an invalid character is encountered, *mbstowcs()* shall return **(size\_t)-1** and may set *errno* to  
24451 **XSI** indicate the error. Otherwise, *mbstowcs()* shall return the number of the array elements modified  
24452 (or required if *pwcs* is null), not including a terminating 0 code, if any. The array shall not be  
24453 zero-terminated if the value returned is *n*.

24454 **ERRORS**

24455 The *mbstowcs()* function may fail if:

24456 **XSI** [EILSEQ] Invalid byte sequence is detected.

24457 **EXAMPLES**

24458 None.

24459 **APPLICATION USAGE**

24460 None.

24461 **RATIONALE**

24462 None.

24463 **FUTURE DIRECTIONS**

24464 None.

24465 **SEE ALSO**

24466 *mblen()*, *mbtowc()*, *wctomb()*, *wcstombs()*, the Base Definitions volume of IEEE Std 1003.1-200x,  
24467 <stdlib.h>

24468 **CHANGE HISTORY**

24469 First released in Issue 4. Aligned with the ISO C standard.

24470 **Issue 6**

24471 The *mbstowcs()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard. |

24472 Extensions beyond the ISO C standard are now marked. |

24473 **NAME**

24474 mbtowc — convert a character to a wide-character code

24475 **SYNOPSIS**

24476 #include &lt;stdlib.h&gt;

24477 int mbtowc(wchar\_t \*restrict *pwc*, const char \*restrict *s*, size\_t *n*);24478 **DESCRIPTION**

24479 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 24480 conflict between the requirements described here and the ISO C standard is unintentional. This  
 24481 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

24482 If *s* is not a null pointer, *mbtowc()* shall determine the number of the bytes that constitute the  
 24483 character pointed to by *s*. It shall then determine the wide-character code for the value of type  
 24484 **wchar\_t** that corresponds to that character. (The value of the wide-character code corresponding  
 24485 to the null byte is 0.) If the character is valid and *pwc* is not a null pointer, *mbtowc()* shall store  
 24486 the wide-character code in the object pointed to by *pwc*.

24487 The behavior of this function is affected by the *LC\_CTYPE* category of the current locale. For a  
 24488 state-dependent encoding, this function is placed into its initial state by a call for which its  
 24489 character pointer argument, *s*, is a null pointer. Subsequent calls with *s* as other than a null  
 24490 pointer shall cause the internal state of the function to be altered as necessary. A call with *s* as a  
 24491 null pointer shall cause this function to return a non-zero value if encodings have state  
 24492 dependency, and 0 otherwise. If the implementation employs special bytes to change the shift  
 24493 state, these bytes shall not produce separate wide-character codes, but shall be grouped with an  
 24494 adjacent character. Changing the *LC\_CTYPE* category causes the shift state of this function to be  
 24495 unspecified. At most *n* bytes of the array pointed to by *s* shall be examined.

24496 The implementation shall behave as if no function defined in this volume of  
 24497 IEEE Std 1003.1-200x calls *mbtowc()*.

24498 **RETURN VALUE**

24499 If *s* is a null pointer, *mbtowc()* shall return a non-zero or 0 value, if character encodings,  
 24500 respectively, do or do not have state-dependent encodings. If *s* is not a null pointer, *mbtowc()*  
 24501 shall either return 0 (if *s* points to the null byte), or return the number of bytes that constitute the  
 24502 **CX** converted character (if the next *n* or fewer bytes form a valid character), or return -1 and may  
 24503 set *errno* to indicate the error (if they do not form a valid character).

24504 In no case shall the value returned be greater than *n* or the value of the {MB\_CUR\_MAX} macro.

24505 **ERRORS**

24506 The *mbtowc()* function may fail if:

24507 **XSI** [EILSEQ] Invalid character sequence is detected.

24508 **EXAMPLES**

24509 None.

24510 **APPLICATION USAGE**

24511 None.

24512 **RATIONALE**

24513 None.

24514 **FUTURE DIRECTIONS**

24515 None.

24516 **SEE ALSO**

24517            *mblen()*, *mbstowcs()*, *wctomb()*, *wcstombs()*, the Base Definitions volume of IEEE Std 1003.1-200x,  
24518            <**stdlib.h**>

24519 **CHANGE HISTORY**

24520            First released in Issue 4. Aligned with the ISO C standard.

24521 **Issue 6**

24522            The *mbtowc()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard. |

24523            Extensions beyond the ISO C standard are now marked. |

24524 **NAME**

24525 memccpy — copy bytes in memory

24526 **SYNOPSIS**

24527 XSI #include &lt;string.h&gt;

24528 void \*memccpy(void \*restrict *s1*, const void \*restrict *s2*,  
24529 int *c*, size\_t *n*);

24530

24531 **DESCRIPTION**

24532 The *memccpy()* function shall copy bytes from memory area *s2* into *s1*, stopping after the first  
24533 occurrence of byte *c* (converted to an **unsigned char**) is copied, or after *n* bytes are copied,  
24534 whichever comes first. If copying takes place between objects that overlap, the behavior is  
24535 undefined.

24536 **RETURN VALUE**

24537 The *memccpy()* function shall return a pointer to the byte after the copy of *c* in *s1*, or a null  
24538 pointer if *c* was not found in the first *n* bytes of *s2*.

24539 **ERRORS**

24540 No errors are defined.

24541 **EXAMPLES**

24542 None.

24543 **APPLICATION USAGE**24544 The *memccpy()* function does not check for the overflow of the receiving memory area.24545 **RATIONALE**

24546 None.

24547 **FUTURE DIRECTIONS**

24548 None.

24549 **SEE ALSO**

24550 The Base Definitions volume of IEEE Std 1003.1-200x, &lt;string.h&gt;

24551 **CHANGE HISTORY**

24552 First released in Issue 1. Derived from Issue 1 of the SVID.

24553 **Issue 6**

24554 The **restrict** keyword is added to the *memccpy()* prototype for alignment with the  
24555 ISO/IEC 9899:1999 standard.

24556 **NAME**

24557 memchr — find byte in memory

24558 **SYNOPSIS**

24559 #include &lt;string.h&gt;

24560 void \*memchr(const void \*s, int c, size\_t n);

24561 **DESCRIPTION**

24562 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
24563 conflict between the requirements described here and the ISO C standard is unintentional. This  
24564 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

24565 The *memchr()* function shall locate the first occurrence of *c* (converted to an **unsigned char**) in  
24566 the initial *n* bytes (each interpreted as **unsigned char**) of the object pointed to by *s*.

24567 **RETURN VALUE**

24568 The *memchr()* function shall return a pointer to the located byte, or a null pointer if the byte does  
24569 not occur in the object.

24570 **ERRORS**

24571 No errors are defined.

24572 **EXAMPLES**

24573 None.

24574 **APPLICATION USAGE**

24575 None.

24576 **RATIONALE**

24577 None.

24578 **FUTURE DIRECTIONS**

24579 None.

24580 **SEE ALSO**24581 The Base Definitions volume of IEEE Std 1003.1-200x, <**string.h**>24582 **CHANGE HISTORY**

24583 First released in Issue 1. Derived from Issue 1 of the SVID.

24584 **NAME**

24585        memcmp — compare bytes in memory

24586 **SYNOPSIS**

24587        #include &lt;string.h&gt;

24588        int memcmp(const void \*s1, const void \*s2, size\_t n);

24589 **DESCRIPTION**

24590 **CX**        The functionality described on this reference page is aligned with the ISO C standard. Any  
24591 conflict between the requirements described here and the ISO C standard is unintentional. This  
24592 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

24593        The *memcmp()* function shall compare the first *n* bytes (each interpreted as **unsigned char**) of the  
24594 object pointed to by *s1* to the first *n* bytes of the object pointed to by *s2*.

24595        The sign of a non-zero return value shall be determined by the sign of the difference between the  
24596 values of the first pair of bytes (both interpreted as type **unsigned char**) that differ in the objects  
24597 being compared.

24598 **RETURN VALUE**

24599        The *memcmp()* function shall return an integer greater than, equal to, or less than 0, if the object  
24600 pointed to by *s1* is greater than, equal to, or less than the object pointed to by *s2*, respectively.

24601 **ERRORS**

24602        No errors are defined.

24603 **EXAMPLES**

24604        None.

24605 **APPLICATION USAGE**

24606        None.

24607 **RATIONALE**

24608        None.

24609 **FUTURE DIRECTIONS**

24610        None.

24611 **SEE ALSO**24612        The Base Definitions volume of IEEE Std 1003.1-200x, <**string.h**>24613 **CHANGE HISTORY**

24614        First released in Issue 1. Derived from Issue 1 of the SVID.

24615 **NAME**

24616           memcpy — copy bytes in memory

24617 **SYNOPSIS**

24618           #include <string.h>

24619           void \*memcpy(void \*restrict *s1*, const void \*restrict *s2*, size\_t *n*);

24620 **DESCRIPTION**

24621 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
24622       conflict between the requirements described here and the ISO C standard is unintentional. This  
24623       volume of IEEE Std 1003.1-200x defers to the ISO C standard.

24624       The *memcpy()* function shall copy *n* bytes from the object pointed to by *s2* into the object pointed  
24625       to by *s1*. If copying takes place between objects that overlap, the behavior is undefined.

24626 **RETURN VALUE**

24627       The *memcpy()* function shall return *s1*; no return value is reserved to indicate an error.

24628 **ERRORS**

24629       No errors are defined.

24630 **EXAMPLES**

24631       None.

24632 **APPLICATION USAGE**

24633       The *memcpy()* function does not check for the overflowing of the receiving memory area.

24634 **RATIONALE**

24635       None.

24636 **FUTURE DIRECTIONS**

24637       None.

24638 **SEE ALSO**

24639       The Base Definitions volume of IEEE Std 1003.1-200x, <**string.h**>

24640 **CHANGE HISTORY**

24641       First released in Issue 1. Derived from Issue 1 of the SVID.

24642 **Issue 6**

24643       The *memcpy()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

24644 **NAME**

24645 memmove — copy bytes in memory with overlapping areas

24646 **SYNOPSIS**

24647 #include <string.h>

24648 void \*memmove(void \*s1, const void \*s2, size\_t n);

24649 **DESCRIPTION**

24650 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
24651 conflict between the requirements described here and the ISO C standard is unintentional. This  
24652 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

24653 The *memmove()* function shall copy *n* bytes from the object pointed to by *s2* into the object  
24654 pointed to by *s1*. Copying takes place as if the *n* bytes from the object pointed to by *s2* are first  
24655 copied into a temporary array of *n* bytes that does not overlap the objects pointed to by *s1* and  
24656 *s2*, and then the *n* bytes from the temporary array are copied into the object pointed to by *s1*.

24657 **RETURN VALUE**

24658 The *memmove()* function shall return *s1*; no return value is reserved to indicate an error.

24659 **ERRORS**

24660 No errors are defined.

24661 **EXAMPLES**

24662 None.

24663 **APPLICATION USAGE**

24664 None.

24665 **RATIONALE**

24666 None.

24667 **FUTURE DIRECTIONS**

24668 None.

24669 **SEE ALSO**

24670 The Base Definitions volume of IEEE Std 1003.1-200x, <**string.h**>

24671 **CHANGE HISTORY**

24672 First released in Issue 4. Derived from the ANSI C standard.

24673 **NAME**

24674       memset — set bytes in memory

24675 **SYNOPSIS**

24676       #include &lt;string.h&gt;

24677       void \*memset(void \*s, int c, size\_t n);

24678 **DESCRIPTION**

24679 cx       The functionality described on this reference page is aligned with the ISO C standard. Any  
24680       conflict between the requirements described here and the ISO C standard is unintentional. This  
24681       volume of IEEE Std 1003.1-200x defers to the ISO C standard.

24682       The *memset()* function shall copy *c* (converted to an **unsigned char**) into each of the first *n* bytes  
24683       of the object pointed to by *s*.

24684 **RETURN VALUE**24685       The *memset()* function shall return *s*; no return value is reserved to indicate an error.24686 **ERRORS**

24687       No errors are defined.

24688 **EXAMPLES**

24689       None.

24690 **APPLICATION USAGE**

24691       None.

24692 **RATIONALE**

24693       None.

24694 **FUTURE DIRECTIONS**

24695       None.

24696 **SEE ALSO**24697       The Base Definitions volume of IEEE Std 1003.1-200x, <**string.h**>24698 **CHANGE HISTORY**

24699       First released in Issue 1. Derived from Issue 1 of the SVID.

24700 **NAME**

24701 mkdir — make a directory

24702 **SYNOPSIS**

24703 #include <sys/stat.h>

24704 int mkdir(const char \*path, mode\_t mode);

24705 **DESCRIPTION**

24706 The *mkdir()* function shall create a new directory with name *path*. The file permission bits of the  
 24707 new directory shall be initialized from *mode*. These file permission bits of the *mode* argument  
 24708 shall be modified by the process' file creation mask.

24709 When bits in *mode* other than the file permission bits are set, the meaning of these additional bits  
 24710 is implementation-defined.

24711 The directory's user ID shall be set to the process' effective user ID. The directory's group ID  
 24712 shall be set to the group ID of the parent directory or to the effective group ID of the process.  
 24713 Implementations shall provide a way to initialize the directory's group ID to the group ID of the  
 24714 parent directory. Implementations may, but need not, provide an implementation-defined way  
 24715 to initialize the directory's group ID to the effective group ID of the calling process.

24716 The newly created directory shall be an empty directory.

24717 If *path* names a symbolic link, *mkdir()* shall fail and set *errno* to [EEXIST].

24718 Upon successful completion, *mkdir()* shall mark for update the *st\_atime*, *st\_ctime*, and *st\_mtime*  
 24719 fields of the directory. Also, the *st\_ctime* and *st\_mtime* fields of the directory that contains the  
 24720 new entry shall be marked for update.

24721 **RETURN VALUE**

24722 Upon successful completion, *mkdir()* shall return 0. Otherwise, -1 shall be returned, no directory  
 24723 shall be created, and *errno* shall be set to indicate the error.

24724 **ERRORS**

24725 The *mkdir()* function shall fail if:

24726 [EACCES] Search permission is denied on a component of the path prefix, or write  
 24727 permission is denied on the parent directory of the directory to be created.

24728 [EEXIST] The named file exists.

24729 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*  
 24730 argument.

24731 [EMLINK] The link count of the parent directory would exceed {LINK\_MAX}.

24732 [ENAMETOOLONG]

24733 The length of the *path* argument exceeds {PATH\_MAX} or a pathname  
 24734 component is longer than {NAME\_MAX}.

24735 [ENOENT] A component of the path prefix specified by *path* does not name an existing  
 24736 directory or *path* is an empty string.

24737 [ENOSPC] The file system does not contain enough space to hold the contents of the new  
 24738 directory or to extend the parent directory of the new directory.

24739 [ENOTDIR] A component of the path prefix is not a directory.

24740 [EROFS] The parent directory resides on a read-only file system.

24741 The *mkdir()* function may fail if:

24742 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
24743 resolution of the *path* argument.

24744 [ENAMETOOLONG]

24745 As a result of encountering a symbolic link in resolution of the *path* argument, |  
24746 the length of the substituted pathname string exceeded {PATH\_MAX}. |

#### 24747 EXAMPLES

##### 24748 **Creating a Directory**

24749 The following example shows how to create a directory named */home/cnd/mod1*, with  
24750 read/write/search permissions for owner and group, and with read/search permissions for  
24751 others.

```
24752 #include <sys/types.h>
```

```
24753 #include <sys/stat.h>
```

```
24754 int status;
```

```
24755 ...
```

```
24756 status = mkdir("/home/cnd/mod1", S_IRWXU | S_IRWXG | S_IROTH | S_IXOTH);
```

#### 24757 APPLICATION USAGE

24758 None.

#### 24759 RATIONALE

24760 The *mkdir()* function originated in 4.2 BSD and was added to System V in Release 3.0.

24761 4.3 BSD detects [ENAMETOOLONG].

24762 The POSIX.1-1990 standard required that the group ID of a newly created directory be set to the |  
24763 group ID of its parent directory or to the effective group ID of the creating process. FIPS 151-2 |  
24764 required that implementations provide a way to have the group ID be set to the group ID of the |  
24765 containing directory, but did not prohibit implementations also supporting a way to set the |  
24766 group ID to the effective group ID of the creating process. Conforming applications should not |  
24767 assume which group ID will be used. If it matters, an application can use *chown()* to set the |  
24768 group ID after the directory is created, or determine under what conditions the implementation |  
24769 will set the desired group ID. |

#### 24770 FUTURE DIRECTIONS

24771 None.

#### 24772 SEE ALSO

24773 *umask()*, the Base Definitions volume of IEEE Std 1003.1-200x, *<sys/stat.h>*, *<sys/types.h>*

#### 24774 CHANGE HISTORY

24775 First released in Issue 3.

24776 Entry included for alignment with the POSIX.1-1988 standard.

#### 24777 Issue 6

24778 In the SYNOPSIS, the optional include of the *<sys/types.h>* header is removed.

24779 The following new requirements on POSIX implementations derive from alignment with the |  
24780 Single UNIX Specification:

- 24781 • The requirement to include *<sys/types.h>* has been removed. Although *<sys/types.h>* was  
24782 required for conforming implementations of previous POSIX specifications, it was not  
24783 required for UNIX applications.

24784

- The [ELOOP] mandatory error condition is added.

24785

- A second [ENAMETOOLONG] is added as an optional error condition.

24786

The following changes were made to align with the IEEE P1003.1a draft standard:

24787

- The [ELOOP] optional error condition is added.

24788 **NAME**24789 `mkfifo` — make a FIFO special file24790 **SYNOPSIS**24791 `#include <sys/stat.h>`24792 `int mkfifo(const char *path, mode_t mode);`24793 **DESCRIPTION**

24794 The `mkfifo()` function shall create a new FIFO special file named by the pathname pointed to by  
 24795 `path`. The file permission bits of the new FIFO shall be initialized from `mode`. The file permission  
 24796 bits of the `mode` argument shall be modified by the process' file creation mask.

24797 When bits in `mode` other than the file permission bits are set, the effect is implementation-  
 24798 defined.

24799 If `path` names a symbolic link, `mkfifo()` shall fail and set `errno` to [EEXIST].

24800 The FIFO's user ID shall be set to the process' effective user ID. The FIFO's group ID shall be set  
 24801 to the group ID of the parent directory or to the effective group ID of the process.  
 24802 Implementation shall provide a way to initialize the FIFO's group ID to the group ID of the  
 24803 parent directory. Implementations may, but need not, provide an implementation-defined way  
 24804 to initialize the FIFO's group ID to the effective group ID of the calling process.

24805 Upon successful completion, `mkfifo()` shall mark for update the `st_atime`, `st_ctime`, and `st_mtime`  
 24806 fields of the file. Also, the `st_ctime` and `st_mtime` fields of the directory that contains the new  
 24807 entry shall be marked for update.

24808 **RETURN VALUE**

24809 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned, no FIFO shall  
 24810 be created, and `errno` shall be set to indicate the error.

24811 **ERRORS**

24812 The `mkfifo()` function shall fail if:

24813 [EACCES] A component of the path prefix denies search permission, or write permission  
 24814 is denied on the parent directory of the FIFO to be created.

24815 [EEXIST] The named file already exists.

24816 [ELOOP] A loop exists in symbolic links encountered during resolution of the `path`  
 24817 argument.

24818 [ENAMETOOLONG]

24819 The length of the `path` argument exceeds {PATH\_MAX} or a pathname  
 24820 component is longer than {NAME\_MAX}.

24821 [ENOENT] A component of the path prefix specified by `path` does not name an existing  
 24822 directory or `path` is an empty string.

24823 [ENOSPC] The directory that would contain the new file cannot be extended or the file  
 24824 system is out of file-allocation resources.

24825 [ENOTDIR] A component of the path prefix is not a directory.

24826 [EROFS] The named file resides on a read-only file system.

24827 The `mkfifo()` function may fail if:

24828 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
 24829 resolution of the `path` argument.

24830 [ENAMETOOLONG]  
 24831 As a result of encountering a symbolic link in resolution of the *path* argument, |  
 24832 the length of the substituted pathname string exceeded {PATH\_MAX}. |

#### 24833 EXAMPLES

##### 24834 Creating a FIFO File

24835 The following example shows how to create a FIFO file named `/home/cnd/mod_done`, with  
 24836 read/write permissions for owner, and with read permissions for group and others.

```
24837 #include <sys/types.h>
24838 #include <sys/stat.h>
24839
24840 int status;
24841 ...
24842 status = mkfifo("/home/cnd/mod_done", S_IWUSR | S_IRUSR |
24843               S_IRGRP | S_IROTH);
```

#### 24843 APPLICATION USAGE

24844 None.

#### 24845 RATIONALE

24846 The syntax of this function is intended to maintain compatibility with historical  
 24847 implementations of *mknod()*. The latter function was included in the 1984 `/usr/group` standard  
 24848 but only for use in creating FIFO special files. The *mknod()* function was originally excluded  
 24849 from the POSIX.1-1988 standard as implementation-defined and replaced by *mkdir()* and  
 24850 *mkfifo()*. The *mknod()* function is now included for alignment with the Single UNIX  
 24851 Specification.

24852 The POSIX.1-1990 standard required that the group ID of a newly created FIFO be set to the |  
 24853 group ID of its parent directory or to the effective group ID of the creating process. FIPS 151-2 |  
 24854 required that implementations provide a way to have the group ID be set to the group ID of the |  
 24855 containing directory, but did not prohibit implementations also supporting a way to set the |  
 24856 group ID to the effective group ID of the creating process. Conforming applications should not |  
 24857 assume which group ID will be used. If it matters, an application can use *chown()* to set the |  
 24858 group ID after the FIFO is created, or determine under what conditions the implementation will |  
 24859 set the desired group ID. |

#### 24860 FUTURE DIRECTIONS

24861 None.

#### 24862 SEE ALSO

24863 *umask()*, the Base Definitions volume of IEEE Std 1003.1-200x, `<sys/stat.h>`, `<sys/types.h>`

#### 24864 CHANGE HISTORY

24865 First released in Issue 3.

24866 Entry included for alignment with the POSIX.1-1988 standard.

#### 24867 Issue 6

24868 In the SYNOPSIS, the optional include of the `<sys/types.h>` header is removed.

24869 The following new requirements on POSIX implementations derive from alignment with the |  
 24870 Single UNIX Specification:

- 24871 • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was  
 24872 required for conforming implementations of previous POSIX specifications, it was not  
 24873 required for UNIX applications.

- 24874           • The [ELOOP] mandatory error condition is added.
- 24875           • A second [ENAMETOOLONG] is added as an optional error condition.
- 24876       The following changes were made to align with the IEEE P1003.1a draft standard:
- 24877           • The [ELOOP] optional error condition is added.

24878 **NAME**

24879 `mknod` — make a directory, a special or regular file

24880 **SYNOPSIS**

24881 xSI `#include <sys/stat.h>`

24882 `int mknod(const char *path, mode_t mode, dev_t dev);`

24883

24884 **DESCRIPTION**

24885 The `mknod()` function shall create a new file named by the pathname to which the argument `path` |  
 24886 points.

24887 The file type for `path` is OR'ed into the `mode` argument, and the application shall select one of the  
 24888 following symbolic constants:

24889

24890

Name	Description
S_IFIFO	FIFO-special
S_IFCHR	Character-special (non-portable)
S_IFDIR	Directory (non-portable)
S_IFBLK	Block-special (non-portable)
S_IFREG	Regular (non-portable)

24891

24892

24893

24894

24895

24896 The only portable use of `mknod()` is to create a FIFO-special file. If `mode` is not S\_IFIFO or `dev` is  
 24897 not 0, the behavior of `mknod()` is unspecified.

24898 The permissions for the new file are OR'ed into the `mode` argument, and may be selected from  
 24899 any combination of the following symbolic constants:

24900

24901

Name	Description
S_ISUID	Set user ID on execution.
S_ISGID	Set group ID on execution.
S_IRWXU	Read, write, or execute (search) by owner.
S_IRUSR	Read by owner.
S_IWUSR	Write by owner.
S_IXUSR	Execute (search) by owner.
S_IRWXG	Read, write, or execute (search) by group.
S_IRGRP	Read by group.
S_IWGRP	Write by group.
S_IXGRP	Execute (search) by group.
S_IRWXO	Read, write, or execute (search) by others.
S_IROTH	Read by others.
S_IWOTH	Write by others.
S_IXOTH	Execute (search) by others.
S_ISVTX	On directories, restricted deletion flag.

24902

24903

24904

24905

24906

24907

24908

24909

24910

24911

24912

24913

24914

24915

24916

24917 The user ID of the file shall be initialized to the effective user ID of the process. The group ID of |  
 24918 the file shall be initialized to either the effective group ID of the process or the group ID of the |  
 24919 parent directory. Implementations shall provide a way to initialize the file's group ID to the |  
 24920 group ID of the parent directory. Implementations may, but need not, provide an |  
 24921 implementation-defined way to initialize the file's gorup ID to the effective group ID of the |  
 24922 calling proces. |

- 24923 The owner, group, and other permission bits of *mode* shall be modified by the file mode creation  
 24924 mask of the process. The *mknod()* function shall clear each bit whose corresponding bit in the file  
 24925 mode creation mask of the process is set.
- 24926 If *path* names a symbolic link, *mknod()* shall fail and set *errno* to [EEXIST].
- 24927 Upon successful completion, *mknod()* shall mark for update the *st\_atime*, *st\_ctime*, and *st\_mtime*  
 24928 fields of the file. Also, the *st\_ctime* and *st\_mtime* fields of the directory that contains the new  
 24929 entry shall be marked for update.
- 24930 Only a process with appropriate privileges may invoke *mknod()* for file types other than FIFO-  
 24931 special.
- 24932 **RETURN VALUE**
- 24933 Upon successful completion, *mknod()* shall return 0. Otherwise, it shall return -1, the new file  
 24934 shall not be created, and *errno* shall be set to indicate the error.
- 24935 **ERRORS**
- 24936 The *mknod()* function shall fail if:
- 24937 [EACCES] A component of the path prefix denies search permission, or write permission  
 24938 is denied on the parent directory.
- 24939 [EEXIST] The named file exists.
- 24940 [EINVAL] An invalid argument exists.
- 24941 [EIO] An I/O error occurred while accessing the file system.
- 24942 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*  
 24943 argument.
- 24944 [ENAMETOOLONG]  
 24945 The length of a pathname exceeds {PATH\_MAX} or a pathname component is  
 24946 longer than {NAME\_MAX}.
- 24947 [ENOENT] A component of the path prefix specified by *path* does not name an existing  
 24948 directory or *path* is an empty string.
- 24949 [ENOSPC] The directory that would contain the new file cannot be extended or the file  
 24950 system is out of file allocation resources.
- 24951 [ENOTDIR] A component of the path prefix is not a directory.
- 24952 [EPERM] The invoking process does not have appropriate privileges and the file type is  
 24953 not FIFO-special.
- 24954 [EROFS] The directory in which the file is to be created is located on a read-only file  
 24955 system.
- 24956 The *mknod()* function may fail if:
- 24957 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
 24958 resolution of the *path* argument.
- 24959 [ENAMETOOLONG]  
 24960 Pathname resolution of a symbolic link produced an intermediate result  
 24961 whose length exceeds {PATH\_MAX}.

24962 **EXAMPLES**24963 **Creating a FIFO Special File**

24964 The following example shows how to create a FIFO special file named `/home/cnd/mod_done`,  
 24965 with read/write permissions for owner, and with read permissions for group and others.

```
24966 #include <sys/types.h>
24967 #include <sys/stat.h>

24968 dev_t dev;
24969 int status;
24970 ...
24971 status = mknod("/home/cnd/mod_done", S_IFIFO | S_IWUSR |
24972 S_IRUSR | S_IRGRP | S_IROTH, dev);
```

24973 **APPLICATION USAGE**

24974 `mkfifo()` is preferred over this function for making FIFO special files.

24975 **RATIONALE**

24976 The POSIX.1-1990 standard required that the group ID of a newly created file be set to the group  
 24977 ID of its parent directory or to the effective group ID of the creating process. FIPS 151-2 required  
 24978 that implementations provide a way to have the group ID be set to the group ID of the  
 24979 containing directory, but did not prohibit implementations also supporting a way to set the  
 24980 group ID to the effective group ID of the creating process. Conforming applications should not  
 24981 assume which group ID will be used. If it matters, an application can use `chown()` to set the  
 24982 group ID after the file is created, or determine under what conditions the implementation will  
 24983 set the desired group ID.

24984 **FUTURE DIRECTIONS**

24985 None.

24986 **SEE ALSO**

24987 `chmod()`, `creat()`, `exec`, `mkdir()`, `mkfifo()`, `open()`, `stat()`, `umask()`, the Base Definitions volume of  
 24988 IEEE Std 1003.1-200x, `<sys/stat.h>`

24989 **CHANGE HISTORY**

24990 First released in Issue 4, Version 2.

24991 **Issue 5**

24992 Moved from X/OPEN UNIX extension to BASE.

24993 **Issue 6**

24994 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

24995 The wording of the mandatory [ELOOP] error condition is updated, and a second optional  
 24996 [ELOOP] error condition is added.

24997 **NAME**24998 `mkstemp` — make a unique filename24999 **SYNOPSIS**25000 XSI `#include <stdlib.h>`25001 `int mkstemp(char *template);`

25002

25003 **DESCRIPTION**

25004 The `mkstemp()` function shall replace the contents of the string pointed to by *template* by a unique  
25005 filename, and return a file descriptor for the file open for reading and writing. The function thus  
25006 prevents any possible race condition between testing whether the file exists and opening it for  
25007 use. The string in *template* should look like a filename with six trailing *Xs*; `mkstemp()` replaces  
25008 each *X* with a character from the portable filename character set. The characters are chosen such  
25009 that the resulting name does not duplicate the name of an existing file at the time of a call to  
25010 `mkstemp()`.

25011 **RETURN VALUE**

25012 Upon successful completion, `mkstemp()` shall return an open file descriptor. Otherwise, `-1` shall  
25013 be returned if no suitable file could be created.

25014 **ERRORS**

25015 No errors are defined.

25016 **EXAMPLES**25017 **Generating a Filename**

25018 The following example creates a file with a 10-character name beginning with the characters  
25019 "file" and opens the file for reading and writing. The value returned as the value of *fd* is a file  
25020 descriptor that identifies the file.

25021 `#include <stdlib.h>`25022 `...`25023 `char template[] = "/tmp/fileXXXXXX";`25024 `int fd;`25025 `fd = mkstemp(template);`25026 **APPLICATION USAGE**

25027 It is possible to run out of letters.

25028 The `mkstemp()` function need not check to determine whether the filename part of *template*  
25029 exceeds the maximum allowable filename length.

25030 **RATIONALE**

25031 None.

25032 **FUTURE DIRECTIONS**

25033 None.

25034 **SEE ALSO**

25035 `getpid()`, `open()`, `tmpfile()`, `tmpnam()`, the Base Definitions volume of IEEE Std 1003.1-200x,  
25036 `<stdlib.h>`

25037 **CHANGE HISTORY**

25038           First released in Issue 4, Version 2.

25039 **Issue 5**

25040           Moved from X/OPEN UNIX extension to BASE.

25041 **NAME**25042 mktemp — make a unique filename (**LEGACY**)25043 **SYNOPSIS**25044 XSI `#include <stdlib.h>`25045 `char *mktemp(char *template);`

25046

25047 **DESCRIPTION**

25048 The *mktemp()* function shall replace the contents of the string pointed to by *template* by a unique  
25049 filename and return *template*. The application shall initialize *template* to be a filename with six  
25050 trailing *X*s; *mktemp()* shall replace each *X* with a single byte character from the portable filename  
25051 character set.

25052 **RETURN VALUE**

25053 The *mktemp()* function shall return the pointer *template*. If a unique name cannot be created,  
25054 *template* shall point to a null string.

25055 **ERRORS**

25056 No errors are defined.

25057 **EXAMPLES**25058 **Generating a Filename**

25059 The following example replaces the contents of the "template" string with a 10-character  
25060 filename beginning with the characters "file" and returns a pointer to the "template" string  
25061 that contains the new filename.

25062 `#include <stdlib.h>`25063 `...`25064 `char *template = "/tmp/fileXXXXXX";`25065 `char *ptr;`25066 `ptr = mktemp(template);`25067 **APPLICATION USAGE**

25068 Between the time a pathname is created and the file opened, it is possible for some other process  
25069 to create a file with the same name. The *mkstemp()* function avoids this problem and is preferred  
25070 over this function.

25071 **RATIONALE**

25072 None.

25073 **FUTURE DIRECTIONS**

25074 This function may be withdrawn in a future version.

25075 **SEE ALSO**25076 *mkstemp()*, *tmpfile()*, *tmpnam()*, the Base Definitions volume of IEEE Std 1003.1-200x, `<stdlib.h>`25077 **CHANGE HISTORY**

25078 First released in Issue 4, Version 2.

25079 **Issue 5**

25080 Moved from X/OPEN UNIX extension to BASE.

25081 **Issue 6**

25082 This function is marked LEGACY.

25083 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

25084 **NAME**

25085           mktime — convert broken-down time into time since the Epoch

25086 **SYNOPSIS**

25087           #include &lt;time.h&gt;

25088           time\_t mktime(struct tm \*timeptr);

25089 **DESCRIPTION**

25090 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
 25091 conflict between the requirements described here and the ISO C standard is unintentional. This  
 25092 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

25093       The *mktime()* function shall convert the broken-down time, expressed as local time, in the  
 25094 structure pointed to by *timeptr*, into a time since the Epoch value with the same encoding as that  
 25095 of the values returned by *time()*. The original values of the *tm\_wday* and *tm\_yday* components of  
 25096 the structure are ignored, and the original values of the other components are not restricted to  
 25097 the ranges described in <**time.h**>.

25098 **CX**       A positive or 0 value for *tm\_isdst* shall cause *mktime()* to presume initially that Daylight Savings  
 25099 Time, respectively, is or is not in effect for the specified time. A negative value for *tm\_isdst* shall  
 25100 cause *mktime()* to attempt to determine whether Daylight Saving Time is in effect for the  
 25101 specified time.

25102       Local timezone information shall be set as though *mktime()* called *tzset()*.

25103       The relationship between the **tm** structure (defined in the <**time.h**> header) and the time in  
 25104 seconds since the Epoch is that the result shall be as specified in the expression given in the  
 25105 definition of seconds since the Epoch (see the Base Definitions volume of IEEE Std 1003.1-200x,  
 25106 Section 4.14, Seconds Since the Epoch) corrected for timezone and any seasonal time  
 25107 adjustments, where the names in the structure and in the expression correspond.

25108       Upon successful completion, the values of the *tm\_wday* and *tm\_yday* components of the structure  
 25109 shall be set appropriately, and the other components are set to represent the specified time since  
 25110 the Epoch, but with their values forced to the ranges indicated in the <**time.h**> entry; the final  
 25111 value of *tm\_mday* shall not be set until *tm\_mon* and *tm\_year* are determined.

25112 **RETURN VALUE**

25113       The *mktime()* function shall return the specified time since the Epoch encoded as a value of type  
 25114 **time\_t**. If the time since the Epoch cannot be represented, the function shall return the value  
 25115 (**time\_t**)-1.

25116 **ERRORS**

25117       No errors are defined.

25118 **EXAMPLES**

25119       What day of the week is July 4, 2001?

25120       #include &lt;stdio.h&gt;

25121       #include &lt;time.h&gt;

25122       struct tm time\_str;

25123       char daybuf[20];

25124       int main(void)

25125       {

25126           time\_str.tm\_year = 2001 - 1900;

25127           time\_str.tm\_mon = 7 - 1;

25128           time\_str.tm\_mday = 4;

```
25129         time_str.tm_hour = 0;
25130         time_str.tm_min = 0;
25131         time_str.tm_sec = 1;
25132         time_str.tm_isdst = -1;
25133         if (mktime(&time_str) == -1)
25134             (void)puts("-unknown-");
25135         else {
25136             (void)strftime(daybuf, sizeof(daybuf), "%A", &time_str);
25137             (void)puts(daybuf);
25138         }
25139         return 0;
25140     }
```

**25141 APPLICATION USAGE**

25142 None.

**25143 RATIONALE**

25144 None.

**25145 FUTURE DIRECTIONS**

25146 None.

**25147 SEE ALSO**

25148 *asctime()*, *clock()*, *ctime()*, *difftime()*, *gmtime()*, *localtime()*, *strftime()*, *strptime()*, *time()*, *utime()*,  
25149 the Base Definitions volume of IEEE Std 1003.1-200x, <**time.h**>

**25150 CHANGE HISTORY**

25151 First released in Issue 3.

25152 Entry included for alignment with the POSIX.1-1988 standard and the ANSI C standard.

**25153 Issue 6**

25154 Extensions beyond the ISO C standard are now marked.

25155 **NAME**25156 mlock, munlock — lock or unlock a range of process address space (**REALTIME**)25157 **SYNOPSIS**25158 MLR `#include <sys/mman.h>`25159 `int mlock(const void *addr, size_t len);`25160 `int munlock(const void *addr, size_t len);`

25161

25162 **DESCRIPTION**

25163 The *mlock()* function shall cause those whole pages containing any part of the address space of  
 25164 the process starting at address *addr* and continuing for *len* bytes to be memory-resident until  
 25165 unlocked or until the process exits or *execs* another process image. The implementation may  
 25166 require that *addr* be a multiple of {PAGESIZE}.

25167 The *munlock()* function shall unlock those whole pages containing any part of the address space  
 25168 of the process starting at address *addr* and continuing for *len* bytes, regardless of how many  
 25169 times *mlock()* has been called by the process for any of the pages in the specified range. The  
 25170 implementation may require that *addr* be a multiple of {PAGESIZE}.

25171 If any of the pages in the range specified to a call to *munlock()* are also mapped into the address  
 25172 spaces of other processes, any locks established on those pages by another process are  
 25173 unaffected by the call of this process to *munlock()*. If any of the pages in the range specified by a  
 25174 call to *munlock()* are also mapped into other portions of the address space of the calling process  
 25175 outside the range specified, any locks established on those pages via the other mappings are also  
 25176 unaffected by this call.

25177 Upon successful return from *mlock()*, pages in the specified range shall be locked and memory-  
 25178 resident. Upon successful return from *munlock()*, pages in the specified range shall be unlocked  
 25179 with respect to the address space of the process. Memory residency of unlocked pages is  
 25180 unspecified.

25181 The appropriate privilege is required to lock process memory with *mlock()*.

25182 **RETURN VALUE**

25183 Upon successful completion, the *mlock()* and *munlock()* functions shall return a value of zero.  
 25184 Otherwise, no change is made to any locks in the address space of the process, and the function  
 25185 shall return a value of  $-1$  and set *errno* to indicate the error.

25186 **ERRORS**

25187 The *mlock()* and *munlock()* functions shall fail if:

25188 [ENOMEM] Some or all of the address range specified by the *addr* and *len* arguments does  
 25189 not correspond to valid mapped pages in the address space of the process.

25190 The *mlock()* function shall fail if:

25191 [EAGAIN] Some or all of the memory identified by the operation could not be locked  
 25192 when the call was made.

25193 The *mlock()* and *munlock()* functions may fail if:

25194 [EINVAL] The *addr* argument is not a multiple of {PAGESIZE}.

25195 The *mlock()* function may fail if:

25196 [ENOMEM] Locking the pages mapped by the specified range would exceed an  
 25197 implementation-defined limit on the amount of memory that the process may  
 25198 lock.

25199 [EPERM] The calling process does not have the appropriate privilege to perform the  
25200 requested operation.

25201 **EXAMPLES**

25202 None.

25203 **APPLICATION USAGE**

25204 None.

25205 **RATIONALE**

25206 None.

25207 **FUTURE DIRECTIONS**

25208 None.

25209 **SEE ALSO**

25210 *exec*, *exit()*, *fork()*, *mlockall()*, *munmap()*, the Base Definitions volume of IEEE Std 1003.1-200x,  
25211 **<sys/mman.h>**

25212 **CHANGE HISTORY**

25213 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

25214 **Issue 6**

25215 The *mlock()* and *munlock()* functions are marked as part of the Range Memory Locking option.

25216 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
25217 implementation does not support the Range Memory Locking option.

25218 **NAME**25219 mlockall, munlockall — lock/unlock the address space of a process (**REALTIME**)25220 **SYNOPSIS**25221 ML 

```
#include <sys/mman.h>
```

25222 

```
int mlockall(int flags);
```

25223 

```
int munlockall(void);
```

25224

25225 **DESCRIPTION**

25226 The *mlockall()* function shall cause all of the pages mapped by the address space of a process to  
 25227 be memory-resident until unlocked or until the process exits or *execs* another process image. The  
 25228 *flags* argument determines whether the pages to be locked are those currently mapped by the  
 25229 address space of the process, those that are mapped in the future, or both. The *flags* argument is  
 25230 constructed from the bitwise-inclusive OR of one or more of the following symbolic constants,  
 25231 defined in *<sys/mman.h>*:

25232 MCL\_CURRENT Lock all of the pages currently mapped into the address space of the process.

25233 MCL\_FUTURE Lock all of the pages that become mapped into the address space of the  
25234 process in the future, when those mappings are established.

25235 If MCL\_FUTURE is specified, and the automatic locking of future mappings eventually causes  
 25236 the amount of locked memory to exceed the amount of available physical memory or any other  
 25237 implementation-defined limit, the behavior is implementation-defined. The manner in which the  
 25238 implementation informs the application of these situations is also implementation-defined.

25239 The *munlockall()* function shall unlock all currently mapped pages of the address space of the  
 25240 process. Any pages that become mapped into the address space of the process after a call to  
 25241 *munlockall()* shall not be locked, unless there is an intervening call to *mlockall()* specifying  
 25242 MCL\_FUTURE or a subsequent call to *mlockall()* specifying MCL\_CURRENT. If pages mapped  
 25243 into the address space of the process are also mapped into the address spaces of other processes  
 25244 and are locked by those processes, the locks established by the other processes shall be  
 25245 unaffected by a call by this process to *munlockall()*.

25246 Upon successful return from the *mlockall()* function that specifies MCL\_CURRENT, all currently  
 25247 mapped pages of the process' address space shall be memory-resident and locked. Upon return  
 25248 from the *munlockall()* function, all currently mapped pages of the process' address space shall be  
 25249 unlocked with respect to the process' address space. The memory residency of unlocked pages is  
 25250 unspecified.

25251 The appropriate privilege is required to lock process memory with *mlockall()*.25252 **RETURN VALUE**

25253 Upon successful completion, the *mlockall()* function shall return a value of zero. Otherwise, no  
 25254 additional memory shall be locked, and the function shall return a value of  $-1$  and set *errno* to  
 25255 indicate the error. The effect of failure of *mlockall()* on previously existing locks in the address  
 25256 space is unspecified.

25257 If it is supported by the implementation, the *munlockall()* function shall always return a value of  
 25258 zero. Otherwise, the function shall return a value of  $-1$  and set *errno* to indicate the error.

25259 **ERRORS**25260 The *mlockall()* function shall fail if:

25261 [EAGAIN] Some or all of the memory identified by the operation could not be locked  
 25262 when the call was made.

25263 [EINVAL] The *flags* argument is zero, or includes unimplemented flags.

25264 The *mlockall()* function may fail if:

25265 [ENOMEM] Locking all of the pages currently mapped into the address space of the  
25266 process would exceed an implementation-defined limit on the amount of  
25267 memory that the process may lock.

25268 [EPERM] The calling process does not have the appropriate privilege to perform the  
25269 requested operation.

25270 **EXAMPLES**

25271 None.

25272 **APPLICATION USAGE**

25273 None.

25274 **RATIONALE**

25275 None.

25276 **FUTURE DIRECTIONS**

25277 None.

25278 **SEE ALSO**

25279 *exec*, *exit()*, *fork()*, *mlock()*, *munmap()*, the Base Definitions volume of IEEE Std 1003.1-200x,  
25280 <*sys/mman.h*>

25281 **CHANGE HISTORY**

25282 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

25283 **Issue 6**

25284 The *mlockall()* and *munlockall()* functions are marked as part of the Process Memory Locking  
25285 option.

25286 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
25287 implementation does not support the Process Memory Locking option.

## 25288 NAME

25289 mmap — map pages of memory

## 25290 SYNOPSIS

25291 MF|SHM #include &lt;sys/mman.h&gt;

```
25292 void *mmap(void *addr, size_t len, int prot, int flags,
25293           int fildes, off_t off);
25294
```

## 25295 DESCRIPTION

25296 The *mmap()* function shall establish a mapping between a process' address space and a file, |  
 25297 TYM shared memory object, or typed memory object. The format of the call is as follows:

```
25298 pa=mmap(addr, len, prot, flags, fildes, off);
```

25299 The *mmap()* function shall establish a mapping between the address space of the process at an |  
 25300 address *pa* for *len* bytes to the memory object represented by the file descriptor *fildes* at offset *off* |  
 25301 for *len* bytes. The value of *pa* is an implementation-defined function of the parameter *addr* and |  
 25302 the values of *flags*, further described below. A successful *mmap()* call shall return *pa* as its result. |  
 25303 The address range starting at *pa* and continuing for *len* bytes shall be legitimate for the possible |  
 25304 (not necessarily current) address space of the process. The range of bytes starting at *off* and |  
 25305 continuing for *len* bytes shall be legitimate for the possible (not necessarily current) offsets in the |  
 25306 TYM file, shared memory object, or typed memory object represented by *fildes*.

25307 TYM If *fildes* represents a typed memory object opened with either the |  
 25308 POSIX\_TYPED\_MEM\_ALLOCATE flag or the POSIX\_TYPED\_MEM\_ALLOCATE\_CONTIG |  
 25309 flag, the memory object to be mapped shall be that portion of the typed memory object allocated |  
 25310 by the implementation as specified below. In this case, if *off* is non-zero, the behavior of *mmap()* |  
 25311 is undefined. If *fildes* refers to a valid typed memory object that is not accessible from the calling |  
 25312 process, *mmap()* shall fail.

25313 The mapping established by *mmap()* shall replace any previous mappings for those whole pages |  
 25314 containing any part of the address space of the process starting at *pa* and continuing for *len* |  
 25315 bytes.

25316 If the size of the mapped file changes after the call to *mmap()* as a result of some other operation |  
 25317 on the mapped file, the effect of references to portions of the mapped region that correspond to |  
 25318 added or removed portions of the file is unspecified.

25319 TYM The *mmap()* function shall be supported for regular files, shared memory objects, and typed |  
 25320 memory objects. Support for any other type of file is unspecified.

25321 The parameter *prot* determines whether read, write, execute, or some combination of accesses |  
 25322 are permitted to the data being mapped. The *prot* shall be either PROT\_NONE or the bitwise- |  
 25323 inclusive OR of one or more of the other flags in the following table, defined in the |  
 25324 <sys/mman.h> header.

25325

25326

25327

25328

25329

25330

Symbolic Constant	Description
PROT_READ	Data can be read.
PROT_WRITE	Data can be written.
PROT_EXEC	Data can be executed.
PROT_NONE	Data cannot be accessed.

25331 If an implementation cannot support the combination of access types specified by *prot*, the call |  
 25332 MPR to *mmap()* shall fail. An implementation may permit accesses other than those specified by *prot*; |  
 25333 however, if the Memory Protection option is supported, the implementation shall not permit a

25334 write to succeed where PROT\_WRITE has not been set or shall not permit any access where  
 25335 PROT\_NONE alone has been set. The implementation shall support at least the following values  
 25336 of *prot*: PROT\_NONE, PROT\_READ, PROT\_WRITE, and the bitwise-inclusive OR of  
 25337 PROT\_READ and PROT\_WRITE. If the Memory Protection option is not supported, the result of  
 25338 any access that conflicts with the specified protection is undefined. The file descriptor *fildes* shall  
 25339 have been opened with read permission, regardless of the protection options specified. If  
 25340 PROT\_WRITE is specified, the application shall ensure that it has opened the file descriptor  
 25341 *fildes* with write permission unless MAP\_PRIVATE is specified in the *flags* parameter as  
 25342 described below.

25343 The parameter *flags* provides other information about the handling of the mapped data. The  
 25344 value of *flags* is the bitwise-inclusive OR of these options, defined in <sys/mman.h>:

25345

25346

25347

25348

25349

Symbolic Constant	Description
MAP_SHARED	Changes are shared.
MAP_PRIVATE	Changes are private.
MAP_FIXED	Interpret <i>addr</i> exactly.

25350 Implementations that do not support the Memory Mapped Files option are not required to  
 25351 support MAP\_PRIVATE.

25352 XSI It is implementation-defined whether MAP\_FIXED shall be supported. MAP\_FIXED shall be  
 25353 supported on XSI-conformant systems.

25354 MAP\_SHARED and MAP\_PRIVATE describe the disposition of write references to the memory  
 25355 object. If MAP\_SHARED is specified, write references shall change the underlying object. If  
 25356 MAP\_PRIVATE is specified, modifications to the mapped data by the calling process shall be  
 25357 visible only to the calling process and shall not change the underlying object. It is unspecified  
 25358 whether modifications to the underlying object done after the MAP\_PRIVATE mapping is  
 25359 established are visible through the MAP\_PRIVATE mapping. Either MAP\_SHARED or  
 25360 MAP\_PRIVATE can be specified, but not both. The mapping type is retained across *fork*().

25361 TYM When *fildes* represents a typed memory object opened with either the  
 25362 POSIX\_TYPED\_MEM\_ALLOCATE flag or the POSIX\_TYPED\_MEM\_ALLOCATE\_CONTIG  
 25363 flag, *mmap*( ) shall, if there are enough resources available, map *len* bytes allocated from the  
 25364 corresponding typed memory object which were not previously allocated to any process in any  
 25365 processor that may access that typed memory object. If there are not enough resources available,  
 25366 the function shall fail. If *fildes* represents a typed memory object opened with the  
 25367 POSIX\_TYPED\_MEM\_ALLOCATE\_CONTIG flag, these allocated bytes shall be contiguous  
 25368 within the typed memory object. If *fildes* represents a typed memory object opened with the  
 25369 POSIX\_TYPED\_MEM\_ALLOCATE flag, these allocated bytes may be composed of non-  
 25370 contiguous fragments within the typed memory object. If *fildes* represents a typed memory  
 25371 object opened with neither the POSIX\_TYPED\_MEM\_ALLOCATE\_CONTIG flag nor the  
 25372 POSIX\_TYPED\_MEM\_ALLOCATE flag, *len* bytes starting at offset *off* within the typed memory  
 25373 object are mapped, exactly as when mapping a file or shared memory object. In this case, if two  
 25374 processes map an area of typed memory using the same *off* and *len* values and using file  
 25375 descriptors that refer to the same memory pool (either from the same port or from a different  
 25376 port), both processes shall map the same region of storage.

25377 When MAP\_FIXED is set in the *flags* argument, the implementation is informed that the value of  
 25378 *pa* shall be *addr*, exactly. If MAP\_FIXED is set, *mmap*( ) may return MAP\_FAILED and set *errno* to  
 25379 [EINVAL]. If a MAP\_FIXED request is successful, the mapping established by *mmap*( ) replaces  
 25380 any previous mappings for the process' pages in the range [*pa*,*pa+len*).

25381 When MAP\_FIXED is not set, the implementation uses *addr* in an implementation-defined  
 25382 manner to arrive at *pa*. The *pa* so chosen shall be an area of the address space that the  
 25383 implementation deems suitable for a mapping of *len* bytes to the file. All implementations  
 25384 interpret an *addr* value of 0 as granting the implementation complete freedom in selecting *pa*,  
 25385 subject to constraints described below. A non-zero value of *addr* is taken to be a suggestion of a  
 25386 process address near which the mapping should be placed. When the implementation selects a  
 25387 value for *pa*, it never places a mapping at address 0, nor does it replace any extant mapping.

25388 The *off* argument is constrained to be aligned and sized according to the value returned by  
 25389 *sysconf()* when passed *\_SC\_PAGESIZE* or *\_SC\_PAGE\_SIZE*. When MAP\_FIXED is specified, the  
 25390 application shall ensure that the argument *addr* also meets these constraints. The  
 25391 implementation performs mapping operations over whole pages. Thus, while the argument *len*  
 25392 need not meet a size or alignment constraint, the implementation shall include, in any mapping  
 25393 operation, any partial page specified by the range [*pa*,*pa+len*).

25394 The system shall always zero-fill any partial page at the end of an object. Further, the system  
 25395 shall never write out any modified portions of the last page of an object which are beyond its  
 25396 MPR end. References within the address range starting at *pa* and continuing for *len* bytes to whole  
 25397 pages following the end of an object shall result in delivery of a SIGBUS signal.

25398 An implementation may generate SIGBUS signals when a reference would cause an error in the  
 25399 mapped object, such as out-of-space condition.

25400 The *mmap()* function shall add an extra reference to the file associated with the file descriptor  
 25401 *fdes* which is not removed by a subsequent *close()* on that file descriptor. This reference shall be  
 25402 removed when there are no more mappings to the file.

25403 The *st\_atime* field of the mapped file may be marked for update at any time between the *mmap()*  
 25404 call and the corresponding *munmap()* call. The initial read or write reference to a mapped region  
 25405 shall cause the file's *st\_atime* field to be marked for update if it has not already been marked for  
 25406 update.

25407 The *st\_ctime* and *st\_mtime* fields of a file that is mapped with MAP\_SHARED and PROT\_WRITE  
 25408 shall be marked for update at some point in the interval between a write reference to the  
 25409 mapped region and the next call to *msync()* with MS\_ASYNC or MS\_SYNC for that portion of  
 25410 the file by any process. If there is no such call and if the underlying file is modified as a result of  
 25411 a write reference, then these fields shall be marked for update at some time after the write  
 25412 reference.

25413 There may be implementation-defined limits on the number of memory regions that can be  
 25414 mapped (per process or per system).

25415 XSI If such a limit is imposed, whether the number of memory regions that can be mapped by a  
 25416 process is decreased by the use of *shmat()* is implementation-defined.

25417 If *mmap()* fails for reasons other than [EBADF], [EINVAL], or [ENOTSUP], some of the  
 25418 mappings in the address range starting at *addr* and continuing for *len* bytes may have been  
 25419 unmapped.

#### 25420 RETURN VALUE

25421 Upon successful completion, the *mmap()* function shall return the address at which the mapping  
 25422 was placed (*pa*); otherwise, it shall return a value of MAP\_FAILED and set *errno* to indicate the  
 25423 error. The symbol MAP\_FAILED is defined in the <sys/mman.h> header. No successful return  
 25424 from *mmap()* shall return the value MAP\_FAILED.

25425 **ERRORS**

- 25426 The *mmap()* function shall fail if:
- 25427 [EACCES] The *fildest* argument is not open for read, regardless of the protection specified,  
25428 or *fildest* is not open for write and PROT\_WRITE was specified for a  
25429 MAP\_SHARED type mapping.
- 25430 ML [EAGAIN] The mapping could not be locked in memory, if required by *mlockall()*, due to  
25431 a lack of resources.
- 25432 [EBADF] The *fildest* argument is not a valid open file descriptor.
- 25433 [EINVAL] The *addr* argument (if MAP\_FIXED was specified) or *off* is not a multiple of  
25434 the page size as returned by *sysconf()*, or are considered invalid by the  
25435 implementation.
- 25436 [EINVAL] The value of *flags* is invalid (neither MAP\_PRIVATE nor MAP\_SHARED is  
25437 set).
- 25438 [EMFILE] The number of mapped regions would exceed an implementation-defined  
25439 limit (per process or per system).
- 25440 [ENODEV] The *fildest* argument refers to a file whose type is not supported by *mmap()*.
- 25441 [ENOMEM] MAP\_FIXED was specified, and the range [*addr,addr+len*) exceeds that allowed  
25442 for the address space of a process; or, if MAP\_FIXED was not specified and  
25443 there is insufficient room in the address space to effect the mapping.
- 25444 ML [ENOMEM] The mapping could not be locked in memory, if required by *mlockall()*,  
25445 because it would require more space than the system is able to supply.
- 25446 MAP\_FIXED or MAP\_PRIVATE was specified in the *flags* argument and the  
25447 implementation does not support this functionality.
- 25448 TYM [ENOMEM] Not enough unallocated memory resources remain in the typed memory  
25449 object designated by *fildest* to allocate *len* bytes.
- 25450 [ENOTSUP] The implementation does not support the combination of accesses requested  
25451 in the *prot* argument.
- 25452 [ENXIO] Addresses in the range [*off,off+len*) are invalid for the object specified by *fildest*.
- 25453 [ENXIO] MAP\_FIXED was specified in *flags* and the combination of *addr*, *len*, and *off* is  
25454 invalid for the object specified by *fildest*.
- 25455 TYM [ENXIO] The *fildest* argument refers to a typed memory object that is not accessible from  
25456 the calling process.
- 25457 [EOVERFLOW] The file is a regular file and the value of *off* plus *len* exceeds the offset  
25458 maximum established in the open file description associated with *fildest*.

25459 **EXAMPLES**

25460 None.

25461 **APPLICATION USAGE**

25462 Use of *mmap()* may reduce the amount of memory available to other memory allocation  
25463 functions.

25464 Use of MAP\_FIXED may result in unspecified behavior in further use of *malloc()* and *shmat()*.  
25465 The use of MAP\_FIXED is discouraged, as it may prevent an implementation from making the  
25466 most effective use of resources.

25467 The application must ensure correct synchronization when using *mmap()* in conjunction with  
 25468 any other file access method, such as *read()* and *write()*, standard input/output, and *shmat()*.

25469 The *mmap()* function allows access to resources via address space manipulations, instead of  
 25470 *read()/write()*. Once a file is mapped, all a process has to do to access it is use the data at the  
 25471 address to which the file was mapped. So, using pseudo-code to illustrate the way in which an  
 25472 existing program might be changed to use *mmap()*, the following:

```
25473 fildes = open(...)
25474 lseek(fildes, some_offset)
25475 read(fildes, buf, len)
25476 /* Use data in buf. */
```

25477 becomes:

```
25478 fildes = open(...)
25479 address = mmap(0, len, PROT_READ, MAP_PRIVATE, fildes, some_offset)
25480 /* Use data at address. */
```

25481 The [EINVAL] error above is marked EX because it is defined as an optional error in the POSIX  
 25482 Realtime Extension.

#### 25483 RATIONALE

25484 After considering several other alternatives, it was decided to adopt the *mmap()* definition found  
 25485 in SVR4 for mapping memory objects into process address spaces. The SVR4 definition is  
 25486 minimal, in that it describes only what has been built, and what appears to be necessary for a  
 25487 general and portable mapping facility.

25488 Note that while *mmap()* was first designed for mapping files, it is actually a general-purpose  
 25489 mapping facility. It can be used to map any appropriate object, such as memory, files, devices,  
 25490 and so on, into the address space of a process.

25491 When a mapping is established, it is possible that the implementation may need to map more  
 25492 than is requested into the address space of the process because of hardware requirements. An  
 25493 application, however, cannot count on this behavior. Implementations that do not use a paged  
 25494 architecture may simply allocate a common memory region and return the address of it; such  
 25495 implementations probably do not allocate any more than is necessary. References past the end of  
 25496 the requested area are unspecified.

25497 If an application requests a mapping that would overlay existing mappings in the process, it  
 25498 might be desirable that an implementation detect this and inform the application. However, the  
 25499 default, portable (not MAP\_FIXED) operation does not overlay existing mappings. On the other  
 25500 hand, if the program specifies a fixed address mapping (which requires some implementation  
 25501 knowledge to determine a suitable address, if the function is supported at all), then the program  
 25502 is presumed to be successfully managing its own address space and should be trusted when it  
 25503 asks to map over existing data structures. Furthermore, it is also desirable to make as few system  
 25504 calls as possible, and it might be considered onerous to require an *munmap()* before an *mmap()*  
 25505 to the same address range. This volume of IEEE Std 1003.1-200x specifies that the new mappings  
 25506 replace any existing mappings, following existing practice in this regard.

25507 It is not expected, when the Memory Protection option is supported, that all hardware  
 25508 implementations are able to support all combinations of permissions at all addresses. When this  
 25509 option is supported, implementations are required to disallow write access to mappings without  
 25510 write permission and to disallow access to mappings without any access permission. Other than  
 25511 these restrictions, implementations may allow access types other than those requested by the  
 25512 application. For example, if the application requests only PROT\_WRITE, the implementation  
 25513 may also allow read access. A call to *mmap()* fails if the implementation cannot support allowing

25514 all the access requested by the application. For example, some implementations cannot support  
25515 a request for both write access and execute access simultaneously. All implementations  
25516 supporting the Memory Protection option must support requests for no access, read access,  
25517 write access, and both read and write access. Strictly conforming code must only rely on the  
25518 required checks. These restrictions allow for portability across a wide range of hardware.

25519 The MAP\_FIXED address treatment is likely to fail for non-page-aligned values and for certain  
25520 architecture-dependent address ranges. Conforming implementations cannot count on being  
25521 able to choose address values for MAP\_FIXED without utilizing non-portable, implementation-  
25522 defined knowledge. Nonetheless, MAP\_FIXED is provided as a standard interface conforming to  
25523 existing practice for utilizing such knowledge when it is available.

25524 Similarly, in order to allow implementations that do not support virtual addresses, support for  
25525 directly specifying any mapping addresses via MAP\_FIXED is not required and thus a  
25526 conforming application may not count on it.

25527 The MAP\_PRIVATE function can be implemented efficiently when memory protection hardware  
25528 is available. When such hardware is not available, implementations can implement such  
25529 “mappings” by simply making a real copy of the relevant data into process private memory,  
25530 though this tends to behave similarly to *read()*.

25531 The function has been defined to allow for many different models of using shared memory.  
25532 However, all uses are not equally portable across all machine architectures. In particular, the  
25533 *mmap()* function allows the system as well as the application to specify the address at which to  
25534 map a specific region of a memory object. The most portable way to use the function is always to  
25535 let the system choose the address, specifying NULL as the value for the argument *addr* and not  
25536 to specify MAP\_FIXED.

25537 If it is intended that a particular region of a memory object be mapped at the same address in a  
25538 group of processes (on machines where this is even possible), then MAP\_FIXED can be used to  
25539 pass in the desired mapping address. The system can still be used to choose the desired address  
25540 if the first such mapping is made without specifying MAP\_FIXED, and then the resulting  
25541 mapping address can be passed to subsequent processes for them to pass in via MAP\_FIXED.  
25542 The availability of a specific address range cannot be guaranteed, in general.

25543 The *mmap()* function can be used to map a region of memory that is larger than the current size  
25544 of the object. Memory access within the mapping but beyond the current end of the underlying  
25545 objects may result in SIGBUS signals being sent to the process. The reason for this is that the size  
25546 of the object can be manipulated by other processes and can change at any moment. The  
25547 implementation should tell the application that a memory reference is outside the object where  
25548 this can be detected; otherwise, written data may be lost and read data may not reflect actual  
25549 data in the object.

25550 Note that references beyond the end of the object do not extend the object as the new end cannot  
25551 be determined precisely by most virtual memory hardware. Instead, the size can be directly  
25552 manipulated by *ftruncate()*.

25553 Process memory locking does apply to shared memory regions, and the MEMLOCK\_FUTURE  
25554 argument to *memlockall()* can be relied upon to cause new shared memory regions to be  
25555 automatically locked.

25556 Existing implementations of *mmap()* return the value `-1` when unsuccessful. Since the casting of  
25557 this value to type `void *` cannot be guaranteed by the ISO C standard to be distinct from a  
25558 successful value, this volume of IEEE Std 1003.1-200x defines the symbol MAP\_FAILED, which a  
25559 conforming implementation does not return as the result of a successful call.

25560 **FUTURE DIRECTIONS**

25561 None.

25562 **SEE ALSO**

25563 *exec*, *fcntl()*, *fork()*, *lockf()*, *msync()*, *munmap()*, *mprotect()*, *posix\_typed\_mem\_open()*, *shmat()*,  
 25564 *sysconf()*, the Base Definitions volume of IEEE Std 1003.1-200x, <sys/mman.h>

25565 **CHANGE HISTORY**

25566 First released in Issue 4, Version 2.

25567 **Issue 5**

25568 Moved from X/OPEN UNIX extension to BASE.

25569 Aligned with *mmap()* in the POSIX Realtime Extension as follows:

- 25570 • The DESCRIPTION is extensively reworded.
- 25571 • The [EAGAIN] and [ENOTSUP] mandatory error conditions are added.
- 25572 • New cases of [ENOMEM] and [ENXIO] are added as mandatory error conditions.
- 25573 • The value returned on failure is the value of the constant MAP\_FAILED; this was previously  
 25574 defined as -1.

25575 Large File Summit extensions are added.

25576 **Issue 6**25577 The *mmap()* function is marked as part of the Memory Mapped Files option.

25578 The Open Group Corrigendum U028/6 is applied, changing (void \*)-1 to MAP\_FAILED.

25579 The following new requirements on POSIX implementations derive from alignment with the  
 25580 Single UNIX Specification:

- 25581 • The DESCRIPTION is updated to described the use of MAP\_FIXED.
- 25582 • The DESCRIPTION is updated to describe the addition of an extra reference to the file  
 25583 associated with the file descriptor passed to *mmap()*.
- 25584 • The DESCRIPTION is updated to state that there may be implementation-defined limits on  
 25585 the number of memory regions that can be mapped.
- 25586 • The DESCRIPTION is updated to describe constraints on the alignment and size of the *off*  
 25587 argument.
- 25588 • The [EINVAL] and [EMFILE] error conditions are added.
- 25589 • The [EOVERFLOW] error condition is added. This change is to support large files.

25590 The following changes are made for alignment with the ISO POSIX-1: 1996 standard:

- 25591 • The DESCRIPTION is updated to describe the cases when MAP\_PRIVATE and MAP\_FIXED  
 25592 need not be supported.

25593 The following changes are made for alignment with IEEE Std 1003.1j-2000:

- 25594 • Semantics for typed memory objects are added to the DESCRIPTION.
- 25595 • New [ENOMEM] and [ENXIO] errors are added to the ERRORS section.
- 25596 • The *posix\_typed\_mem\_open()* function is added to the SEE ALSO section.

25597 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

25598 **NAME**

25599 modf, modff, modfl — decompose a floating-point number

25600 **SYNOPSIS**

25601 #include <math.h>

25602 double modf(double *x*, double \**iptr*);

25603 float modff(float *value*, float \**iptr*);

25604 long double modfl(long double *value*, long double \**iptr*);

25605 **DESCRIPTION**

25606 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
25607 conflict between the requirements described here and the ISO C standard is unintentional. This  
25608 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

25609 These functions shall break the argument *x* into integral and fractional parts, each of which has  
25610 the same sign as the argument. It stores the integral part as a **double** (for the *modf()* function), a  
25611 **float** (for the *modff()* function), or a **long double** (for the *modfl()* function), in the object pointed  
25612 to by *iptr*.

25613 **RETURN VALUE**

25614 Upon successful completion, these functions shall return the signed fractional part of *x*.

25615 **MX** If *x* is NaN, a NaN shall be returned, and \**iptr* shall be set to a NaN.

25616 If *x* is ±Inf, ±0 shall be returned, and \**iptr* shall be set to ±Inf.

25617 **ERRORS**

25618 No errors are defined.

25619 **EXAMPLES**

25620 None.

25621 **APPLICATION USAGE**

25622 The *modf()* function computes the function result and \**iptr* such that:

25623 a = modf(x, &iptr) ;

25624 x == a+\*iptr ;

25625 allowing for the usual floating-point inaccuracies.

25626 **RATIONALE**

25627 None.

25628 **FUTURE DIRECTIONS**

25629 None.

25630 **SEE ALSO**

25631 *frexp()*, *isnan()*, *ldexp()*, the Base Definitions volume of IEEE Std 1003.1-200x, <math.h>

25632 **CHANGE HISTORY**

25633 First released in Issue 1. Derived from Issue 1 of the SVID.

25634 **Issue 5**

25635 The DESCRIPTION is updated to indicate how an application should check for an error. This  
25636 text was previously published in the APPLICATION USAGE section.

25637 **Issue 6**

25638 The *modff()* and *modfl()* functions are added for alignment with the ISO/IEC 9899:1999  
25639 standard.

25640 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
25641 revised to align with the ISO/IEC 9899:1999 standard.

25642 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
25643 marked.

25644 **NAME**

25645 mprotect — set protection of memory mapping

25646 **SYNOPSIS**

25647 MPR #include &lt;sys/mman.h&gt;

25648 int mprotect(void \*addr, size\_t len, int prot);

25649

25650 **DESCRIPTION**

25651 The *mprotect()* function shall change the access protections to be that specified by *prot* for those  
 25652 whole pages containing any part of the address space of the process starting at address *addr* and  
 25653 continuing for *len* bytes. The parameter *prot* determines whether read, write, execute, or some  
 25654 combination of accesses are permitted to the data being mapped. The *prot* argument should be  
 25655 either PROT\_NONE or the bitwise-inclusive OR of one or more of PROT\_READ, PROT\_WRITE,  
 25656 and PROT\_EXEC.

25657 If an implementation cannot support the combination of access types specified by *prot*, the call  
 25658 to *mprotect()* shall fail.

25659 An implementation may permit accesses other than those specified by *prot*; however, no  
 25660 implementation shall permit a write to succeed where PROT\_WRITE has not been set or shall  
 25661 permit any access where PROT\_NONE alone has been set. Implementations shall support at  
 25662 least the following values of *prot*: PROT\_NONE, PROT\_READ, PROT\_WRITE, and the bitwise-  
 25663 inclusive OR of PROT\_READ and PROT\_WRITE. If PROT\_WRITE is specified, the application  
 25664 shall ensure that it has opened the mapped objects in the specified address range with write  
 25665 permission, unless MAP\_PRIVATE was specified in the original mapping, regardless of whether  
 25666 the file descriptors used to map the objects have since been closed.

25667 The implementation shall require that *addr* be a multiple of the page size as returned by  
 25668 *sysconf()*.

25669 The behavior of this function is unspecified if the mapping was not established by a call to  
 25670 *mmap()*.

25671 When *mprotect()* fails for reasons other than [EINVAL], the protections on some of the pages in  
 25672 the range [*addr,addr+len*) may have been changed.

25673 **RETURN VALUE**

25674 Upon successful completion, *mprotect()* shall return 0; otherwise, it shall return -1 and set *errno*  
 25675 to indicate the error.

25676 **ERRORS**25677 The *mprotect()* function shall fail if:

25678 [EACCES] The *prot* argument specifies a protection that violates the access permission  
 25679 the process has to the underlying memory object.

25680 [EAGAIN] The *prot* argument specifies PROT\_WRITE over a MAP\_PRIVATE mapping  
 25681 and there are insufficient memory resources to reserve for locking the private  
 25682 page.

25683 [EINVAL] The *addr* argument is not a multiple of the page size as returned by *sysconf()*.

25684 [ENOMEM] Addresses in the range [*addr,addr+len*) are invalid for the address space of a  
 25685 process, or specify one or more pages which are not mapped.

25686 [ENOMEM] The *prot* argument specifies PROT\_WRITE on a MAP\_PRIVATE mapping, and  
 25687 it would require more space than the system is able to supply for locking the  
 25688 private pages, if required.

25689 [ENOTSUP] The implementation does not support the combination of accesses requested  
25690 in the *prot* argument.

25691 **EXAMPLES**

25692 None.

25693 **APPLICATION USAGE**

25694 The [EINVAL] error above is marked EX because it is defined as an optional error in the POSIX  
25695 Realtime Extension.

25696 **RATIONALE**

25697 None.

25698 **FUTURE DIRECTIONS**

25699 None.

25700 **SEE ALSO**

25701 *mmap()*, *sysconf()*, the Base Definitions volume of IEEE Std 1003.1-200x, <sys/mman.h>

25702 **CHANGE HISTORY**

25703 First released in Issue 4, Version 2.

25704 **Issue 5**

25705 Moved from X/OPEN UNIX extension to BASE.

25706 Aligned with *mprotect()* in the POSIX Realtime Extension as follows:

- 25707 • The DESCRIPTION is largely reworded.
- 25708 • [ENOTSUP] and a second form of [ENOMEM] are added as mandatory error conditions.
- 25709 • [EAGAIN] is moved from the optional to the mandatory error conditions.

25710 **Issue 6**

25711 The *mprotect()* function is marked as part of the Memory Protection option.

25712 The following new requirements on POSIX implementations derive from alignment with the  
25713 Single UNIX Specification:

- 25714 • The DESCRIPTION is updated to state that implementations require *addr* to be a multiple of  
25715 the page size as returned by *sysconf()*.
- 25716 • The [EINVAL] error condition is added.

25717 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

25718 **NAME**25719 mq\_close — close a message queue (**REALTIME**)25720 **SYNOPSIS**

25721 MSG #include &lt;mqueue.h&gt;

25722 int mq\_close(mqd\_t mqdes);

25723

25724 **DESCRIPTION**

25725 The *mq\_close()* function shall remove the association between the message queue descriptor,  
25726 *mqdes*, and its message queue. The results of using this message queue descriptor after  
25727 successful return from this *mq\_close()*, and until the return of this message queue descriptor  
25728 from a subsequent *mq\_open()*, are undefined.

25729 If the process has successfully attached a notification request to the message queue via this  
25730 *mqdes*, this attachment shall be removed, and the message queue is available for another process  
25731 to attach for notification.

25732 **RETURN VALUE**

25733 Upon successful completion, the *mq\_close()* function shall return a value of zero; otherwise, the  
25734 function shall return a value of -1 and set *errno* to indicate the error.

25735 **ERRORS**25736 The *mq\_close()* function shall fail if:25737 [EBADF] The *mqdes* argument is not a valid message queue descriptor.25738 **EXAMPLES**

25739 None.

25740 **APPLICATION USAGE**

25741 None.

25742 **RATIONALE**

25743 None.

25744 **FUTURE DIRECTIONS**

25745 None.

25746 **SEE ALSO**

25747 *mq\_open()*, *mq\_unlink()*, *msgctl()*, *msgget()*, *msgrcv()*, *msgsnd()*, the Base Definitions volume of  
25748 IEEE Std 1003.1-200x, <mqueue.h>

25749 **CHANGE HISTORY**

25750 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

25751 **Issue 6**25752 The *mq\_close()* function is marked as part of the Message Passing option.

25753 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
25754 implementation does not support the Message Passing option.

25755 **NAME**25756 mq\_getattr — get message queue attributes (**REALTIME**)25757 **SYNOPSIS**

25758 MSG #include &lt;mqqueue.h&gt;

25759 int mq\_getattr(mqd\_t mqdes, struct mq\_attr \*mqstat);

25760

25761 **DESCRIPTION**25762 The *mqdes* argument specifies a message queue descriptor.25763 The *mq\_getattr()* function shall obtain status information and attributes of the message queue |  
25764 and the open message queue description associated with the message queue descriptor. |25765 The results shall be returned in the **mq\_attr** structure referenced by the *mqstat* argument. |25766 Upon return, the following members shall have the values associated with the open message |  
25767 queue description as set when the message queue was opened and as modified by subsequent |  
25768 *mq\_setattr()* calls: *mq\_flags*. |25769 The following attributes of the message queue shall be returned as set at message queue |  
25770 creation: *mq\_maxmsg*, *mq\_msgsize*. |25771 Upon return, the following members within the **mq\_attr** structure referenced by the *mqstat* |  
25772 argument shall be set to the current state of the message queue: |25773 *mq\_curmsgs* The number of messages currently on the queue.25774 **RETURN VALUE**25775 Upon successful completion, the *mq\_getattr()* function shall return zero. Otherwise, the function |  
25776 shall return  $-1$  and set *errno* to indicate the error. |25777 **ERRORS**25778 The *mq\_getattr()* function shall fail if:25779 [EBADF] The *mqdes* argument is not a valid message queue descriptor.25780 **EXAMPLES**

25781 None.

25782 **APPLICATION USAGE**

25783 None.

25784 **RATIONALE**

25785 None.

25786 **FUTURE DIRECTIONS**

25787 None.

25788 **SEE ALSO**25789 *mq\_open()*, *mq\_send()*, *mq\_setattr()*, *mq\_timedsend()*, *msgctl()*, *msgget()*, *msgrcv()*, *msgsnd()*, the |  
25790 Base Definitions volume of IEEE Std 1003.1-200x, <mqqueue.h>25791 **CHANGE HISTORY**

25792 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

25793 **Issue 6**25794 The *mq\_getattr()* function is marked as part of the Message Passing option.25795 The [ENOSYS] error condition has been removed as stubs need not be provided if an |  
25796 implementation does not support the Message Passing option. |

25797  
25798

The *mq\_timedsend()* function is added to the SEE ALSO section for alignment with IEEE Std 1003.1d-1999.

25799 **NAME**25800 mq\_notify — notify process that a message is available (**REALTIME**)25801 **SYNOPSIS**

25802 MSG #include &lt;mqueue.h&gt;

25803 int mq\_notify(mqd\_t mqdes, const struct sigevent \*notification);

25804

25805 **DESCRIPTION**

25806 If the argument *notification* is not NULL, this function shall register the calling process to be |  
 25807 notified of message arrival at an empty message queue associated with the specified message |  
 25808 queue descriptor, *mqdes*. The notification specified by the *notification* argument shall be sent to |  
 25809 the process when the message queue transitions from empty to non-empty. At any time, only |  
 25810 one process may be registered for notification by a message queue. If the calling process or any |  
 25811 other process has already registered for notification of message arrival at the specified message |  
 25812 queue, subsequent attempts to register for that message queue shall fail. |

25813 If *notification* is NULL and the process is currently registered for notification by the specified |  
 25814 message queue, the existing registration shall be removed. |

25815 When the notification is sent to the registered process, its registration shall be removed. The |  
 25816 message queue shall then be available for registration.

25817 If a process has registered for notification of message arrival at a message queue and some |  
 25818 thread is blocked in *mq\_receive()* waiting to receive a message when a message arrives at the |  
 25819 queue, the arriving message shall satisfy the appropriate *mq\_receive()*. The resulting behavior is |  
 25820 as if the message queue remains empty, and no notification shall be sent. |

25821 **RETURN VALUE**

25822 Upon successful completion, the *mq\_notify()* function shall return a value of zero; otherwise, the |  
 25823 function shall return a value of -1 and set *errno* to indicate the error.

25824 **ERRORS**25825 The *mq\_notify()* function shall fail if:25826 [EBADF] The *mqdes* argument is not a valid message queue descriptor.

25827 [EBUSY] A process is already registered for notification by the message queue.

25828 **EXAMPLES**

25829 None.

25830 **APPLICATION USAGE**

25831 None.

25832 **RATIONALE**

25833 None.

25834 **FUTURE DIRECTIONS**

25835 None.

25836 **SEE ALSO**

25837 *mq\_open()*, *mq\_send()*, *mq\_timedsend()*, *msgctl()*, *msgget()*, *msgrcv()*, *msgsnd()*, the Base |  
 25838 Definitions volume of IEEE Std 1003.1-200x, <mqueue.h>

25839 **CHANGE HISTORY**

25840 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

25841 **Issue 6**

25842 The *mq\_notify()* function is marked as part of the Message Passing option.

25843 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
25844 implementation does not support the Message Passing option.

25845 The *mq\_timedsend()* function is added to the SEE ALSO section for alignment with  
25846 IEEE Std 1003.1d-1999.

25847 **NAME**25848 `mq_open` — open a message queue (**REALTIME**)25849 **SYNOPSIS**25850 MSG `#include <mqueue.h>`25851 `mqd_t mq_open(const char *name, int oflag, ...);`

25852

25853 **DESCRIPTION**

25854 The `mq_open()` function shall establish the connection between a process and a message queue  
 25855 with a message queue descriptor. It shall create an open message queue description that refers to  
 25856 the message queue, and a message queue descriptor that refers to that open message queue  
 25857 description. The message queue descriptor is used by other functions to refer to that message  
 25858 queue. The *name* argument points to a string naming a message queue. It is unspecified whether  
 25859 the name appears in the file system and is visible to other functions that take pathnames as |  
 25860 arguments. The *name* argument shall conform to the construction rules for a pathname. If *name* |  
 25861 begins with the slash character, then processes calling `mq_open()` with the same value of *name* |  
 25862 shall refer to the same message queue object, as long as that name has not been removed. If *name* |  
 25863 does not begin with the slash character, the effect is implementation-defined. The interpretation  
 25864 of slash characters other than the leading slash character in *name* is implementation-defined. If  
 25865 the *name* argument is not the name of an existing message queue and creation is not requested,  
 25866 `mq_open()` shall fail and return an error.

25867 A message queue descriptor may be implemented using a file descriptor, in which case  
 25868 applications can open up to at least {OPEN\_MAX} file and message queues.

25869 The *oflag* argument requests the desired receive and/or send access to the message queue. The  
 25870 requested access permission to receive messages or send messages shall be granted if the calling |  
 25871 process would be granted read or write access, respectively, to an equivalently protected file. |

25872 The value of *oflag* is the bitwise-inclusive OR of values from the following list. Applications |  
 25873 shall specify exactly one of the first three values (access modes) below in the value of *oflag*: |

25874 **O\_RDONLY** Open the message queue for receiving messages. The process can use the  
 25875 returned message queue descriptor with `mq_receive()`, but not `mq_send()`. A  
 25876 message queue may be open multiple times in the same or different processes  
 25877 for receiving messages.

25878 **O\_WRONLY** Open the queue for sending messages. The process can use the returned  
 25879 message queue descriptor with `mq_send()` but not `mq_receive()`. A message  
 25880 queue may be open multiple times in the same or different processes for  
 25881 sending messages.

25882 **O\_RDWR** Open the queue for both receiving and sending messages. The process can use  
 25883 any of the functions allowed for **O\_RDONLY** and **O\_WRONLY**. A message  
 25884 queue may be open multiple times in the same or different processes for  
 25885 sending messages.

25886 Any combination of the remaining flags may be specified in the value of *oflag*:

25887 **O\_CREAT** Create a message queue. It requires two additional arguments: *mode*, which |  
 25888 shall be of type **mode\_t**, and *attr*, which shall be a pointer to a **mq\_attr** |  
 25889 structure. If the pathname *name* has already been used to create a message |  
 25890 queue that still exists, then this flag shall have no effect, except as noted under |  
 25891 **O\_EXCL**. Otherwise, a message queue shall be created without any messages |  
 25892 in it. The user ID of the message queue shall be set to the effective user ID of |  
 25893 the process, and the group ID of the message queue shall be set to the effective |

25894 group ID of the process. The file permission bits shall be set to the value of |  
 25895 *mode*. When bits in *mode* other than file permission bits are set, the effect is |  
 25896 implementation-defined. If *attr* is NULL, the message queue shall be created |  
 25897 with implementation-defined default message queue attributes. If *attr* is non- |  
 25898 NULL and the calling process has the appropriate privilege on *name*, the |  
 25899 message queue *mq\_maxmsg* and *mq\_msgsize* attributes shall be set to the values |  
 25900 of the corresponding members in the **mq\_attr** structure referred to by *attr*. If |  
 25901 *attr* is non-NULL, but the calling process does not have the appropriate |  
 25902 privilege on *name*, the *mq\_open()* function shall fail and return an error |  
 25903 without creating the message queue.

25904 O\_EXCL If O\_EXCL and O\_CREAT are set, *mq\_open()* shall fail if the message queue |  
 25905 *name* exists. The check for the existence of the message queue and the creation |  
 25906 of the message queue if it does not exist shall be atomic with respect to other |  
 25907 threads executing *mq\_open()* naming the same *name* with O\_EXCL and |  
 25908 O\_CREAT set. If O\_EXCL is set and O\_CREAT is not set, the result is |  
 25909 undefined.

25910 O\_NONBLOCK Determines whether a *mq\_send()* or *mq\_receive()* waits for resources or |  
 25911 messages that are not currently available, or fails with *errno* set to [EAGAIN]; |  
 25912 see *mq\_send()* and *mq\_receive()* for details.

25913 The *mq\_open()* function does not add or remove messages from the queue.

25914 **RETURN VALUE**

25915 Upon successful completion, the function shall return a message queue descriptor; otherwise, |  
 25916 the function shall return (**mqd\_t**)−1 and set *errno* to indicate the error.

25917 **ERRORS**

25918 The *mq\_open()* function shall fail if:

25919 [EACCES] The message queue exists and the permissions specified by *oflag* are denied, or |  
 25920 the message queue does not exist and permission to create the message queue |  
 25921 is denied.

25922 [EEXIST] O\_CREAT and O\_EXCL are set and the named message queue already exists.

25923 [EINTR] The *mq\_open()* function was interrupted by a signal.

25924 [EINVAL] The *mq\_open()* function is not supported for the given name.

25925 [EINVAL] O\_CREAT was specified in *oflag*, the value of *attr* is not NULL, and either |  
 25926 *mq\_maxmsg* or *mq\_msgsize* was less than or equal to zero.

25927 [EMFILE] Too many message queue descriptors or file descriptors are currently in use by |  
 25928 this process.

25929 [ENAMETOOLONG]

25930 The length of the *name* argument exceeds {PATH\_MAX} or a pathname |  
 25931 component is longer than {NAME\_MAX}.

25932 [ENFILE] Too many message queues are currently open in the system.

25933 [ENOENT] O\_CREAT is not set and the named message queue does not exist.

25934 [ENOSPC] There is insufficient space for the creation of the new message queue.

25935 **EXAMPLES**

25936 None.

25937 **APPLICATION USAGE**

25938 None.

25939 **RATIONALE**

25940 None.

25941 **FUTURE DIRECTIONS**

25942 None.

25943 **SEE ALSO**

25944 *mq\_close()*, *mq\_getattr()*, *mq\_receive()*, *mq\_send()*, *mq\_setattr()*, *mq\_timedreceive()*, *mq\_timedsend()*,  
25945 *mq\_unlink()*, *msgctl()*, *msgget()*, *msgrcv()*, *msgsnd()*, the Base Definitions volume of  
25946 IEEE Std 1003.1-200x, <mqqueue.h>

25947 **CHANGE HISTORY**

25948 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

25949 **Issue 6**25950 The *mq\_open()* function is marked as part of the Message Passing option.

25951 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
25952 implementation does not support the Message Passing option.

25953 The *mq\_timedreceive()* and *mq\_timedsend()* functions are added to the SEE ALSO section for  
25954 alignment with IEEE Std 1003.1d-1999.

25955 The DESCRIPTION of O\_EXCL is updated in response to IEEE PASC Interpretation 1003.1c #48.

## 25956 NAME

25957 mq\_receive, mq\_timedreceive — receive a message from a message queue (**REALTIME**)

## 25958 SYNOPSIS

25959 MSG #include <mqueue.h>

```
25960 ssize_t mq_receive(mqd_t mqdes, char *msg_ptr, size_t msg_len,
25961 unsigned *msg_prio);
25962
```

25963 MSG TMO #include <mqueue.h>

25964 #include <time.h>

```
25965 ssize_t mq_timedreceive(mqd_t mqdes, char *restrict msg_ptr,
25966 size_t msg_len, unsigned *restrict msg_prio,
25967 const struct timespec *restrict abs_timeout);
25968
```

## 25969 DESCRIPTION

25970 The *mq\_receive()* function shall receive the oldest of the highest priority message(s) from the |  
 25971 message queue specified by *mqdes*. If the size of the buffer in bytes, specified by the *msg\_len* |  
 25972 argument, is less than the *mq\_msgsize* attribute of the message queue, the function shall fail and |  
 25973 return an error. Otherwise, the selected message shall be removed from the queue and copied to |  
 25974 the buffer pointed to by the *msg\_ptr* argument. |

25975 If the value of *msg\_len* is greater than {SSIZE\_MAX}, the result is implementation-defined.

25976 If the argument *msg\_prio* is not NULL, the priority of the selected message shall be stored in the |  
 25977 location referenced by *msg\_prio*.

25978 If the specified message queue is empty and O\_NONBLOCK is not set in the message queue |  
 25979 description associated with *mqdes*, *mq\_receive()* shall block until a message is enqueued on the |  
 25980 message queue or until *mq\_receive()* is interrupted by a signal. If more than one thread is waiting |  
 25981 to receive a message when a message arrives at an empty queue and the Priority Scheduling |  
 25982 option is supported, then the thread of highest priority that has been waiting the longest shall be |  
 25983 selected to receive the message. Otherwise, it is unspecified which waiting thread receives the |  
 25984 message. If the specified message queue is empty and O\_NONBLOCK is set in the message |  
 25985 queue description associated with *mqdes*, no message shall be removed from the queue, and |  
 25986 *mq\_receive()* shall return an error.

25987 TMO The *mq\_timedreceive()* function shall receive the oldest of the highest priority messages from the |  
 25988 message queue specified by *mqdes* as described for the *mq\_receive()* function. However, if |  
 25989 O\_NONBLOCK was not specified when the message queue was opened via the *mq\_open()* |  
 25990 function, and no message exists on the queue to satisfy the receive, the wait for such a message |  
 25991 shall be terminated when the specified timeout expires. If O\_NONBLOCK is set, this function is |  
 25992 equivalent to *mq\_receive()*. |

25993 The timeout expires when the absolute time specified by *abs\_timeout* passes, as measured by the |  
 25994 clock on which timeouts are based (that is, when the value of that clock equals or exceeds |  
 25995 *abs\_timeout*), or if the absolute time specified by *abs\_timeout* has already been passed at the time |  
 25996 of the call.

25997 TMO TMR If the Timers option is supported, the timeout shall be based on the CLOCK\_REALTIME clock; if |  
 25998 the Timers option is not supported, the timeout shall be based on the system clock as returned |  
 25999 by the *time()* function. |

26000 TMO The resolution of the timeout shall be the resolution of the clock on which it is based. The |  
 26001 *timespec* argument is defined in the <time.h> header. |

26002 Under no circumstance shall the operation fail with a timeout if a message can be removed from  
 26003 the message queue immediately. The validity of the *abs\_timeout* parameter need not be checked  
 26004 if a message can be removed from the message queue immediately.

#### 26005 RETURN VALUE

26006 TMO Upon successful completion, the *mq\_receive()* and *mq\_timedreceive()* functions shall return the  
 26007 length of the selected message in bytes and the message shall be removed from the queue.  
 26008 Otherwise, no message shall be removed from the queue, the functions shall return a value of -1,  
 26009 and set *errno* to indicate the error.

#### 26010 ERRORS

26011 TMO The *mq\_receive()* and *mq\_timedreceive()* functions shall fail if:

26012 [EAGAIN] O\_NONBLOCK was set in the message description associated with *mqdes*,  
 26013 and the specified message queue is empty.

26014 [EBADF] The *mqdes* argument is not a valid message queue descriptor open for reading.

26015 [EMSGSIZE] The specified message buffer size, *msg\_len*, is less than the message size  
 26016 attribute of the message queue.

26017 TMO [EINTR] The *mq\_receive()* or *mq\_timedreceive()* operation was interrupted by a signal.

26018 TMO [EINVAL] The process or thread would have blocked, and the *abs\_timeout* parameter  
 26019 specified a nanoseconds field value less than zero or greater than or equal to  
 26020 1 000 million.

26021 TMO [ETIMEDOUT] The O\_NONBLOCK flag was not set when the message queue was opened,  
 26022 but no message arrived on the queue before the specified timeout expired.

26023 TMO The *mq\_receive()* and *mq\_timedreceive()* functions may fail if:

26024 [EBADMSG] The implementation has detected a data corruption problem with the  
 26025 message.

#### 26026 EXAMPLES

26027 None.

#### 26028 APPLICATION USAGE

26029 None.

#### 26030 RATIONALE

26031 None.

#### 26032 FUTURE DIRECTIONS

26033 None.

#### 26034 SEE ALSO

26035 *mq\_open()*, *mq\_send()*, *mq\_timedsend()*, *msgctl()*, *msgget()*, *msgrcv()*, *msgsnd()*, *time()*, the Base  
 26036 Definitions volume of IEEE Std 1003.1-200x, <mqqueue.h>, <time.h>

#### 26037 CHANGE HISTORY

26038 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

#### 26039 Issue 6

26040 The *mq\_receive()* function is marked as part of the Message Passing option.

26041 The Open Group Corrigendum U021/4 is applied. The DESCRIPTION is changed to refer to  
 26042 *msg\_len* rather than *maxsize*.

26043 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
 26044 implementation does not support the Message Passing option.

26045 The following new requirements on POSIX implementations derive from alignment with the  
26046 Single UNIX Specification:

- 26047 • In this function it is possible for the return value to exceed the range of the type **ssize\_t** (since  
26048 **size\_t** has a larger range of positive values than **ssize\_t**). A sentence restricting the size of  
26049 the **size\_t** object is added to the description to resolve this conflict.

26050 The *mq\_timedreceive()* function is added for alignment with IEEE Std 1003.1d-1999.

26051 The **restrict** keyword is added to the *mq\_timedreceive()* prototype for alignment with the  
26052 ISO/IEC 9899:1999 standard.

26053 IEEE PASC Interpretation 1003.1 #109 is applied, correcting the return type for *mq\_timedreceive()*  
26054 from **int** to **ssize\_t**.

## 26055 NAME

26056 mq\_send, mq\_timedsend — send a message to a message queue (REALTIME)

## 26057 SYNOPSIS

26058 MSG #include &lt;mqueue.h&gt;

26059 int mq\_send(mqd\_t mqdes, const char \*msg\_ptr, size\_t msg\_len,  
26060 unsigned msg\_prio);

26061

26062 MSG TMO #include &lt;mqueue.h&gt;

26063 #include &lt;time.h&gt;

26064 int mq\_timedsend(mqd\_t mqdes, const char \*msg\_ptr, size\_t msg\_len,  
26065 unsigned msg\_prio, const struct timespec \*abs\_timeout);

26066

## 26067 DESCRIPTION

26068 The *mq\_send()* function shall add the message pointed to by the argument *msg\_ptr* to the  
26069 message queue specified by *mqdes*. The *msg\_len* argument specifies the length of the message, in  
26070 bytes, pointed to by *msg\_ptr*. The value of *msg\_len* shall be less than or equal to the *mq\_msgsize*  
26071 attribute of the message queue, or *mq\_send()* shall fail.

26072 If the specified message queue is not full, *mq\_send()* shall behave as if the message is inserted  
26073 into the message queue at the position indicated by the *msg\_prio* argument. A message with a  
26074 larger numeric value of *msg\_prio* shall be inserted before messages with lower values of  
26075 *msg\_prio*. A message shall be inserted after other messages in the queue, if any, with equal  
26076 *msg\_prio*. The value of *msg\_prio* shall be less than {MQ\_PRIO\_MAX}.

26077 If the specified message queue is full and O\_NONBLOCK is not set in the message queue  
26078 description associated with *mqdes*, *mq\_send()* shall block until space becomes available to  
26079 enqueue the message, or until *mq\_send()* is interrupted by a signal. If more than one thread is  
26080 waiting to send when space becomes available in the message queue and the Priority Scheduling  
26081 option is supported, then the thread of the highest priority that has been waiting the longest  
26082 shall be unblocked to send its message. Otherwise, it is unspecified which waiting thread is  
26083 unblocked. If the specified message queue is full and O\_NONBLOCK is set in the message  
26084 queue description associated with *mqdes*, the message shall not be queued and *mq\_send()* shall  
26085 return an error.

26086 TMO The *mq\_timedsend()* function shall add a message to the message queue specified by *mqdes* in the  
26087 manner defined for the *mq\_send()* function. However, if the specified message queue is full and  
26088 O\_NONBLOCK is not set in the message queue description associated with *mqdes*, the wait for  
26089 sufficient room in the queue shall be terminated when the specified timeout expires. If  
26090 O\_NONBLOCK is set in the message queue description, this function shall be equivalent to  
26091 *mq\_send()*.

26092 The timeout shall expire when the absolute time specified by *abs\_timeout* passes, as measured by  
26093 the clock on which timeouts are based (that is, when the value of that clock equals or exceeds  
26094 *abs\_timeout*), or if the absolute time specified by *abs\_timeout* has already been passed at the time  
26095 of the call.

26096 TMO TMR If the Timers option is supported, the timeout shall be based on the CLOCK\_REALTIME clock; if  
26097 the Timers option is not supported, the timeout shall be based on the system clock as returned  
26098 by the *time()* function.

26099 TMO The resolution of the timeout shall be the resolution of the clock on which it is based. The  
26100 *timespec* argument is defined in the <time.h> header.

26101 Under no circumstance shall the operation fail with a timeout if there is sufficient room in the  
 26102 queue to add the message immediately. The validity of the *abs\_timeout* parameter need not be  
 26103 checked when there is sufficient room in the queue.

#### 26104 RETURN VALUE

26105 TMO Upon successful completion, the *mq\_send()* and *mq\_timedsend()* functions shall return a value of  
 26106 zero. Otherwise, no message shall be enqueued, the functions shall return  $-1$ , and *errno* shall be  
 26107 set to indicate the error.

#### 26108 ERRORS

26109 TMO The *mq\_send()* and *mq\_timedsend()* functions shall fail if:

26110 [EAGAIN] The O\_NONBLOCK flag is set in the message queue description associated  
 26111 with *mqdes*, and the specified message queue is full.

26112 [EBADF] The *mqdes* argument is not a valid message queue descriptor open for writing.

26113 TMO [EINTR] A signal interrupted the call to *mq\_send()* or *mq\_timedsend()*.

26114 [EINVAL] The value of *msg\_prio* was outside the valid range.

26115 TMO [EINVAL] The process or thread would have blocked, and the *abs\_timeout* parameter  
 26116 specified a nanoseconds field value less than zero or greater than or equal to  
 26117 1 000 million.

26118 [EMSGSIZE] The specified message length, *msg\_len*, exceeds the message size attribute of  
 26119 the message queue.

26120 TMO [ETIMEDOUT] The O\_NONBLOCK flag was not set when the message queue was opened,  
 26121 but the timeout expired before the message could be added to the queue.

#### 26122 EXAMPLES

26123 None.

#### 26124 APPLICATION USAGE

26125 The value of the symbol {MQ\_PRIO\_MAX} limits the number of priority levels supported by the  
 26126 application. Message priorities range from 0 to {MQ\_PRIO\_MAX}-1.

#### 26127 RATIONALE

26128 None.

#### 26129 FUTURE DIRECTIONS

26130 None.

#### 26131 SEE ALSO

26132 *mq\_open()*, *mq\_receive()*, *mq\_setattr()*, *mq\_timedreceive()*, *time()*, the Base Definitions volume of  
 26133 IEEE Std 1003.1-200x, <mqqueue.h>, <time.h>

#### 26134 CHANGE HISTORY

26135 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

#### 26136 Issue 6

26137 The *mq\_send()* function is marked as part of the Message Passing option.

26138 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
 26139 implementation does not support the Message Passing option.

26140 The *mq\_timedsend()* function is added for alignment with IEEE Std 1003.1d-1999.

26141 **NAME**26142 mq\_setattr — set message queue attributes (**REALTIME**)26143 **SYNOPSIS**

26144 MSG #include &lt;mqqueue.h&gt;

```
26145 int mq_setattr(mqd_t mqdes, const struct mq_attr *restrict mqstat,
26146               struct mq_attr *restrict omqstat);
26147
```

26148 **DESCRIPTION**

26149 The *mq\_setattr()* function shall set attributes associated with the open message queue |  
 26150 description referenced by the message queue descriptor specified by *mqdes*. |

26151 The message queue attributes corresponding to the following members defined in the **mq\_attr** |  
 26152 structure shall be set to the specified values upon successful completion of *mq\_setattr()*: |

26153 *mq\_flags* The value of this member is the bitwise-logical OR of zero or more of |  
 26154 O\_NONBLOCK and any implementation-defined flags.

26155 The values of the *mq\_maxmsg*, *mq\_msgsize*, and *mq\_curmsgs* members of the **mq\_attr** structure |  
 26156 shall be ignored by *mq\_setattr()*.

26157 If *omqstat* is non-NULL, the *mq\_setattr()* function shall store, in the location referenced by |  
 26158 *omqstat*, the previous message queue attributes and the current queue status. These values shall |  
 26159 be the same as would be returned by a call to *mq\_getattr()* at that point. |

26160 **RETURN VALUE**

26161 Upon successful completion, the function shall return a value of zero and the attributes of the |  
 26162 message queue shall have been changed as specified.

26163 Otherwise, the message queue attributes shall be unchanged, and the function shall return a |  
 26164 value of -1 and set *errno* to indicate the error.

26165 **ERRORS**26166 The *mq\_setattr()* function shall fail if:

26167 [EBADF] The *mqdes* argument is not a valid message queue descriptor.

26168 **EXAMPLES**

26169 None.

26170 **APPLICATION USAGE**

26171 None.

26172 **RATIONALE**

26173 None.

26174 **FUTURE DIRECTIONS**

26175 None.

26176 **SEE ALSO**

26177 *mq\_open()*, *mq\_send()*, *mq\_timedsend()*, *msgctl()*, *msgget()*, *msgrcv()*, *msgsnd()*, the Base |  
 26178 Definitions volume of IEEE Std 1003.1-200x, <mqqueue.h>

26179 **CHANGE HISTORY**

26180 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

26181 **Issue 6**

26182 The *mq\_setattr()* function is marked as part of the Message Passing option.

26183 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
26184 implementation does not support the Message Passing option.

26185 The *mq\_timedsend()* function is added to the SEE ALSO section for alignment with  
26186 IEEE Std 1003.1d-1999.

26187 The **restrict** keyword is added to the *mq\_setattr()* prototype for alignment with the  
26188 ISO/IEC 9899:1999 standard.

26189 **NAME**

26190 mq\_timedreceive — receive a message from a message queue (**ADVANCED REALTIME**)

26191 **SYNOPSIS**

```
26192 MSG TMO #include <mqueue.h>
```

```
26193 #include <time.h>
```

```
26194 ssize_t mq_timedreceive(mqd_t mqdes, char *restrict msg_ptr,
```

```
26195 size_t msg_len, unsigned *restrict msg_prio,
```

```
26196 const struct timespec *restrict abs_timeout);
```

```
26197
```

26198 **DESCRIPTION**

26199 Refer to *mq\_receive()*.

26200 **NAME**

26201 mq\_timedsend — send a message to a message queue (**ADVANCED REALTIME**)

26202 **SYNOPSIS**

26203 MSG TMO #include <mqueue.h>

26204 #include <time.h>

```
26205 int mq_timedsend(mqd_t mqdes, const char *msg_ptr, size_t msg_len,  
26206                 unsigned msg_prio, const struct timespec *abs_timeout);
```

26207

26208 **DESCRIPTION**

26209 Refer to *mq\_send()*.

26210 **NAME**26211 mq\_unlink — remove a message queue (**REALTIME**)26212 **SYNOPSIS**

26213 MSG #include &lt;mqueue.h&gt;

26214 int mq\_unlink(const char \*name);

26215

26216 **DESCRIPTION**

26217 The *mq\_unlink()* function shall remove the message queue named by the pathname *name*. After |  
 26218 a successful call to *mq\_unlink()* with *name*, a call to *mq\_open()* with *name* shall fail if the flag |  
 26219 O\_CREAT is not set in *flags*. If one or more processes have the message queue open when |  
 26220 *mq\_unlink()* is called, destruction of the message queue shall be postponed until all references to |  
 26221 the message queue have been closed.

26222 Calls to *mq\_open()* to recreate the message queue may fail until the message queue is actually |  
 26223 removed. However, the *mq\_unlink()* call need not block until all references have been closed; it |  
 26224 may return immediately.

26225 **RETURN VALUE**

26226 Upon successful completion, the function shall return a value of zero. Otherwise, the named |  
 26227 message queue shall be unchanged by this function call, and the function shall return a value of |  
 26228 -1 and set *errno* to indicate the error.

26229 **ERRORS**26230 The *mq\_unlink()* function shall fail if:

26231 [EACCES] Permission is denied to unlink the named message queue.

26232 [ENAMETOOLONG]

26233 The length of the *name* argument exceeds {PATH\_MAX} or a pathname |  
 26234 component is longer than {NAME\_MAX}.

26235 [ENOENT] The named message queue does not exist.

26236 **EXAMPLES**

26237 None.

26238 **APPLICATION USAGE**

26239 None.

26240 **RATIONALE**

26241 None.

26242 **FUTURE DIRECTIONS**

26243 None.

26244 **SEE ALSO**

26245 *mq\_close()*, *mq\_open()*, *msgctl()*, *msgget()*, *msgrcv()*, *msgsnd()*, the Base Definitions volume of |  
 26246 IEEE Std 1003.1-200x, <mqueue.h>

26247 **CHANGE HISTORY**

26248 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

26249 **Issue 6**26250 The *mq\_unlink()* function is marked as part of the Message Passing option.

26251 The Open Group Corrigendum U021/5 is applied, clarifying that upon unsuccessful completion, |  
 26252 the named message queue is unchanged by this function.

26253  
26254

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Message Passing option.

26255 **NAME**

26256 mrand48 — generate uniformly distributed pseudo-random signed long integers

26257 **SYNOPSIS**

```
26258 xSI #include <stdlib.h>
```

```
26259 long mrand48(void);
```

26260

26261 **DESCRIPTION**

26262 Refer to *drand48()*.

## 26263 NAME

26264 msgctl — XSI message control operations

## 26265 SYNOPSIS

26266 XSI 

```
#include <sys/msg.h>
```

26267 

```
int msgctl(int msqid, int cmd, struct msqid_ds *buf);
```

26268

## 26269 DESCRIPTION

26270 The *msgctl()* function operates on XSI message queues (see the Base Definitions volume of  
 26271 IEEE Std 1003.1-200x, Section 3.224, Message Queue). It is unspecified whether this function  
 26272 interoperates with the realtime interprocess communication facilities defined in Section 2.8 (on  
 26273 page 491).

26274 The *msgctl()* function shall provide message control operations as specified by *cmd*. The  
 26275 following values for *cmd*, and the message control operations they specify, are:

26276 IPC\_STAT Place the current value of each member of the **msqid\_ds** data structure  
 26277 associated with *msqid* into the structure pointed to by *buf*. The contents of this  
 26278 structure are defined in `<sys/msg.h>`.

26279 IPC\_SET Set the value of the following members of the **msqid\_ds** data structure  
 26280 associated with *msqid* to the corresponding value found in the structure  
 26281 pointed to by *buf*:

26282 msg\_perm.uid  
 26283 msg\_perm.gid  
 26284 msg\_perm.mode  
 26285 msg\_qbytes

26286 IPC\_SET can only be executed by a process with appropriate privileges or that  
 26287 has an effective user ID equal to the value of **msg\_perm.cuid** or  
 26288 **msg\_perm.uid** in the **msqid\_ds** data structure associated with *msqid*. Only a  
 26289 process with appropriate privileges can raise the value of **msg\_qbytes**.

26290 IPC\_RMID Remove the message queue identifier specified by *msqid* from the system and  
 26291 destroy the message queue and **msqid\_ds** data structure associated with it.  
 26292 IPC\_RMID can only be executed by a process with appropriate privileges or  
 26293 one that has an effective user ID equal to the value of **msg\_perm.cuid** or  
 26294 **msg\_perm.uid** in the **msqid\_ds** data structure associated with *msqid*.

## 26295 RETURN VALUE

26296 Upon successful completion, *msgctl()* shall return 0; otherwise, it shall return `-1` and set *errno* to  
 26297 indicate the error.

## 26298 ERRORS

26299 The *msgctl()* function shall fail if:

26300 [EACCES] The argument *cmd* is IPC\_STAT and the calling process does not have read  
 26301 permission; see Section 2.7 (on page 489).

26302 [EINVAL] The value of *msqid* is not a valid message queue identifier; or the value of *cmd*  
 26303 is not a valid command.

26304 [EPERM] The argument *cmd* is IPC\_RMID or IPC\_SET and the effective user ID of the  
 26305 calling process is not equal to that of a process with appropriate privileges  
 26306 and it is not equal to the value of **msg\_perm.cuid** or **msg\_perm.uid** in the data  
 26307 structure associated with *msqid*.

26308 [EPERM] The argument *cmd* is IPC\_SET, an attempt is being made to increase to the  
26309 value of **msg\_qbytes**, and the effective user ID of the calling process does not  
26310 have appropriate privileges.

26311 **EXAMPLES**

26312 None.

26313 **APPLICATION USAGE**

26314 The POSIX Realtime Extension defines alternative interfaces for interprocess communication  
26315 (IPC). Application developers who need to use IPC should design their applications so that  
26316 modules using the IPC routines described in Section 2.7 (on page 489) can be easily modified to  
26317 use the alternative interfaces.

26318 **RATIONALE**

26319 None.

26320 **FUTURE DIRECTIONS**

26321 None.

26322 **SEE ALSO**

26323 *mq\_close()*, *mq\_getattr()*, *mq\_notify()*, *mq\_open()*, *mq\_receive()*, *mq\_send()*, *mq\_setattr()*,  
26324 *mq\_unlink()*, *msgget()*, *msgrcv()*, *msgsnd()*, the Base Definitions volume of IEEE Std 1003.1-200x,  
26325 <sys/msg.h>, Section 2.7 (on page 489)

26326 **CHANGE HISTORY**

26327 First released in Issue 2. Derived from Issue 2 of the SVID.

26328 **Issue 5**

26329 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE  
26330 DIRECTIONS to a new APPLICATION USAGE section.

## 26331 NAME

26332 msgget — get the XSI message queue identifier

## 26333 SYNOPSIS

26334 XSI #include &lt;sys/msg.h&gt;

26335 int msgget(key\_t key, int msgflg);

26336

## 26337 DESCRIPTION

26338 The *msgget()* function operates on XSI message queues (see the Base Definitions volume of  
 26339 IEEE Std 1003.1-200x, Section 3.224, Message Queue). It is unspecified whether this function  
 26340 interoperates with the realtime interprocess communication facilities defined in Section 2.8 (on  
 26341 page 491).

26342 The *msgget()* function shall return the message queue identifier associated with the argument  
 26343 *key*.

26344 A message queue identifier, associated message queue, and data structure (see <sys/msg.h>),  
 26345 shall be created for the argument *key* if one of the following is true:

- 26346 • The argument *key* is equal to IPC\_PRIVATE.
- 26347 • The argument *key* does not already have a message queue identifier associated with it, and  
 26348 (*msgflg* & IPC\_CREAT) is non-zero.

26349 Upon creation, the data structure associated with the new message queue identifier shall be  
 26350 initialized as follows:

- 26351 • *msg\_perm.cuid*, *msg\_perm.uid*, *msg\_perm.cgid*, and *msg\_perm.gid* shall be set equal to the  
 26352 effective user ID and effective group ID, respectively, of the calling process.
- 26353 • The low-order 9 bits of *msg\_perm.mode* shall be set equal to the low-order 9 bits of *msgflg*.
- 26354 • *msg\_qnum*, *msg\_lspid*, *msg\_lrpid*, *msg\_stime*, and *msg\_rtime* shall be set equal to 0.
- 26355 • *msg\_ctime* shall be set equal to the current time.
- 26356 • *msg\_qbytes* shall be set equal to the system limit.

## 26357 RETURN VALUE

26358 Upon successful completion, *msgget()* shall return a non-negative integer, namely a message  
 26359 queue identifier. Otherwise, it shall return  $-1$  and set *errno* to indicate the error.

## 26360 ERRORS

26361 The *msgget()* function shall fail if:

- 26362 [EACCES] A message queue identifier exists for the argument *key*, but operation  
 26363 permission as specified by the low-order 9 bits of *msgflg* would not be granted;  
 26364 see Section 2.7 (on page 489).
- 26365 [EEXIST] A message queue identifier exists for the argument *key* but ((*msgflg* &  
 26366 IPC\_CREAT) && (*msgflg* & IPC\_EXCL)) is non-zero.
- 26367 [ENOENT] A message queue identifier does not exist for the argument *key* and (*msgflg* &  
 26368 IPC\_CREAT) is 0.
- 26369 [ENOSPC] A message queue identifier is to be created but the system-imposed limit on  
 26370 the maximum number of allowed message queue identifiers system-wide  
 26371 would be exceeded.

**26372 EXAMPLES**

26373 None.

**26374 APPLICATION USAGE**

26375 The POSIX Realtime Extension defines alternative interfaces for interprocess communication  
26376 (IPC). Application developers who need to use IPC should design their applications so that  
26377 modules using the IPC routines described in Section 2.7 (on page 489) can be easily modified to  
26378 use the alternative interfaces.

**26379 RATIONALE**

26380 None.

**26381 FUTURE DIRECTIONS**

26382 None.

**26383 SEE ALSO**

26384 *mq\_close()*, *mq\_getattr()*, *mq\_notify()*, *mq\_open()*, *mq\_receive()*, *mq\_send()*, *mq\_setattr()*,  
26385 *mq\_unlink()*, *msgctl()*, *msgrcv()*, *msgsnd()*, the Base Definitions volume of IEEE Std 1003.1-200x,  
26386 `<sys/msg.h>`, Section 2.7 (on page 489)

**26387 CHANGE HISTORY**

26388 First released in Issue 2. Derived from Issue 2 of the SVID.

**26389 Issue 5**

26390 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE  
26391 DIRECTIONS to a new APPLICATION USAGE section.

## 26392 NAME

26393 msgrcv — XSI message receive operation

## 26394 SYNOPSIS

26395 XSI 

```
#include <sys/msg.h>
```

```
26396 ssize_t msgrcv(int msqid, void *msgp, size_t msgsz, long msgtyp,
26397               int msgflg);
26398
```

## 26399 DESCRIPTION

26400 The *msgrcv()* function operates on XSI message queues (see the Base Definitions volume of  
 26401 IEEE Std 1003.1-200x, Section 3.224, Message Queue). It is unspecified whether this function  
 26402 interoperates with the realtime interprocess communication facilities defined in Section 2.8 (on  
 26403 page 491).

26404 The *msgrcv()* function shall read a message from the queue associated with the message queue  
 26405 identifier specified by *msqid* and place it in the user-defined buffer pointed to by *msgp*.

26406 The application shall ensure that the argument *msgp* points to a user-defined buffer that contains  
 26407 first a field of type **long** specifying the type of the message, and then a data portion that holds  
 26408 the data bytes of the message. The structure below is an example of what this user-defined  
 26409 buffer might look like:

```
26410 struct mymsg {
26411     long   mtype;      /* Message type. */
26412     char   mtext[1];  /* Message text. */
26413 }
```

26414 The structure member *mtype* is the received message's type as specified by the sending process.

26415 The structure member *mtext* is the text of the message.

26416 The argument *msgsz* specifies the size in bytes of *mtext*. The received message shall be truncated |  
 26417 to *msgsz* bytes if it is larger than *msgsz* and (*msgflg* & MSG\_NOERROR) is non-zero. The |  
 26418 truncated part of the message shall be lost and no indication of the truncation shall be given to |  
 26419 the calling process. |

26420 If the value of *msgsz* is greater than {SSIZE\_MAX}, the result is implementation-defined.

26421 The argument *msgtyp* specifies the type of message requested as follows:

- 26422 • If *msgtyp* is 0, the first message on the queue shall be received. |
- 26423 • If *msgtyp* is greater than 0, the first message of type *msgtyp* shall be received. |
- 26424 • If *msgtyp* is less than 0, the first message of the lowest type that is less than or equal to the  
 26425 absolute value of *msgtyp* shall be received. |

26426 The argument *msgflg* specifies the action to be taken if a message of the desired type is not on the  
 26427 queue. These are as follows:

- 26428 • If (*msgflg* & IPC\_NOWAIT) is non-zero, the calling thread shall return immediately with a  
 26429 return value of -1 and *errno* set to [ENOMSG].
- 26430 • If (*msgflg* & IPC\_NOWAIT) is 0, the calling thread shall suspend execution until one of the  
 26431 following occurs:
  - 26432 — A message of the desired type is placed on the queue.
  - 26433 — The message queue identifier *msqid* is removed from the system; when this occurs, *errno*  
 26434 shall be set equal to [EIDRM] and -1 shall be returned.

26435 — The calling thread receives a signal that is to be caught; in this case a message is not  
 26436 received and the calling thread resumes execution in the manner prescribed in *sigaction()*.

26437 Upon successful completion, the following actions are taken with respect to the data structure  
 26438 associated with *msqid*:

- 26439 • **msg\_qnum** shall be decremented by 1. |
- 26440 • **msg\_lrpid** shall be set equal to the process ID of the calling process. |
- 26441 • **msg\_rtime** shall be set equal to the current time. |

#### 26442 RETURN VALUE

26443 Upon successful completion, *msgrcv()* shall return a value equal to the number of bytes actually  
 26444 placed into the buffer *mtext*. Otherwise, no message shall be received, *msgrcv()* shall return  
 26445 (**ssize\_t**)−1, and *errno* shall be set to indicate the error.

#### 26446 ERRORS

26447 The *msgrcv()* function shall fail if:

- |       |          |   |
|-------|----------|---|
| 26448 | [E2BIG]  | The value of <i>mtext</i> is greater than <i>msgsz</i> and ( <i>msgflg</i> & MSG_NOERROR) is 0. |
| 26449 | [EACCES] | Operation permission is denied to the calling process; see Section 2.7 (on page                 |
| 26450 |          | 489).   |
| 26451 | [EIDRM]  | The message queue identifier <i>msqid</i> is removed from the system.                           |
| 26452 | [EINTR]  | The <i>msgrcv()</i> function was interrupted by a signal.                                       |
| 26453 | [EINVAL] | <i>msqid</i> is not a valid message queue identifier.   |
| 26454 | [ENOMSG] | The queue does not contain a message of the desired type and ( <i>msgflg</i> &                  |
| 26455 |          | IPC_NOWAIT) is non-zero.  |

#### 26456 EXAMPLES

##### 26457 Receiving a Message

26458 The following example receives the first message on the queue (based on the value of the *msgtyp*  
 26459 argument, 0). The queue is identified by the *msqid* argument (assuming that the value has  
 26460 previously been set). This call specifies that an error should be reported if no message is  
 26461 available, but not if the message is too large. The message size is calculated directly using the  
 26462 *sizeof* operator.

```

26463 #include <sys/msg.h>
26464 ...
26465 int result;
26466 int msqid;
26467 struct message {
26468     long type;
26469     char text[20];
26470 } msg;
26471 long msgtyp = 0;
26472 ...
26473 result = msgrcv(msqid, (void *) &msg, sizeof(msg.text),
26474               msgtyp, MSG_NOERROR | IPC_NOWAIT);
  
```

**26475 APPLICATION USAGE**

26476 The POSIX Realtime Extension defines alternative interfaces for interprocess communication  
26477 (IPC). Application developers who need to use IPC should design their applications so that  
26478 modules using the IPC routines described in Section 2.7 (on page 489) can be easily modified to  
26479 use the alternative interfaces.

**26480 RATIONALE**

26481 None.

**26482 FUTURE DIRECTIONS**

26483 None.

**26484 SEE ALSO**

26485 *mq\_close()*, *mq\_getattr()*, *mq\_notify()*, *mq\_open()*, *mq\_receive()*, *mq\_send()*, *mq\_setattr()*,  
26486 *mq\_unlink()*, *msgctl()*, *msgget()*, *msgsnd()*, *sigaction()*, the Base Definitions volume of  
26487 IEEE Std 1003.1-200x, <sys/msg.h>, Section 2.7 (on page 489)

**26488 CHANGE HISTORY**

26489 First released in Issue 2. Derived from Issue 2 of the SVID.

**26490 Issue 5**

26491 The type of the return value is changed from **int** to **ssize\_t**, and a warning is added to the  
26492 DESCRIPTION about values of *msgsz* larger than {SSIZE\_MAX}.

26493 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE  
26494 DIRECTIONS to the APPLICATION USAGE section.

**26495 Issue 6**

26496 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

26497 **NAME**

26498 msgsnd — XSI message send operation

26499 **SYNOPSIS**26500 XSI 

```
#include <sys/msg.h>
```

26501 

```
int msgsnd(int msqid, const void *msgp, size_t msgsz, int msgflg);
```

26502

26503 **DESCRIPTION**

26504 The *msgsnd()* function operates on XSI message queues (see the Base Definitions volume of  
 26505 IEEE Std 1003.1-200x, Section 3.224, Message Queue). It is unspecified whether this function  
 26506 interoperates with the realtime interprocess communication facilities defined in Section 2.8 (on  
 26507 page 491).

26508 The *msgsnd()* function shall send a message to the queue associated with the message queue  
 26509 identifier specified by *msqid*.

26510 The application shall ensure that the argument *msgp* points to a user-defined buffer that contains  
 26511 first a field of type **long** specifying the type of the message, and then a data portion that holds  
 26512 the data bytes of the message. The structure below is an example of what this user-defined  
 26513 buffer might look like:

```
26514 struct mymsg {
26515     long    mtype;        /* Message type. */
26516     char    mtext[1];    /* Message text. */
26517 }
```

26518 The structure member *mtype* is a non-zero positive type **long** that can be used by the receiving  
 26519 process for message selection.

26520 The structure member *mtext* is any text of length *msgsz* bytes. The argument *msgsz* can range  
 26521 from 0 to a system-imposed maximum.

26522 The argument *msgflg* specifies the action to be taken if one or more of the following are true:

- 26523 • The number of bytes already on the queue is equal to **msg\_qbytes**; see `<sys/msg.h>`.
- 26524 • The total number of messages on all queues system-wide is equal to the system-imposed  
 26525 limit.

26526 These actions are as follows:

- 26527 • If (*msgflg* & `IPC_NOWAIT`) is non-zero, the message shall not be sent and the calling thread  
 26528 shall return immediately.
- 26529 • If (*msgflg* & `IPC_NOWAIT`) is 0, the calling thread shall suspend execution until one of the  
 26530 following occurs:
  - 26531 — The condition responsible for the suspension no longer exists, in which case the message  
 26532 is sent.
  - 26533 — The message queue identifier *msqid* is removed from the system; when this occurs, *errno*  
 26534 shall be set equal to `[EIDRM]` and `-1` shall be returned.
  - 26535 — The calling thread receives a signal that is to be caught; in this case the message is not  
 26536 sent and the calling thread resumes execution in the manner prescribed in *sigaction()*.

26537 Upon successful completion, the following actions are taken with respect to the data structure  
 26538 associated with *msqid*; see `<sys/msg.h>`:

- 26539           • **msg\_qnum** shall be incremented by 1. |
- 26540           • **msg\_lspid** shall be set equal to the process ID of the calling process. |
- 26541           • **msg\_stime** shall be set equal to the current time. |

#### 26542 RETURN VALUE

26543           Upon successful completion, *msgsnd()* shall return 0; otherwise, no message shall be sent,  
26544           *msgsnd()* shall return -1, and *errno* shall be set to indicate the error.

#### 26545 ERRORS

26546           The *msgsnd()* function shall fail if:

- 26547           [EACCES]           Operation permission is denied to the calling process; see Section 2.7 (on page  
26548                               489).
- 26549           [EAGAIN]           The message cannot be sent for one of the reasons cited above and (*msgflg* &  
26550                               IPC\_NOWAIT) is non-zero.
- 26551           [EIDRM]           The message queue identifier *msqid* is removed from the system.
- 26552           [EINTR]           The *msgsnd()* function was interrupted by a signal.
- 26553           [EINVAL]           The value of *msqid* is not a valid message queue identifier, or the value of  
26554                               *mtype* is less than 1; or the value of *msgsz* is less than 0 or greater than the  
26555                               system-imposed limit.

#### 26556 EXAMPLES

##### 26557           **Sending a Message**

26558           The following example sends a message to the queue identified by the *msqid* argument  
26559           (assuming that value has previously been set). This call specifies that an error should be  
26560           reported if no message is available. The message size is calculated directly using the *sizeof*  
26561           operator.

```
26562           #include <sys/msg.h>
26563           ...
26564           int result;
26565           int msqid;
26566           struct message {
26567               long type;
26568               char text[20];
26569           } msg;

26570           msg.type = 1;
26571           strcpy(msg.text, "This is message 1");
26572           ...
26573           result = msgsnd(msqid, (void *) &msg, sizeof(msg.text), IPC_NOWAIT);
```

#### 26574 APPLICATION USAGE

26575           The POSIX Realtime Extension defines alternative interfaces for interprocess communication  
26576           (IPC). Application developers who need to use IPC should design their applications so that  
26577           modules using the IPC routines described in Section 2.7 (on page 489) can be easily modified to  
26578           use the alternative interfaces.

**26579 RATIONALE**

26580 None.

**26581 FUTURE DIRECTIONS**

26582 None.

**26583 SEE ALSO**

26584 *mq\_close()*, *mq\_getattr()*, *mq\_notify()*, *mq\_open()*, *mq\_receive()*, *mq\_send()*, *mq\_setattr()*,  
26585 *mq\_unlink()*, *msgctl()*, *msgget()*, *msgrcv()*, *sigaction()*, the Base Definitions volume of  
26586 IEEE Std 1003.1-200x, <sys/msg.h>, Section 2.7 (on page 489)

**26587 CHANGE HISTORY**

26588 First released in Issue 2. Derived from Issue 2 of the SVID.

**26589 Issue 5**

26590 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE  
26591 DIRECTIONS to a new APPLICATION USAGE section.

**26592 Issue 6**

26593 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

26594 **NAME**

26595       msync — synchronize memory with physical storage

26596 **SYNOPSIS**

26597 MF SIO   #include &lt;sys/mman.h&gt;

26598       int msync(void \*addr, size\_t len, int flags);

26599

26600 **DESCRIPTION**

26601       The *msync()* function shall write all modified data to permanent storage locations, if any, in  
 26602       those whole pages containing any part of the address space of the process starting at address  
 26603       *addr* and continuing for *len* bytes. If no such storage exists, *msync()* need not have any effect. If  
 26604       requested, the *msync()* function shall then invalidate cached copies of data.

26605       The implementation shall require that *addr* be a multiple of the page size as returned by  
 26606       *sysconf()*.

26607       For mappings to files, the *msync()* function shall ensure that all write operations are completed  
 26608       as defined for synchronized I/O data integrity completion. It is unspecified whether the  
 26609       implementation also writes out other file attributes. When the *msync()* function is called on  
 26610       MAP\_PRIVATE mappings, any modified data shall not be written to the underlying object and  
 26611       shall not cause such data to be made visible to other processes. It is unspecified whether data in  
 26612       SHM|TYM MAP\_PRIVATE mappings has any permanent storage locations. The effect of *msync()* on a  
 26613       shared memory object or a typed memory object is unspecified. The behavior of this function is  
 26614       unspecified if the mapping was not established by a call to *mmap()*.

26615       The *flags* argument is constructed from the bitwise-inclusive OR of one or more of the following  
 26616       flags defined in the <sys/mman.h> header:

26617

26618

26619

26620

26621

Symbolic Constant	Description
MS_ASYNC	Perform asynchronous writes.
MS_SYNC	Perform synchronous writes.
MS_INVALIDATE	Invalidate cached data.

26622       When MS\_ASYNC is specified, *msync()* shall return immediately once all the write operations  
 26623       are initiated or queued for servicing; when MS\_SYNC is specified, *msync()* shall not return until  
 26624       all write operations are completed as defined for synchronized I/O data integrity completion.  
 26625       Either MS\_ASYNC or MS\_SYNC is specified, but not both.

26626       When MS\_INVALIDATE is specified, *msync()* shall invalidate all cached copies of mapped data  
 26627       that are inconsistent with the permanent storage locations such that subsequent references shall  
 26628       obtain data that was consistent with the permanent storage locations sometime between the call  
 26629       to *msync()* and the first subsequent memory reference to the data.

26630       If *msync()* causes any write to a file, the file's *st\_ctime* and *st\_mtime* fields shall be marked for  
 26631       update.

26632 **RETURN VALUE**

26633       Upon successful completion, *msync()* shall return 0; otherwise, it shall return -1 and set *errno* to  
 26634       indicate the error.

26635 **ERRORS**26636       The *msync()* function shall fail if:

26637       [EBUSY]       Some or all of the addresses in the range starting at *addr* and continuing for *len*  
 26638       bytes are locked, and MS\_INVALIDATE is specified.

- 26639 [EINVAL] The value of *flags* is invalid.
- 26640 [EINVAL] The value of *addr* is not a multiple of the page size, {PAGESIZE}.
- 26641 [ENOMEM] The addresses in the range starting at *addr* and continuing for *len* bytes are outside the range allowed for the address space of a process or specify one or more pages that are not mapped.
- 26642
- 26643

26644 **EXAMPLES**

26645 None.

26646 **APPLICATION USAGE**

26647 The *msync()* function is only supported if the Memory Mapped Files option and the Synchronized Input and Output option are supported, and thus need not be available on all implementations.

26648

26649

26650 The *msync()* function should be used by programs that require a memory object to be in a known state; for example, in building transaction facilities.

26651

26652 Normal system activity can cause pages to be written to disk. Therefore, there are no guarantees that *msync()* is the only control over when pages are or are not written to disk.

26653

26654 **RATIONALE**

26655 The *msync()* function writes out data in a mapped region to the permanent storage for the underlying object. The call to *msync()* ensures data integrity of the file.

26656

26657 After the data is written out, any cached data may be invalidated if the MS\_INVALIDATE flag was specified. This is useful on systems that do not support read/write consistency.

26658

26659 **FUTURE DIRECTIONS**

26660 None.

26661 **SEE ALSO**26662 *mmap()*, *sysconf()*, the Base Definitions volume of IEEE Std 1003.1-200x, <sys/mman.h>26663 **CHANGE HISTORY**

26664 First released in Issue 4, Version 2.

26665 **Issue 5**

26666 Moved from X/OPEN UNIX extension to BASE.

26667 Aligned with *msync()* in the POSIX Realtime Extension as follows:

- 26668
- The DESCRIPTION is extensively reworded.
  - [EBUSY] and a new form of [EINVAL] are added as mandatory error conditions.
- 26669

26670 **Issue 6**

26671 The *msync()* function is marked as part of the Memory Mapped Files and Synchronized Input and Output options.

26672

26673 The following changes are made for alignment with the ISO POSIX-1: 1996 standard:

- 26674
- The [EBUSY] mandatory error condition is added.

26675 The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

26676

- 26677
- The DESCRIPTION is updated to state that implementations require *addr* to be a multiple of the page size.
  - The second [EINVAL] error condition is made mandatory.
- 26678
- 26679

26680  
26681

The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by adding reference to typed memory objects.

26682 **NAME**

26683           munlock — unlock a range of process address space

26684 **SYNOPSIS**

26685 MLR       #include &lt;sys/mman.h&gt;

26686           int munlock(const void \*addr, size\_t len);

26687

26688 **DESCRIPTION**26689           Refer to *mlock()*.

26690 **NAME**

26691           munlockall — unlock the address space of a process

26692 **SYNOPSIS**

26693 ML       #include <sys/mman.h>

26694       int munlockall(void);

26695

26696 **DESCRIPTION**

26697       Refer to *mlockall()*.

26698 **NAME**

26699           munmap — unmap pages of memory

26700 **SYNOPSIS**

26701 MF|SHM   #include &lt;sys/mman.h&gt;

26702           int munmap(void \*addr, size\_t len);

26703

26704 **DESCRIPTION**

26705           The *munmap()* function shall remove any mappings for those entire pages containing any part of  
 26706           the address space of the process starting at *addr* and continuing for *len* bytes. Further references |  
 26707           to these pages shall result in the generation of a SIGSEGV signal to the process. If there are no |  
 26708           mappings in the specified address range, then *munmap()* has no effect.

26709           The implementation shall require that *addr* be a multiple of the page size {PAGESIZE}.

26710           If a mapping to be removed was private, any modifications made in this address range shall be  
 26711           discarded.

26712 ML|MLR   Any memory locks (see *mlock()* and *mlockall()*) associated with this address range shall be  
 26713           removed, as if by an appropriate call to *munlock()*.

26714 TYM      If a mapping removed from a typed memory object causes the corresponding address range of  
 26715           the memory pool to be inaccessible by any process in the system except through allocatable  
 26716           mappings (that is, mappings of typed memory objects opened with the  
 26717           POSIX\_TYPED\_MEM\_MAP\_ALLOCATABLE flag), then that range of the memory pool shall  
 26718           become deallocated and may become available to satisfy future typed memory allocation  
 26719           requests.

26720           A mapping removed from a typed memory object opened with the  
 26721           POSIX\_TYPED\_MEM\_MAP\_ALLOCATABLE flag shall not affect in any way the availability of  
 26722           that typed memory for allocation.

26723           The behavior of this function is unspecified if the mapping was not established by a call to  
 26724           *mmap()*.

26725 **RETURN VALUE**

26726           Upon successful completion, *munmap()* shall return 0; otherwise, it shall return -1 and set *errno*  
 26727           to indicate the error.

26728 **ERRORS**

26729           The *munmap()* function shall fail if:

26730           [EINVAL]       Addresses in the range [*addr*,*addr+len*) are outside the valid range for the  
 26731           address space of a process.

26732           [EINVAL]       The *len* argument is 0.

26733           [EINVAL]       The *addr* argument is not a multiple of the page size as returned by *sysconf()*.

26734 **EXAMPLES**

26735 None.

26736 **APPLICATION USAGE**

26737 The *munmap()* function is only supported if the Memory Mapped Files option or the Shared  
26738 Memory Objects option is supported.

26739 **RATIONALE**26740 The *munmap()* function corresponds to SVR4, just as the *mmap()* function does.

26741 It is possible that an application has applied process memory locking to a region that contains  
26742 shared memory. If this has occurred, the *munmap()* call ignores those locks and, if necessary,  
26743 causes those locks to be removed.

26744 **FUTURE DIRECTIONS**

26745 None.

26746 **SEE ALSO**

26747 *mlock()*, *mlockall()*, *mmap()*, *posix\_typed\_mem\_open()*, *sysconf()*, the Base Definitions volume of  
26748 IEEE Std 1003.1-200x, <signal.h>, <sys/mman.h>

26749 **CHANGE HISTORY**

26750 First released in Issue 4, Version 2.

26751 **Issue 5**

26752 Moved from X/OPEN UNIX extension to BASE.

26753 Aligned with *munmap()* in the POSIX Realtime Extension as follows:

- 26754 • The DESCRIPTION is extensively reworded.
- 26755 • The SIGBUS error is no longer permitted to be generated.

26756 **Issue 6**

26757 The *munmap()* function is marked as part of the Memory Mapped Files and Shared Memory  
26758 Objects option.

26759 The following new requirements on POSIX implementations derive from alignment with the  
26760 Single UNIX Specification:

- 26761 • The DESCRIPTION is updated to state that implementations require *addr* to be a multiple of  
26762 the page size.
- 26763 • The [EINVAL] error conditions are added.

26764 The following changes are made for alignment with IEEE Std 1003.1j-2000:

- 26765 • Semantics for typed memory objects are added to the DESCRIPTION.
- 26766 • The *posix\_typed\_mem\_open()* function is added to the SEE ALSO section.

26767 **NAME**

26768 nan, nanf, nanl — return quiet NaN

26769 **SYNOPSIS**

26770 #include <math.h>

26771 double nan(const char \*tagp);

26772 float nanf(const char \*tagp);

26773 long double nanl(const char \*tagp);

26774 **DESCRIPTION**

26775 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
26776 conflict between the requirements described here and the ISO C standard is unintentional. This  
26777 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

26778 The function call *nan*("n-char-sequence") shall be equivalent to:

26779 strtod("NAN(n-char-sequence)", (char \*\*) NULL);

26780 The function call *nan*("") shall be equivalent to:

26781 strtod("NAN()", (char \*\*) NULL)

26782 If *tagp* does not point to an *n-char* sequence or an empty string, the function call shall be  
26783 equivalent to:

26784 strtod("NAN", (char \*\*) NULL)

26785 Function calls to *nanf*() and *nanl*() are equivalent to the corresponding function calls to *strtof*()  
26786 and *strtold*().

26787 **RETURN VALUE**

26788 These functions shall return a quiet NaN, if available, with content indicated through *tagp*.

26789 If the implementation does not support quiet NaNs, these functions shall return zero.

26790 **ERRORS**

26791 No errors are defined.

26792 **EXAMPLES**

26793 None.

26794 **APPLICATION USAGE**

26795 None.

26796 **RATIONALE**

26797 None.

26798 **FUTURE DIRECTIONS**

26799 None.

26800 **SEE ALSO**

26801 *strtod*(), *strtold*(), the Base Definitions volume of IEEE Std 1003.1-200x, <math.h>

26802 **CHANGE HISTORY**

26803 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

26804 **NAME**26805 nanosleep — high resolution sleep (**REALTIME**)26806 **SYNOPSIS**26807 TMR `#include <time.h>`26808 `int nanosleep(const struct timespec *rqtp, struct timespec *rmtp);`

26809

26810 **DESCRIPTION**26811 The *nanosleep()* function shall cause the current thread to be suspended from execution until  
26812 either the time interval specified by the *rqtp* argument has elapsed or a signal is delivered to the  
26813 calling thread, and its action is to invoke a signal-catching function or to terminate the process.26814 The suspension time may be longer than requested because the argument value is rounded up to  
26815 an integer multiple of the sleep resolution or because of the scheduling of other activity by the  
26816 system. But, except for the case of being interrupted by a signal, the suspension time shall not be  
26817 less than the time specified by *rqtp*, as measured by the system clock, **CLOCK\_REALTIME**.26818 The use of the *nanosleep()* function has no effect on the action or blockage of any signal.26819 **RETURN VALUE**26820 If the *nanosleep()* function returns because the requested time has elapsed, its return value shall  
26821 be zero.26822 If the *nanosleep()* function returns because it has been interrupted by a signal, it shall return a  
26823 value of `-1` and set *errno* to indicate the interruption. If the *rmtp* argument is non-NULL, the  
26824 **timespec** structure referenced by it is updated to contain the amount of time remaining in the  
26825 interval (the requested time minus the time actually slept). If the *rmtp* argument is NULL, the  
26826 remaining time is not returned.26827 If *nanosleep()* fails, it shall return a value of `-1` and set *errno* to indicate the error.26828 **ERRORS**26829 The *nanosleep()* function shall fail if:26830 [EINTR] The *nanosleep()* function was interrupted by a signal.26831 [EINVAL] The *rqtp* argument specified a nanosecond value less than zero or greater than  
26832 or equal to 1000 million.26833 **EXAMPLES**

26834 None.

26835 **APPLICATION USAGE**

26836 None.

26837 **RATIONALE**26838 It is common to suspend execution of a process for an interval in order to poll the status of a  
26839 non-interrupting function. A large number of actual needs can be met with a simple extension to  
26840 *sleep()* that provides finer resolution.26841 In the POSIX.1-1990 standard and SVR4, it is possible to implement such a routine, but the  
26842 frequency of wakeup is limited by the resolution of the *alarm()* and *sleep()* functions. In 4.3 BSD,  
26843 it is possible to write such a routine using no static storage and reserving no system facilities.  
26844 Although it is possible to write a function with similar functionality to *sleep()* using the  
26845 remainder of the timers function, such a function requires the use of signals and the reservation  
26846 of some signal number. This volume of IEEE Std 1003.1-200x requires that *nanosleep()* be non-  
26847 intrusive of the signals function.

26848           The *nanosleep()* function shall return a value of 0 on success and –1 on failure or if interrupted.  
26849           This latter case is different from *sleep()*. This was done because the remaining time is returned  
26850           via an argument structure pointer, *rmtp*, instead of as the return value.

26851 **FUTURE DIRECTIONS**

26852           None.

26853 **SEE ALSO**

26854           *sleep()*, the Base Definitions volume of IEEE Std 1003.1-200x, <**time.h**>

26855 **CHANGE HISTORY**

26856           First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

26857 **Issue 6**

26858           The *nanosleep()* function is marked as part of the Timers option.

26859           The [ENOSYS] error condition has been removed as stubs need not be provided if an  
26860           implementation does not support the Timers option.

26861 **NAME**

26862 nearbyint, nearbyintf, nearbyintl — floating-point rounding functions

26863 **SYNOPSIS**

26864 #include &lt;math.h&gt;

26865 double nearbyint(double x);

26866 float nearbyintf(float x);

26867 long double nearbyintl(long double x);

26868 **DESCRIPTION**

26869 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 26870 conflict between the requirements described here and the ISO C standard is unintentional. This  
 26871 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

26872 These functions shall round their argument to an integer value in floating-point format, using  
 26873 the current rounding direction and without raising the inexact floating-point exception.

26874 An application wishing to check for error situations should set *errno* to zero and call  
 26875 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 26876 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 26877 zero, an error has occurred.

26878 **RETURN VALUE**

26879 Upon successful completion, these functions shall return the rounded integer value.

26880 MX If *x* is NaN, a NaN shall be returned.26881 If *x* is  $\pm 0$ ,  $\pm 0$  shall be returned.26882 If *x* is  $\pm\text{Inf}$ , *x* shall be returned.

26883 XSI If the correct value would cause overflow, a range error shall occur and *nearbyint*(), *nearbyintf*(),  
 26884 and *nearbyintl*() shall return the value of the macro HUGE\_VAL, HUGE\_VALF, and  
 26885 HUGE\_VALL, respectively.

26886 **ERRORS**

26887 These functions shall fail if:

26888 XSI **Range Error** The result would cause an overflow.

26889 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero, |  
 26890 then *errno* shall be set to [ERANGE]. If the integer expression |  
 26891 (math\_errhandling & MATH\_ERREXCEPT) is non-zero, then the overflow |  
 26892 floating-point exception shall be raised. |

26893 **EXAMPLES**

26894 None.

26895 **APPLICATION USAGE**

26896 On error, the expressions (math\_errhandling & MATH\_ERRNO) and (math\_errhandling &  
 26897 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

26898 **RATIONALE**

26899 None.

26900 **FUTURE DIRECTIONS**

26901 None.

26902 **SEE ALSO**

26903            *feclearexcept()*, *fetestexcept()*, the Base Definitions volume of IEEE Std 1003.1-200x, Section 4.18, |  
26904            Treatment of Error Conditions for Mathematical Functions, <math.h> |

26905 **CHANGE HISTORY**

26906            First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

## 26907 NAME

26908 nextafter, nextafterf, nextafterl, nexttoward, nexttowardf, nexttowardl — next representable  
26909 floating-point number

## 26910 SYNOPSIS

```
26911 #include <math.h>

26912 double nextafter(double x, double y);
26913 float nextafterf(float x, float y);
26914 long double nextafterl(long double x, long double y);
26915 double nexttoward(double x, long double y);
26916 float nexttowardf(float x, long double y);
26917 long double nexttowardl(long double x, long double y);
```

## 26918 DESCRIPTION

26919 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
26920 conflict between the requirements described here and the ISO C standard is unintentional. This  
26921 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

26922 The *nextafter()*, *nextafterf()*, and *nextafterl()* functions shall compute the next representable  
26923 floating-point value following *x* in the direction of *y*. Thus, if *y* is less than *x*, *nextafter()* shall  
26924 return the largest representable floating-point number less than *x*. The *nextafter()*, *nextafterf()*,  
26925 and *nextafterl()* functions shall return *y* if *x* equals *y*.

26926 The *nexttoward()*, *nexttowardf()*, and *nexttowardl()* functions shall be equivalent to the |  
26927 corresponding *nextafter()* functions, except that the second parameter shall have type **long** |  
26928 **double** and the functions shall return *y* converted to the type of the function if *x* equals *y*. |

26929 An application wishing to check for error situations should set *errno* to zero and call  
26930 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
26931 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
26932 zero, an error has occurred.

## 26933 RETURN VALUE

26934 Upon successful completion, these functions shall return the next representable floating-point  
26935 value following *x* in the direction of *y*.

26936 If *x*==*y*, *y* (of the type *x*) shall be returned.

26937 If *x* is finite and the correct function value would overflow, a range error shall occur and  
26938 ±HUGE\_VAL, ±HUGE\_VALF, and ±HUGE\_VALL (with the same sign as *x*) shall be returned as  
26939 appropriate for the return type of the function.

26940 MX If *x* or *y* is NaN, a NaN shall be returned.

26941 If *x*!=*y* and the correct function value is subnormal, zero, or underflows, a range error shall  
26942 occur, and either the correct function value (if representable) or 0.0 shall be returned.

## 26943 ERRORS

26944 These functions shall fail if:

26945 Range Error The correct value overflows

26946 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero, |  
26947 then *errno* shall be set to [ERANGE]. If the integer expression |  
26948 (math\_errhandling & MATH\_ERREXCEPT) is non-zero, then the overflow |  
26949 floating-point exception shall be raised. |

26950 MX Range Error The correct value is subnormal or underflows

26951 If the integer expression (`math_errhandling & MATH_ERRNO`) is non-zero, |  
26952 then *errno* shall be set to [ERANGE]. If the integer expression |  
26953 (`math_errhandling & MATH_ERREXCEPT`) is non-zero, then the underflow |  
26954 floating-point exception shall be raised. |

**26955 EXAMPLES**

26956 None.

**26957 APPLICATION USAGE**

26958 On error, the expressions (`math_errhandling & MATH_ERRNO`) and (`math_errhandling &`  
26959 `MATH_ERREXCEPT`) are independent of each other, but at least one of them must be non-zero.

**26960 RATIONALE**

26961 None.

**26962 FUTURE DIRECTIONS**

26963 None.

**26964 SEE ALSO**

26965 *feclearexcept()*, *fetestexcept()*, the Base Definitions volume of IEEE Std 1003.1-200x, Section 4.18, |  
26966 Treatment of Error Conditions for Mathematical Functions, <**math.h**> |

**26967 CHANGE HISTORY**

26968 First released in Issue 4, Version 2.

**26969 Issue 5**

26970 Moved from X/OPEN UNIX extension to BASE.

**26971 Issue 6**

26972 The *nextafter()* function is no longer marked as an extension.

26973 The *nextafterf()*, *nextafterl()*, *nexttoward()*, *nexttowardf()*, *nexttowardl()* functions are added for  
26974 alignment with the ISO/IEC 9899:1999 standard.

26975 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
26976 revised to align with the ISO/IEC 9899:1999 standard.

26977 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
26978 marked.

26979 **NAME**

26980           nexttoward, nexttowardf, nexttowardl — next representable floating-point number

26981 **SYNOPSIS**

26982           #include &lt;math.h&gt;

26983           double nexttoward(double *x*, long double *y*);26984           float nexttowardf(float *x*, long double *y*);26985           long double nexttowardl(long double *x*, long double *y*);26986 **DESCRIPTION**26987           Refer to *nextafter()*.

## 26988 NAME

26989 nftw — walk a file tree

## 26990 SYNOPSIS

26991 xSI #include &lt;ftw.h&gt;

```
26992 int nftw(const char *path, int (*fn)(const char *,
26993     const struct stat *, int, struct FTW *), int depth, int flags);
26994
```

## 26995 DESCRIPTION

26996 The *nftw()* function shall recursively descend the directory hierarchy rooted in *path*. The *nftw()*  
 26997 function has a similar effect to *ftw()* except that it takes an additional argument *flags*, which is a  
 26998 bitwise-inclusive OR of zero or more of the following flags:

26999 FTW\_CHDIR If set, *nftw()* shall change the current working directory to each directory as it  
 27000 reports files in that directory. If clear, *nftw()* shall not change the current  
 27001 working directory.

27002 FTW\_DEPTH If set, *nftw()* shall report all files in a directory before reporting the directory  
 27003 itself. If clear, *nftw()* shall report any directory before reporting the files in that  
 27004 directory.

27005 FTW\_MOUNT If set, *nftw()* shall only report files in the same file system as *path*. If clear,  
 27006 *nftw()* shall report all files encountered during the walk.

27007 FTW\_PHYS If set, *nftw()* shall perform a physical walk and shall not follow symbolic links.

27008 If FTW\_PHYS is clear and FTW\_DEPTH is set, *nftw()* shall follow links instead of reporting  
 27009 them, but shall not report any directory that would be a descendant of itself. If FTW\_PHYS is  
 27010 clear and FTW\_DEPTH is clear, *nftw()* shall follow links instead of reporting them, but shall not  
 27011 report the contents of any directory that would be a descendant of itself.

27012 At each file it encounters, *nftw()* shall call the user-supplied function *fn* with four arguments:

- 27013 • The first argument is the pathname of the object.
- 27014 • The second argument is a pointer to the **stat** buffer containing information on the object.
- 27015 • The third argument is an integer giving additional information. Its value is one of the  
 27016 following:

27017 FTW\_F The object is a file.

27018 FTW\_D The object is a directory.

27019 FTW\_DP The object is a directory and subdirectories have been visited. (This condition  
 27020 shall only occur if the FTW\_DEPTH flag is included in *flags*.)

27021 FTW\_SL The object is a symbolic link. (This condition shall only occur if the FTW\_PHYS  
 27022 flag is included in *flags*.)

27023 FTW\_SLN The object is a symbolic link that does not name an existing file. (This  
 27024 condition shall only occur if the FTW\_PHYS flag is not included in *flags*.)

27025 FTW\_DNR The object is a directory that cannot be read. The *fn* function shall not be called  
 27026 for any of its descendants.

27027 FTW\_NS The *stat()* function failed on the object because of lack of appropriate  
 27028 permission. The **stat** buffer passed to *fn* is undefined. Failure of *stat()* for any  
 27029 other reason is considered an error and *nftw()* shall return  $-1$ .

27030 • The fourth argument is a pointer to an **FTW** structure. The value of **base** is the offset of the  
 27031 object's filename in the pathname passed as the first argument to *fn*. The value of **level**  
 27032 indicates depth relative to the root of the walk, where the root level is 0.

27033 The results are unspecified if the application-supplied *fn* function does not preserve the current  
 27034 working directory.

27035 The argument *depth* sets the maximum number of file descriptors that are shall be used by *nftw()*  
 27036 while traversing the file tree. At most one file descriptor shall be used for each directory level.

27037 The *nftw()* function need not be reentrant. A function that is not required to be reentrant is not  
 27038 required to be thread-safe.

#### 27039 RETURN VALUE

27040 The *nftw()* function shall continue until the first of the following conditions occurs:

- 27041 • An invocation of *fn* shall return a non-zero value, in which case *nftw()* shall return that value.
- 27042 • The *nftw()* function detects an error other than [EACCES] (see FTW\_DNR and FTW\_NS  
 27043 above), in which case *nftw()* shall return  $-1$  and set *errno* to indicate the error.
- 27044 • The tree is exhausted, in which case *nftw()* shall return 0.

#### 27045 ERRORS

27046 The *nftw()* function shall fail if:

27047 [EACCES] Search permission is denied for any component of *path* or read permission is  
 27048 denied for *path*, or *fn* returns  $-1$  and does not reset *errno*.

27049 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*  
 27050 argument.

27051 [ENAMETOOLONG]

27052 The length of the *path* argument exceeds {PATH\_MAX} or a pathname  
 27053 component is longer than {NAME\_MAX}.

27054 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

27055 [ENOTDIR] A component of *path* is not a directory.

27056 [EOVERFLOW] A field in the **stat** structure cannot be represented correctly in the current  
 27057 programming environment for one or more files found in the file hierarchy.

27058 The *nftw()* function may fail if:

27059 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
 27060 resolution of the *path* argument.

27061 [EMFILE] {OPEN\_MAX} file descriptors are currently open in the calling process.

27062 [ENAMETOOLONG]

27063 Pathname resolution of a symbolic link produced an intermediate result  
 27064 whose length exceeds {PATH\_MAX}.

27065 [ENFILE] Too many files are currently open in the system.

27066 In addition, *errno* may be set if the function pointed by *fn* causes *errno* to be set.

27067 **EXAMPLES**

27068 The following example walks the `/tmp` directory and its subdirectories, calling the `nftw()`  
 27069 function for every directory entry, to a maximum of 5 levels deep.

```
27070 #include <ftw.h>
27071 ...
27072 int nftwfunc(const char *, const struct stat *, int, struct FTW *);
27073
27074 int nftwfunc(const char *filename, const struct stat *statptr,
27075             int fileflags, struct FTW *pftw)
27076 {
27077     return 0;
27078 }
27079 ...
27080 char *startpath = "/tmp";
27081 int depth = 5;
27082 int flags = FTW_CHDIR | FTW_DEPTH | FTW_MOUNT;
27083 int ret;
27084
27085 ret = nftw(startpath, nftwfunc, depth, flags);
```

27084 **APPLICATION USAGE**

27085 None.

27086 **RATIONALE**

27087 None.

27088 **FUTURE DIRECTIONS**

27089 None.

27090 **SEE ALSO**

27091 `lstat()`, `opendir()`, `readdir()`, `stat()`, the Base Definitions volume of IEEE Std 1003.1-200x, `<ftw.h>`

27092 **CHANGE HISTORY**

27093 First released in Issue 4, Version 2.

27094 **Issue 5**

27095 Moved from X/OPEN UNIX extension to BASE.

27096 In the DESCRIPTION, the definition of the *depth* argument is clarified.

27097 **Issue 6**

27098 The Open Group Base Resolution bwg97-003 is applied.

27099 The ERRORS section is updated as follows:

- 27100 • The wording of the mandatory [ELOOP] error condition is updated. |
- 27101 • A second optional [ELOOP] error condition is added. |
- 27102 • The [EOVERFLOW] mandatory error condition is added. |

27103 Text is added to the DESCRIPTION to say that the `nftw()` function need not be reentrant and |  
 27104 that the results are unspecified if the application-supplied *fn* function does not preserve the |  
 27105 current working directory. |

27106 **NAME**

27107 nice — change the nice value of a process

27108 **SYNOPSIS**27109 XSI `#include <unistd.h>`27110 `int nice(int incr);`

27111

27112 **DESCRIPTION**

27113 The *nice()* function shall add the value of *incr* to the nice value of the calling process. A process' |  
 27114 nice value is a non-negative number for which a more positive value shall result in less favorable |  
 27115 scheduling.

27116 A maximum nice value of  $2^{\{NZERO\}}-1$  and a minimum nice value of 0 shall be imposed by the |  
 27117 system. Requests for values above or below these limits shall result in the nice value being set to |  
 27118 the corresponding limit. Only a process with appropriate privileges can lower the nice value.

27119 PS|TPS Calling the *nice()* function has no effect on the priority of processes or threads with policy |  
 27120 SCHED\_FIFO or SCHED\_RR. The effect on processes or threads with other scheduling policies |  
 27121 is implementation-defined.

27122 The nice value set with *nice()* shall be applied to the process. If the process is multi-threaded, the |  
 27123 nice value shall affect all system scope threads in the process.

27124 As  $-1$  is a permissible return value in a successful situation, an application wishing to check for |  
 27125 error situations should set *errno* to 0, then call *nice()*, and if it returns  $-1$ , check to see whether |  
 27126 *errno* is non-zero.

27127 **RETURN VALUE**

27128 Upon successful completion, *nice()* shall return the new nice value  $-\{NZERO\}$ . Otherwise,  $-1$  |  
 27129 shall be returned, the process' nice value shall not be changed, and *errno* shall be set to indicate |  
 27130 the error.

27131 **ERRORS**27132 The *nice()* function shall fail if:

27133 [EPERM] The *incr* argument is negative and the calling process does not have |  
 27134 appropriate privileges.

27135 **EXAMPLES**27136 **Changing the Nice Value**

27137 The following example adds the value of the *incr* argument,  $-20$ , to the nice value of the calling |  
 27138 process.

27139 `#include <unistd.h>`27140 `...`27141 `int incr = -20;`27142 `int ret;`27143 `ret = nice(incr);`27144 **APPLICATION USAGE**

27145 None.

27146 **RATIONALE**

27147           None.

27148 **FUTURE DIRECTIONS**

27149           None.

27150 **SEE ALSO**27151           *getpriority()*, *setpriority()*, the Base Definitions volume of IEEE Std 1003.1-200x, <**limits.h**>,  
27152           <**unistd.h**>27153 **CHANGE HISTORY**

27154           First released in Issue 1. Derived from Issue 1 of the SVID.

27155 **Issue 5**27156           A statement is added to the description indicating the effects of this function on the different  
27157           scheduling policies and multi-threaded processes.

27158 **NAME**

27159 nl\_langinfo — language information

27160 **SYNOPSIS**

27161 xSI #include &lt;langinfo.h&gt;

27162 char \*nl\_langinfo(nl\_item item);

27163

27164 **DESCRIPTION**

27165 The *nl\_langinfo()* function shall return a pointer to a string containing information relevant to  
27166 the particular language or cultural area defined in the program's locale (see <langinfo.h>). The  
27167 manifest constant names and values of *item* are defined in <langinfo.h>. For example:

27168 nl\_langinfo(ABDAY\_1)

27169 would return a pointer to the string "Dom" if the identified language was Portuguese, and  
27170 "Sun" if the identified language was English.

27171 Calls to *setlocale()* with a category corresponding to the category of *item* (see <langinfo.h>), or to  
27172 the category *LC\_ALL*, may overwrite the array pointed to by the return value.

27173 The *nl\_langinfo()* function need not be reentrant. A function that is not required to be reentrant is  
27174 not required to be thread-safe.

27175 **RETURN VALUE**

27176 In a locale where *langinfo* data is not defined, *nl\_langinfo()* shall return a pointer to the  
27177 corresponding string in the POSIX locale. In all locales, *nl\_langinfo()* shall return a pointer to an  
27178 empty string if *item* contains an invalid setting.

27179 This pointer may point to static data that may be overwritten on the next call.

27180 **ERRORS**

27181 No errors are defined.

27182 **EXAMPLES**27183 **Getting Date and Time Formatting Information**

27184 The following example returns a pointer to a string containing date and time formatting  
27185 information, as defined in the *LC\_TIME* category of the current locale.

27186 #include &lt;time.h&gt;

27187 #include &lt;langinfo.h&gt;

27188 ...

27189 strftime(datestring, sizeof(datestring), nl\_langinfo(D\_T\_FMT), tm);

27190 ...

27191 **APPLICATION USAGE**

27192 The array pointed to by the return value should not be modified by the program, but may be  
27193 modified by further calls to *nl\_langinfo()*.

27194 **RATIONALE**

27195 None.

27196 **FUTURE DIRECTIONS**

27197 None.

27198 **SEE ALSO**

27199        *setlocale()*, the Base Definitions volume of IEEE Std 1003.1-200x, <**langinfo.h**>, <**nl\_types.h**>, the  
27200        Base Definitions volume of IEEE Std 1003.1-200x, Chapter 7, Locale

27201 **CHANGE HISTORY**

27202        First released in Issue 2.

27203 **Issue 5**

27204        The last paragraph of the DESCRIPTION is moved from the APPLICATION USAGE section.

27205        A note indicating that this function need not be reentrant is added to the DESCRIPTION.

27206 **NAME**

27207 nrand48 — generate uniformly distributed pseudo-random non-negative long integers

27208 **SYNOPSIS**

27209 xSI #include <stdlib.h>

27210 long nrand48(unsigned short xsubi[3]);

27211

27212 **DESCRIPTION**

27213 Refer to *drand48()*.

27214 **NAME**

27215           ntohl — convert values between host and network byte order

27216 **SYNOPSIS**

27217           #include <arpa/inet.h>

27218           uint32\_t ntohl(uint32\_t *netlong*);

27219 **DESCRIPTION**

27220           Refer to *htonl()*.

27221 **NAME**

27222       ntohs — convert values between host and network byte order

27223 **SYNOPSIS**

27224       #include <arpa/inet.h>

27225       uint16\_t ntohs(uint16\_t *netshort*);

27226 **DESCRIPTION**

27227       Refer to *htonl()*.

## 27228 NAME

27229 open — open a file

## 27230 SYNOPSIS

27231 OH #include &lt;sys/stat.h&gt;

27232 #include &lt;fcntl.h&gt;

27233 int open(const char \*path, int oflag, ... );

## 27234 DESCRIPTION

27235 The *open()* function shall establish the connection between a file and a file descriptor. It shall  
 27236 create an open file description that refers to a file and a file descriptor that refers to that open file  
 27237 description. The file descriptor is used by other I/O functions to refer to that file. The *path*  
 27238 argument points to a pathname naming the file.

27239 The *open()* function shall return a file descriptor for the named file that is the lowest file  
 27240 descriptor not currently open for that process. The open file description is new, and therefore the  
 27241 file descriptor shall not share it with any other process in the system. The FD\_CLOEXEC file  
 27242 descriptor flag associated with the new file descriptor shall be cleared.

27243 The file offset used to mark the current position within the file shall be set to the beginning of the  
 27244 file.

27245 The file status flags and file access modes of the open file description shall be set according to  
 27246 the value of *oflag*.

27247 Values for *oflag* are constructed by a bitwise-inclusive OR of flags from the following list,  
 27248 defined in <fcntl.h>. Applications shall specify exactly one of the first three values (file access  
 27249 modes) below in the value of *oflag*:

27250 O\_RDONLY Open for reading only.

27251 O\_WRONLY Open for writing only.

27252 O\_RDWR Open for reading and writing. The result is undefined if this flag is applied to  
27253 a FIFO.

27254 Any combination of the following may be used:

27255 O\_APPEND If set, the file offset shall be set to the end of the file prior to each write.

27256 O\_CREAT If the file exists, this flag has no effect except as noted under O\_EXCL below. |  
 27257 Otherwise, the file shall be created; the user ID of the file shall be set to the |  
 27258 effective user ID of the process; the group ID of the file shall be set to the |  
 27259 group ID of the file's parent directory or to the effective group ID of the |  
 27260 process; and the access permission bits (see <sys/stat.h>) of the file mode shall |  
 27261 be set to the value of the third argument taken as type **mode\_t** modified as |  
 27262 follows: a bitwise AND is performed on the file-mode bits and the |  
 27263 corresponding bits in the complement of the process' file mode creation mask. |  
 27264 Thus, all bits in the file mode whose corresponding bit in the file mode |  
 27265 creation mask is set are cleared. When bits other than the file permission bits |  
 27266 are set, the effect is unspecified. The third argument does not affect whether |  
 27267 the file is open for reading, writing, or for both. Implementations shall provide |  
 27268 a way to initialize the file's group ID to the group ID of the parent directory. |  
 27269 Implementations may, but need not, provide an implementation-defined way |  
 27270 to initialize the file's group ID to the effective group ID of the calling process. |

27271 SIO O\_DSYNC Write I/O operations on the file descriptor shall complete as defined by |  
 27272 synchronized I/O data integrity completion. |

27273	O_EXCL	If O_CREAT and O_EXCL are set, <i>open()</i> shall fail if the file exists. The check for the existence of the file and the creation of the file if it does not exist shall be atomic with respect to other threads executing <i>open()</i> naming the same filename in the same directory with O_EXCL and O_CREAT set. If O_EXCL and O_CREAT are set, and <i>path</i> names a symbolic link, <i>open()</i> shall fail and set <i>errno</i> to [EEXIST], regardless of the contents of the symbolic link. If O_EXCL is set and O_CREAT is not set, the result is undefined.
27274		
27275		
27276		
27277		
27278		
27279		
27280	O_NOCTTY	If set and <i>path</i> identifies a terminal device, <i>open()</i> shall not cause the terminal device to become the controlling terminal for the process.
27281		
27282	O_NONBLOCK	When opening a FIFO with O_RDONLY or O_WRONLY set:
27283		<ul style="list-style-type: none"> <li>• If O_NONBLOCK is set, an <i>open()</i> for reading-only shall return without delay. An <i>open()</i> for writing-only shall return an error if no process currently has the file open for reading.</li> </ul>
27284		
27285		
27286		<ul style="list-style-type: none"> <li>• If O_NONBLOCK is clear, an <i>open()</i> for reading-only shall block the calling thread until a thread opens the file for writing. An <i>open()</i> for writing-only shall block the calling thread until a thread opens the file for reading.</li> </ul>
27287		
27288		
27289		
27290		When opening a block special or character special file that supports non-blocking opens:
27291		
27292		<ul style="list-style-type: none"> <li>• If O_NONBLOCK is set, the <i>open()</i> function shall return without blocking for the device to be ready or available. Subsequent behavior of the device is device-specific.</li> </ul>
27293		
27294		
27295		<ul style="list-style-type: none"> <li>• If O_NONBLOCK is clear, the <i>open()</i> function shall block the calling thread until the device is ready or available before returning.</li> </ul>
27296		
27297		Otherwise, the behavior of O_NONBLOCK is unspecified.
27298	SIO O_RSYNC	Read I/O operations on the file descriptor shall complete at the same level of integrity as specified by the O_DSYNC and O_SYNC flags. If both O_DSYNC and O_RSYNC are set in <i>oflag</i> , all I/O operations on the file descriptor shall complete as defined by synchronized I/O data integrity completion. If both O_SYNC and O_RSYNC are set in flags, all I/O operations on the file descriptor shall complete as defined by synchronized I/O file integrity completion.
27299		
27300		
27301		
27302		
27303		
27304		
27305	SIO O_SYNC	Write I/O operations on the file descriptor shall complete as defined by synchronized I/O file integrity completion.
27306		
27307	O_TRUNC	If the file exists and is a regular file, and the file is successfully opened O_RDWR or O_WRONLY, its length shall be truncated to 0, and the mode and owner shall be unchanged. It shall have no effect on FIFO special files or terminal device files. Its effect on other file types is implementation-defined. The result of using O_TRUNC with O_RDONLY is undefined.
27308		
27309		
27310		
27311		
27312		If O_CREAT is set and the file did not previously exist, upon successful completion, <i>open()</i> shall mark for update the <i>st_atime</i> , <i>st_ctime</i> , and <i>st_mtime</i> fields of the file and the <i>st_ctime</i> and <i>st_mtime</i> fields of the parent directory.
27313		
27314		
27315		If O_TRUNC is set and the file did previously exist, upon successful completion, <i>open()</i> shall mark for update the <i>st_ctime</i> and <i>st_mtime</i> fields of the file.
27316		

27317 SIO 27318	If both the O_SYNC and O_DSYNC flags are set, the effect is as if only the O_SYNC flag was set.	
27319 XSR 27320 27321 27322 27323 27324	If <i>path</i> refers to a STREAMS file, <i>oflag</i> may be constructed from O_NONBLOCK OR'ed with either O_RDONLY, O_WRONLY, or O_RDWR. Other flag values are not applicable to STREAMS devices and shall have no effect on them. The value O_NONBLOCK affects the operation of STREAMS drivers and certain functions applied to file descriptors associated with STREAMS files. For STREAMS drivers, the implementation of O_NONBLOCK is device-specific.	
27325 XSI 27326 27327	If <i>path</i> names the master side of a pseudo-terminal device, then it is unspecified whether <i>open()</i> locks the slave side so that it cannot be opened. Conforming applications shall call <i>unlockpt()</i> before opening the slave side.	
27328 27329	The largest value that can be represented correctly in an object of type <b>off_t</b> shall be established as the offset maximum in the open file description.	
27330	<b>RETURN VALUE</b>	
27331 27332 27333	Upon successful completion, the function shall open the file and return a non-negative integer representing the lowest numbered unused file descriptor. Otherwise, <b>-1</b> shall be returned and <i>errno</i> set to indicate the error. No files shall be created or modified if the function returns <b>-1</b> .	
27334	<b>ERRORS</b>	
27335	The <i>open()</i> function shall fail if:	
27336 27337 27338 27339	[EACCES]	Search permission is denied on a component of the path prefix, or the file exists and the permissions specified by <i>oflag</i> are denied, or the file does not exist and write permission is denied for the parent directory of the file to be created, or O_TRUNC is specified and write permission is denied.
27340	[EEXIST]	O_CREAT and O_EXCL are set, and the named file exists.
27341	[EINTR]	A signal was caught during <i>open()</i> .
27342 SIO	[EINVAL]	The implementation does not support synchronized I/O for this file.
27343 XSR 27344	[EIO]	The <i>path</i> argument names a STREAMS file and a hangup or error occurred during the <i>open()</i> .
27345	[EISDIR]	The named file is a directory and <i>oflag</i> includes O_WRONLY or O_RDWR.
27346 27347	[ELOOP]	A loop exists in symbolic links encountered during resolution of the <i>path</i> argument.
27348	[EMFILE]	{OPEN_MAX} file descriptors are currently open in the calling process.
27349	[ENAMETOOLONG]	
27350 27351		The length of the <i>path</i> argument exceeds {PATH_MAX} or a pathname component is longer than {NAME_MAX}.
27352	[ENFILE]	The maximum allowable number of files is currently open in the system.
27353 27354 27355	[ENOENT]	O_CREAT is not set and the named file does not exist; or O_CREAT is set and either the path prefix does not exist or the <i>path</i> argument points to an empty string.
27356 XSR 27357	[ENOSR]	The <i>path</i> argument names a STREAMS-based file and the system is unable to allocate a STREAM.
27358 27359	[ENOSPC]	The directory or file system that would contain the new file cannot be expanded, the file does not exist, and O_CREAT is specified.

27360	[ENOTDIR]	A component of the path prefix is not a directory.
27361	[ENXIO]	O_NONBLOCK is set, the named file is a FIFO, O_WRONLY is set, and no process has the file open for reading.
27362		
27363	[ENXIO]	The named file is a character special or block special file, and the device associated with this special file does not exist.
27364		
27365	[EOVERFLOW]	The named file is a regular file and the size of the file cannot be represented correctly in an object of type <code>off_t</code> .
27366		
27367	[EROFS]	The named file resides on a read-only file system and either O_WRONLY, O_RDWR, O_CREAT (if file does not exist), or O_TRUNC is set in the <i>oflag</i> argument.
27368		
27369		
27370		The <i>open()</i> function may fail if:
27371 XSI	[EAGAIN]	The <i>path</i> argument names the slave side of a pseudo-terminal device that is locked.
27372		
27373	[EINVAL]	The value of the <i>oflag</i> argument is not valid.
27374	[ELOOP]	More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the <i>path</i> argument.
27375		
27376	[ENAMETOOLONG]	
27377		As a result of encountering a symbolic link in resolution of the <i>path</i> argument, the length of the substituted pathname string exceeded {PATH_MAX}.
27378		
27379 XSR	[ENOMEM]	The <i>path</i> argument names a STREAMS file and the system is unable to allocate resources.
27380		
27381	[ETXTBSY]	The file is a pure procedure (shared text) file that is being executed and <i>oflag</i> is O_WRONLY or O_RDWR.
27382		

## 27383 EXAMPLES

### 27384 Opening a File for Writing by the Owner

27385 The following example opens the file `/tmp/file`, either by creating it (if it does not already exist),  
 27386 or by truncating its length to 0 (if it does exist). In the former case, if the call creates a new file,  
 27387 the access permission bits in the file mode of the file are set to permit reading and writing by the  
 27388 owner, and to permit reading only by group members and others.

27389 If the call to *open()* is successful, the file is opened for writing.

```

27390 #include <fcntl.h>
27391 ...
27392 int fd;
27393 mode_t mode = S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH;
27394 char *filename = "/tmp/file";
27395 ...
27396 fd = open(filename, O_WRONLY | O_CREAT | O_TRUNC, mode);
27397 ...

```

27398 **Opening a File Using an Existence Check**

27399 The following example uses the *open()* function to try to create the **LOCKFILE** file and open it |  
 27400 for writing. Since the *open()* function specifies the **O\_EXCL** flag, the call fails if the file already |  
 27401 exists. In that case, the program assumes that someone else is updating the password file and  
 27402 exits.

```
27403 #include <fcntl.h>
27404 #include <stdio.h>
27405 #include <stdlib.h>

27406 #define LOCKFILE "/etc/ptmp"
27407 ...
27408 int pfd; /* Integer for file descriptor returned by open() call. */
27409 ...
27410 if ((pfd = open(LOCKFILE, O_WRONLY | O_CREAT | O_EXCL,
27411             S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH)) == -1)
27412 {
27413     fprintf(stderr, "Cannot open /etc/ptmp. Try again later.\n");
27414     exit(1);
27415 }
27416 ...
```

27417 **Opening a File for Writing**

27418 The following example opens a file for writing, creating the file if it does not already exist. If the  
 27419 file does exist, the system truncates the file to zero bytes.

```
27420 #include <fcntl.h>
27421 #include <stdio.h>
27422 #include <stdlib.h>

27423 #define LOCKFILE "/etc/ptmp"
27424 ...
27425 int pfd;
27426 char filename[PATH_MAX+1];
27427 ...
27428 if ((pfd = open(filename, O_WRONLY | O_CREAT | O_TRUNC,
27429             S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH)) == -1)
27430 {
27431     perror("Cannot open output file\n"); exit(1);
27432 }
27433 ...
```

27434 **APPLICATION USAGE**

27435 None.

27436 **RATIONALE**

27437 Except as specified in this volume of IEEE Std 1003.1-200x, the flags allowed in *oflag* are not  
 27438 mutually-exclusive and any number of them may be used simultaneously.

27439 Some implementations permit opening FIFOs with **O\_RDWR**. Since FIFOs could be  
 27440 implemented in other ways, and since two file descriptors can be used to the same effect, this  
 27441 possibility is left as undefined.

27442 See *getgroups()* about the group of a newly created file.

27443 The use of *open()* to create a regular file is preferable to the use of *creat()*, because the latter is  
27444 redundant and included only for historical reasons.

27445 The use of the O\_TRUNC flag on FIFOs and directories (pipes cannot be *open()*-ed) must be  
27446 permissible without unexpected side effects (for example, *creat()* on a FIFO must not remove  
27447 data). Since terminal special files might have type-ahead data stored in the buffer, O\_TRUNC  
27448 should not affect their content, particularly if a program that normally opens a regular file  
27449 should open the current controlling terminal instead. Other file types, particularly  
27450 implementation-defined ones, are left implementation-defined.

27451 IEEE Std 1003.1-200x permits [EACCES] to be returned for conditions other than those explicitly  
27452 listed.

27453 The O\_NOCTTY flag was added to allow applications to avoid unintentionally acquiring a  
27454 controlling terminal as a side effect of opening a terminal file. This volume of  
27455 IEEE Std 1003.1-200x does not specify how a controlling terminal is acquired, but it allows an  
27456 implementation to provide this on *open()* if the O\_NOCTTY flag is not set and other conditions  
27457 specified in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal  
27458 Interface are met. The O\_NOCTTY flag is an effective no-op if the file being opened is not a  
27459 terminal device.

27460 In historical implementations the value of O\_RDONLY is zero. Because of that, it is not possible  
27461 to detect the presence of O\_RDONLY and another option. Future implementations should  
27462 encode O\_RDONLY and O\_WRONLY as bit flags so that:

27463 O\_RDONLY | O\_WRONLY == O\_RDWR

27464 In general, the *open()* function follows the symbolic link if *path* names a symbolic link. However,  
27465 the *open()* function, when called with O\_CREAT and O\_EXCL, is required to fail with [EEXIST]  
27466 if *path* names an existing symbolic link, even if the symbolic link refers to a nonexistent file. This  
27467 behavior is required so that privileged applications can create a new file in a known location  
27468 without the possibility that a symbolic link might cause the file to be created in a different  
27469 location.

27470 For example, a privileged application that must create a file with a predictable name in a user-  
27471 writable directory, such as the user's home directory, could be compromised if the user creates a  
27472 symbolic link with that name that refers to a nonexistent file in a system directory. If the user can  
27473 influence the contents of a file, the user could compromise the system by creating a new system  
27474 configuration or spool file that would then be interpreted by the system. The test for a symbolic  
27475 link which refers to a nonexistent file must be atomic with the creation of a new file.

27476 The POSIX.1-1990 standard required that the group ID of a newly created file be set to the group  
27477 ID of its parent directory or to the effective group ID of the creating process. FIPS 151-2 required  
27478 that implementations provide a way to have the group ID be set to the group ID of the  
27479 containing directory, but did not prohibit implementations also supporting a way to set the  
27480 group ID to the effective group ID of the creating process. Conforming applications should not  
27481 assume which group ID will be used. If it matters, an application can use *chown()* to set the  
27482 group ID after the file is created, or determine under what conditions the implementation will  
27483 set the desired group ID.

#### 27484 FUTURE DIRECTIONS

27485 None.

#### 27486 SEE ALSO

27487 *chmod()*, *close()*, *creat()*, *dup()*, *fcntl()*, *lseek()*, *read()*, *umask()*, *unlockpt()*, *write()*, the Base  
27488 Definitions volume of IEEE Std 1003.1-200x, <fcntl.h>, <sys/stat.h>, <sys/types.h>

27489 **CHANGE HISTORY**

27490 First released in Issue 1. Derived from Issue 1 of the SVID.

27491 **Issue 5**

27492 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX  
27493 Threads Extension.

27494 Large File Summit extensions are added.

27495 **Issue 6**

27496 In the SYNOPSIS, the optional include of the `<sys/types.h>` header is removed.

27497 The following new requirements on POSIX implementations derive from alignment with the  
27498 Single UNIX Specification:

27499 • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was  
27500 required for conforming implementations of previous POSIX specifications, it was not  
27501 required for UNIX applications.

27502 • In the DESCRIPTION, `O_CREAT` is amended to state that the group ID of the file is set to the  
27503 group ID of the file's parent directory or to the effective group ID of the process. This is a  
27504 FIPS requirement.

27505 • In the DESCRIPTION, text is added to indicate setting of the offset maximum in the open file  
27506 description. This change is to support large files.

27507 • In the ERRORS section, the `[E_OVERFLOW]` condition is added. This change is to support  
27508 large files.

27509 • The `[ENXIO]` mandatory error condition is added.

27510 • The `[EINVAL]`, `[ENAMETOOLONG]`, and `[ETXTBSY]` optional error conditions are added.

27511 The DESCRIPTION and ERRORS sections are updated so that items related to the optional XSI  
27512 STREAMS Option Group are marked.

27513 The following changes were made to align with the IEEE P1003.1a draft standard:

27514 • An explanation is added of the effect of the `O_CREAT` and `O_EXCL` flags when the path  
27515 refers to a symbolic link.

27516 • The `[ELOOP]` optional error condition is added.

27517 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

27518 The DESCRIPTION of `O_EXCL` is updated in response to IEEE PASC Interpretation 1003.1c #48.

27519 **NAME**

27520            opendir — open a directory

27521 **SYNOPSIS**

27522            #include <dirent.h>

27523            DIR \*opendir(const char \*dirname);

27524 **DESCRIPTION**

27525            The *opendir()* function shall open a directory stream corresponding to the directory named by  
 27526            the *dirname* argument. The directory stream is positioned at the first entry. If the type **DIR** is  
 27527            implemented using a file descriptor, applications shall only be able to open up to a total of  
 27528            {OPEN\_MAX} files and directories.

27529 **RETURN VALUE**

27530            Upon successful completion, *opendir()* shall return a pointer to an object of type **DIR**.  
 27531            Otherwise, a null pointer shall be returned and *errno* set to indicate the error.

27532 **ERRORS**

27533            The *opendir()* function shall fail if:

27534            [EACCES]            Search permission is denied for the component of the path prefix of *dirname* or  
 27535            read permission is denied for *dirname*.

27536            [ELOOP]            A loop exists in symbolic links encountered during resolution of the *dirname*  
 27537            argument.

27538            [ENAMETOOLONG]       The length of the *dirname* argument exceeds {PATH\_MAX} or a pathname  
 27539            component is longer than {NAME\_MAX}.  
 27540

27541            [ENOENT]            A component of *dirname* does not name an existing directory or *dirname* is an  
 27542            empty string.

27543            [ENOTDIR]           A component of *dirname* is not a directory.

27544            The *opendir()* function may fail if:

27545            [ELOOP]            More than {SYMLOOP\_MAX} symbolic links were encountered during  
 27546            resolution of the *dirname* argument.

27547            [EMFILE]            {OPEN\_MAX} file descriptors are currently open in the calling process.

27548            [ENAMETOOLONG]       As a result of encountering a symbolic link in resolution of the *dirname*  
 27549            argument, the length of the substituted pathname string exceeded  
 27550            {PATH\_MAX}.  
 27551

27552            [ENFILE]            Too many files are currently open in the system.

27553 **EXAMPLES**27554 **Open a Directory Stream**

27555 The following program fragment demonstrates how the *opendir()* function is used.

```

27556 #include <sys/types.h>
27557 #include <dirent.h>
27558 #include <libgen.h>
27559 ...
27560     DIR *dir;
27561     struct dirent *dp;
27562 ...
27563     if ((dir = opendir (".")) == NULL) {
27564         perror ("Cannot open .");
27565         exit (1);
27566     }
27567     while ((dp = readdir (dir)) != NULL) {
27568         ...

```

27569 **APPLICATION USAGE**

27570 The *opendir()* function should be used in conjunction with *readdir()*, *closedir()*, and *rewinddir()* to  
 27571 examine the contents of the directory (see the EXAMPLES section in *readdir()*). This method is  
 27572 recommended for portability.

27573 **RATIONALE**

27574 Based on historical implementations, the rules about file descriptors apply to directory streams  
 27575 as well. However, this volume of IEEE Std 1003.1-200x does not mandate that the directory  
 27576 stream be implemented using file descriptors. The description of *closedir()* clarifies that if a file  
 27577 descriptor is used for the directory stream, it is mandatory that *closedir()* deallocate the file  
 27578 descriptor. When a file descriptor is used to implement the directory stream, it behaves as if the  
 27579 FD\_CLOEXEC had been set for the file descriptor.

27580 The directory entries for dot and dot-dot are optional. This volume of IEEE Std 1003.1-200x does  
 27581 not provide a way to test *a priori* for their existence because an application that is portable must  
 27582 be written to look for (and usually ignore) those entries. Writing code that presumes that they  
 27583 are the first two entries does not always work, as many implementations permit them to be  
 27584 other than the first two entries, with a “normal” entry preceding them. There is negligible value  
 27585 in providing a way to determine what the implementation does because the code to deal with  
 27586 dot and dot-dot must be written in any case and because such a flag would add to the list of  
 27587 those flags (which has proven in itself to be objectionable) and might be abused.

27588 Since the structure and buffer allocation, if any, for directory operations are defined by the  
 27589 implementation, this volume of IEEE Std 1003.1-200x imposes no portability requirements for  
 27590 erroneous program constructs, erroneous data, or the use of unspecified values such as the use  
 27591 or referencing of a *dirp* value or a **dirent** structure value after a directory stream has been closed  
 27592 or after a *fork()* or one of the *exec* function calls.

27593 **FUTURE DIRECTIONS**

27594 None.

27595 **SEE ALSO**

27596 *closedir()*, *lstat()*, *readdir()*, *rewinddir()*, *symlink()*, the Base Definitions volume of  
 27597 IEEE Std 1003.1-200x, **<dirent.h>**, **<limits.h>**, **<sys/types.h>**

27598 **CHANGE HISTORY**

27599 First released in Issue 2.

27600 **Issue 6**

27601 In the SYNOPSIS, the optional include of the `<sys/types.h>` header is removed.

27602 The following new requirements on POSIX implementations derive from alignment with the |  
27603 Single UNIX Specification:

27604 • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was  
27605 required for conforming implementations of previous POSIX specifications, it was not  
27606 required for UNIX applications.

27607 • The [ELOOP] mandatory error condition is added.

27608 • A second [ENAMETOOLONG] is added as an optional error condition.

27609 The following changes were made to align with the IEEE P1003.1a draft standard:

27610 • The [ELOOP] optional error condition is added.

27611 **NAME**

27612            openlog — open a connection to the logging facility

27613 **SYNOPSIS**

27614 xSI        #include &lt;syslog.h&gt;

27615            void openlog(const char \*ident, int logopt, int facility);

27616

27617 **DESCRIPTION**27618            Refer to *closelog()*.

27619 **NAME**

27620           optarg, opterr, optind, optopt — options parsing variables

27621 **SYNOPSIS**

27622           #include &lt;unistd.h&gt;

27623           extern char \*optarg;

27624           extern int opterr, optind, optopt;

27625 **DESCRIPTION**27626           Refer to *getopt()*.

27627 **NAME**

27628 pathconf — get configurable pathname variables |

27629 **SYNOPSIS**

27630 #include &lt;unistd.h&gt;

27631 long pathconf(const char \*path, int name);

27632 **DESCRIPTION**27633 Refer to *fpathconf()*.

27634 **NAME**

27635 pause — suspend the thread until a signal is received

27636 **SYNOPSIS**

27637 #include <unistd.h>

27638 int pause(void);

27639 **DESCRIPTION**

27640 The *pause()* function shall suspend the calling thread until delivery of a signal whose action is  
27641 either to execute a signal-catching function or to terminate the process.

27642 If the action is to terminate the process, *pause()* shall not return.

27643 If the action is to execute a signal-catching function, *pause()* shall return after the signal-catching  
27644 function returns.

27645 **RETURN VALUE**

27646 Since *pause()* suspends thread execution indefinitely unless interrupted by a signal, there is no  
27647 successful completion return value. A value of  $-1$  shall be returned and *errno* set to indicate the  
27648 error.

27649 **ERRORS**

27650 The *pause()* function shall fail if:

27651 [EINTR] A signal is caught by the calling process and control is returned from the  
27652 signal-catching function.

27653 **EXAMPLES**

27654 None.

27655 **APPLICATION USAGE**

27656 Many common uses of *pause()* have timing windows. The scenario involves checking a  
27657 condition related to a signal and, if the signal has not occurred, calling *pause()*. When the signal  
27658 occurs between the check and the call to *pause()*, the process often blocks indefinitely. The  
27659 *sigprocmask()* and *sigsuspend()* functions can be used to avoid this type of problem.

27660 **RATIONALE**

27661 None.

27662 **FUTURE DIRECTIONS**

27663 None.

27664 **SEE ALSO**

27665 *sigsuspend()*, the Base Definitions volume of IEEE Std 1003.1-200x, <unistd.h>

27666 **CHANGE HISTORY**

27667 First released in Issue 1. Derived from Issue 1 of the SVID.

27668 **Issue 5**

27669 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

27670 **Issue 6**

27671 The APPLICATION USAGE section is added.

27672 **NAME**

27673 pclose — close a pipe stream to or from a process

27674 **SYNOPSIS**

27675 cx #include &lt;stdio.h&gt;

27676 int pclose(FILE \*stream);

27677

27678 **DESCRIPTION**

27679 The *pclose()* function shall close a stream that was opened by *popen()*, wait for the command to  
 27680 terminate, and return the termination status of the process that was running the command  
 27681 language interpreter. However, if a call caused the termination status to be unavailable to  
 27682 *pclose()*, then *pclose()* shall return  $-1$  with *errno* set to [ECHILD] to report this situation. This can  
 27683 happen if the application calls one of the following functions:

- 27684 • *wait()*
- 27685 • *waitpid()* with a *pid* argument less than or equal to 0 or equal to the process ID of the  
 27686 command line interpreter
- 27687 • Any other function not defined in this volume of IEEE Std 1003.1-200x that could do one of  
 27688 the above

27689 In any case, *pclose()* shall not return before the child process created by *popen()* has terminated.

27690 If the command language interpreter cannot be executed, the child termination status returned  
 27691 by *pclose()* shall be as if the command language interpreter terminated using *exit(127)* or  
 27692 *\_exit(127)*.

27693 The *pclose()* function shall not affect the termination status of any child of the calling process  
 27694 other than the one created by *popen()* for the associated stream.

27695 If the argument *stream* to *pclose()* is not a pointer to a stream created by *popen()*, the result of  
 27696 *pclose()* is undefined.

27697 **RETURN VALUE**

27698 Upon successful return, *pclose()* shall return the termination status of the command language  
 27699 interpreter. Otherwise, *pclose()* shall return  $-1$  and set *errno* to indicate the error.

27700 **ERRORS**

27701 The *pclose()* function shall fail if:

27702 [ECHILD] The status of the child process could not be obtained, as described above.

27703 **EXAMPLES**

27704 None.

27705 **APPLICATION USAGE**

27706 None.

27707 **RATIONALE**

27708 There is a requirement that *pclose()* not return before the child process terminates. This is  
 27709 intended to disallow implementations that return [EINTR] if a signal is received while waiting.  
 27710 If *pclose()* returned before the child terminated, there would be no way for the application to  
 27711 discover which child used to be associated with the stream, and it could not do the cleanup  
 27712 itself.

27713 If the stream pointed to by *stream* was not created by *popen()*, historical implementations of  
 27714 *pclose()* return  $-1$  without setting *errno*. To avoid requiring *pclose()* to set *errno* in this case,  
 27715 IEEE Std 1003.1-200x makes the behavior unspecified. An application should not use *pclose()* to

27716 close any stream that was not created by *popen()*.

27717 Some historical implementations of *pclose()* either block or ignore the signals SIGINT, SIGQUIT,  
27718 and SIGHUP while waiting for the child process to terminate. Since this behavior is not  
27719 described for the *pclose()* function in IEEE Std 1003.1-200x, such implementations are not  
27720 conforming. Also, some historical implementations return [EINTR] if a signal is received, even  
27721 though the child process has not terminated. Such implementations are also considered non-  
27722 conforming.

27723 Consider, for example, an application that uses:

```
27724 popen("command", "r")
```

27725 to start *command*, which is part of the same application. The parent writes a prompt to its  
27726 standard output (presumably the terminal) and then reads from the stream. The child reads the  
27727 response from the user, does some transformation on the response (pathname expansion, |  
27728 perhaps) and writes the result to its standard output. The parent process reads the result from |  
27729 the pipe, does something with it, and prints another prompt. The cycle repeats. Assuming that  
27730 both processes do appropriate buffer flushing, this would be expected to work.

27731 To conform to IEEE Std 1003.1-200x, *pclose()* must use *waitpid()*, or some similar function,  
27732 instead of *wait()*.

27733 The code sample below illustrates how the *pclose()* function might be implemented on a system  
27734 conforming to IEEE Std 1003.1-200x.

```
27735 int pclose(FILE *stream)
27736 {
27737     int stat;
27738     pid_t pid;
27739
27740     pid = <pid for process created for stream by popen(>
27741     (void) fclose(stream);
27742     while (waitpid(pid, &stat, 0) == -1) {
27743         if (errno != EINTR){
27744             stat = -1;
27745             break;
27746         }
27747     }
27748     return(stat);
27749 }
```

#### 27749 FUTURE DIRECTIONS

27750 None.

#### 27751 SEE ALSO

27752 *fork()*, *popen()*, *waitpid()*, the Base Definitions volume of IEEE Std 1003.1-200x, <stdio.h>

#### 27753 CHANGE HISTORY

27754 First released in Issue 1. Derived from Issue 1 of the SVID.

27755 **NAME**

27756 perror — write error messages to standard error

27757 **SYNOPSIS**

27758 #include &lt;stdio.h&gt;

27759 void perror(const char \*s);

27760 **DESCRIPTION**

27761 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
 27762 conflict between the requirements described here and the ISO C standard is unintentional. This  
 27763 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

27764 The *perror()* function shall map the error number accessed through the symbol *errno* to a  
 27765 language-dependent error message, which shall be written to the standard error stream as  
 27766 follows:

- 27767 • First (if *s* is not a null pointer and the character pointed to by *s* is not the null byte), the string  
 27768 pointed to by *s* followed by a colon and a <space>.
- 27769 • Then an error message string followed by a <newline>.

27770 The contents of the error message strings shall be the same as those returned by *strerror()* with  
 27771 argument *errno*.

27772 cx The *perror()* function shall mark the file associated with the standard error stream as having  
 27773 been written (*st\_ctime*, *st\_mtime* marked for update) at some time between its successful  
 27774 completion and *exit()*, *abort()*, or the completion of *fflush()* or *fclose()* on *stderr*.

27775 The *perror()* function shall not change the orientation of the standard error stream.

27776 **RETURN VALUE**27777 The *perror()* function shall not return a value.27778 **ERRORS**

27779 No errors are defined.

27780 **EXAMPLES**27781 **Printing an Error Message for a Function**

27782 The following example replaces *bufptr* with a buffer that is the necessary size. If an error occurs,  
 27783 the *perror()* function prints a message and the program exits.

```

27784 #include <stdio.h>
27785 #include <stdlib.h>
27786 ...
27787 char *bufptr;
27788 size_t szbuf;
27789 ...
27790 if ((bufptr = malloc(szbuf)) == NULL) {
27791     perror("malloc"); exit(2);
27792 }
27793 ...

```

27794 **APPLICATION USAGE**

27795 None.

27796 **RATIONALE**

27797           None.

27798 **FUTURE DIRECTIONS**

27799           None.

27800 **SEE ALSO**27801           *strerror()*, the Base Definitions volume of IEEE Std 1003.1-200x, <**stdio.h**>27802 **CHANGE HISTORY**

27803           First released in Issue 1. Derived from Issue 1 of the SVID.

27804 **Issue 5**27805           A paragraph is added to the DESCRIPTION indicating that *perror()* does not change the  
27806           orientation of the standard error stream.27807 **Issue 6**

27808           Extensions beyond the ISO C standard are now marked.

27809 **NAME**

27810 pipe — create an interprocess channel

27811 **SYNOPSIS**

27812 #include &lt;unistd.h&gt;

27813 int pipe(int *fildev*[2]);27814 **DESCRIPTION**

27815 The *pipe()* function shall create a pipe and place two file descriptors, one each into the  
 27816 arguments *fildev*[0] and *fildev*[1], that refer to the open file descriptions for the read and write  
 27817 ends of the pipe. Their integer values shall be the two lowest available at the time of the *pipe()*  
 27818 call. The O\_NONBLOCK and FD\_CLOEXEC flags shall be clear on both file descriptors. (The  
 27819 *fcntl()* function can be used to set both these flags.)

27820 Data can be written to the file descriptor *fildev*[1] and read from the file descriptor *fildev*[0]. A  
 27821 read on the file descriptor *fildev*[0] shall access data written to the file descriptor *fildev*[1] on a  
 27822 first-in-first-out basis. It is unspecified whether *fildev*[0] is also open for writing and whether  
 27823 *fildev*[1] is also open for reading.

27824 A process has the pipe open for reading (correspondingly writing) if it has a file descriptor open  
 27825 that refers to the read end, *fildev*[0] (write end, *fildev*[1]).

27826 Upon successful completion, *pipe()* shall mark for update the *st\_atime*, *st\_ctime*, and *st\_mtime*  
 27827 fields of the pipe.

27828 **RETURN VALUE**

27829 Upon successful completion, 0 shall be returned; otherwise, -1 shall be returned and *errno* set to  
 27830 indicate the error.

27831 **ERRORS**27832 The *pipe()* function shall fail if:

27833 [EMFILE] More than {OPEN\_MAX} minus two file descriptors are already in use by this  
 27834 process.

27835 [ENFILE] The number of simultaneously open files in the system would exceed a  
 27836 system-imposed limit.

27837 **EXAMPLES**

27838 None.

27839 **APPLICATION USAGE**

27840 None.

27841 **RATIONALE**

27842 The wording carefully avoids using the verb “to open” in order to avoid any implication of use  
 27843 of *open()*; see also *write()*.

27844 **FUTURE DIRECTIONS**

27845 None.

27846 **SEE ALSO**

27847 *fcntl()*, *read()*, *write()*, the Base Definitions volume of IEEE Std 1003.1-200x, <*fcntl.h*>,  
 27848 <*unistd.h*>

27849 **CHANGE HISTORY**

27850 First released in Issue 1. Derived from Issue 1 of the SVID.

27851 **Issue 6**

27852 The following new requirements on POSIX implementations derive from alignment with the  
27853 Single UNIX Specification:

- 27854 • The DESCRIPTION is updated to indicate that certain dispositions of *fildev[0]* and *fildev[1]*  
27855 are unspecified.

## 27856 NAME

27857 poll — input/output multiplexing

## 27858 SYNOPSIS

27859 XSI #include &lt;poll.h&gt;

27860 int poll(struct pollfd *fds*[], nfd\_t *nfds*, int *timeout*);

27861

## 27862 DESCRIPTION

27863 The *poll()* function provides applications with a mechanism for multiplexing input/output over  
 27864 a set of file descriptors. For each member of the array pointed to by *fds*, *poll()* shall examine the  
 27865 given file descriptor for the event(s) specified in *events*. The number of **pollfd** structures in the  
 27866 *fds* array is specified by *nfds*. The *poll()* function shall identify those file descriptors on which an  
 27867 application can read or write data, or on which certain events have occurred.

27868 The *fds* argument specifies the file descriptors to be examined and the events of interest for each  
 27869 file descriptor. It is a pointer to an array with one member for each open file descriptor of  
 27870 interest. The array's members are **pollfd** structures within which *fd* specifies an open file  
 27871 descriptor and *events* and *revents* are bitmasks constructed by OR'ing a combination of the  
 27872 following event flags:

27873	POLLIN	Data other than high-priority data may be read without blocking.
27874 XSR		For STREAMS, this flag is set in <i>revents</i> even if the message is of zero length.
27875		This flag shall be equivalent to POLLRDNORM   POLLRDBAND.
27876	POLLRDNORM	Normal data may be read without blocking.
27877 XSR		For STREAMS, data on priority band 0 may be read without blocking. This
27878		flag is set in <i>revents</i> even if the message is of zero length.
27879	POLLRDBAND	Priority data may be read without blocking.
27880 XSR		For STREAMS, data on priority bands greater than 0 may be read without
27881		blocking. This flag is set in <i>revents</i> even if the message is of zero length.
27882	POLLPRI	High-priority data may be read without blocking.
27883 XSR		For STREAMS, this flag is set in <i>revents</i> even if the message is of zero length.
27884	POLLOUT	Normal data may be written without blocking.
27885 XSR		For STREAMS, data on priority band 0 may be written without blocking.
27886	POLLWRNORM	Equivalent to POLLOUT.
27887	POLLWRBAND	Priority data may be written.
27888 XSR		For STREAMS, data on priority bands greater than 0 may be written without
27889		blocking. If any priority band has been written to on this STREAM, this event
27890		only examines bands that have been written to at least once.
27891	POLLERR	An error has occurred on the device or stream. This flag is only valid in the
27892		<i>revents</i> bitmask; it shall be ignored in the <i>events</i> member.
27893	POLLHUP	The device has been disconnected. This event and POLLOUT are mutually-
27894		exclusive; a stream can never be writable if a hangup has occurred. However,
27895		this event and POLLIN, POLLRDNORM, POLLRDBAND, or POLLPRI are not
27896		mutually-exclusive. This flag is only valid in the <i>revents</i> bitmask; it shall be
27897		ignored in the <i>events</i> member.

27898	POLLNVAL	The specified <i>fd</i> value is invalid. This flag is only valid in the <i>revents</i> member; it shall be ignored in the <i>events</i> member.
27899		
27900		The significance and semantics of normal, priority, and high-priority data are file and device-specific.
27901		
27902		If the value of <i>fd</i> is less than 0, <i>events</i> shall be ignored, and <i>revents</i> shall be set to 0 in that entry on return from <i>poll()</i> .
27903		
27904		In each <b>pollfd</b> structure, <i>poll()</i> shall clear the <i>revents</i> member, except that where the application requested a report on a condition by setting one of the bits of <i>events</i> listed above, <i>poll()</i> shall set the corresponding bit in <i>revents</i> if the requested condition is true. In addition, <i>poll()</i> shall set the POLLHUP, POLLERR, and POLLNVAL flag in <i>revents</i> if the condition is true, even if the application did not set the corresponding bit in <i>events</i> .
27905		
27906		
27907		
27908		
27909		If none of the defined events have occurred on any selected file descriptor, <i>poll()</i> shall wait at least <i>timeout</i> milliseconds for an event to occur on any of the selected file descriptors. If the value of <i>timeout</i> is 0, <i>poll()</i> shall return immediately. If the value of <i>timeout</i> is -1, <i>poll()</i> shall block until a requested event occurs or until the call is interrupted.
27910		
27911		
27912		
27913		Implementations may place limitations on the granularity of timeout intervals. If the requested timeout interval requires a finer granularity than the implementation supports, the actual timeout interval shall be rounded up to the next supported value.
27914		
27915		
27916		The <i>poll()</i> function shall not be affected by the O_NONBLOCK flag.
27917	XSR	The <i>poll()</i> function shall support regular files, terminal and pseudo-terminal devices, STREAMS-based files, FIFOs, pipes, and sockets. The behavior of <i>poll()</i> on elements of <i>fds</i> that refer to other types of file is unspecified.
27918		
27919		
27920		Regular files shall always poll TRUE for reading and writing.
27921		A file descriptor for a socket that is listening for connections shall indicate that it is ready for reading, once connections are available. A file descriptor for a socket that is connecting asynchronously shall indicate that it is ready for writing, once a connection has been established.
27922		
27923		
27924		<b>RETURN VALUE</b>
27925		Upon successful completion, <i>poll()</i> shall return a non-negative value. A positive value indicates the total number of file descriptors that have been selected (that is, file descriptors for which the <i>revents</i> member is non-zero). A value of 0 indicates that the call timed out and no file descriptors have been selected. Upon failure, <i>poll()</i> shall return -1 and set <i>errno</i> to indicate the error.
27926		
27927		
27928		
27929		<b>ERRORS</b>
27930		The <i>poll()</i> function shall fail if:
27931	[EAGAIN]	The allocation of internal data structures failed but a subsequent request may succeed.
27932		
27933	[EINTR]	A signal was caught during <i>poll()</i> .
27934	XSR	[EINVAL] The <i>nfds</i> argument is greater than {OPEN_MAX}, or one of the <i>fd</i> members refers to a STREAM or multiplexer that is linked (directly or indirectly) downstream from a multiplexer.
27935		
27936		

27937 **EXAMPLES**27938 **Checking for Events on a Stream**

27939 The following example opens a pair of STREAMS devices and then waits for either one to  
27940 become writable. This example proceeds as follows:

- 27941 1. Sets the *timeout* parameter to 500 milliseconds.
- 27942 2. Opens the STREAMS devices */dev/dev0* and */dev/dev1*, and then polls them, specifying  
27943 POLLOUT and POLLWRBAND as the events of interest.
- 27944 The STREAMS device names */dev/dev0* and */dev/dev1* are only examples of how  
27945 STREAMS devices can be named; STREAMS naming conventions may vary among  
27946 systems conforming to the IEEE Std 1003.1-200x.
- 27947 3. Uses the *ret* variable to determine whether an event has occurred on either of the two  
27948 STREAMS. The *poll()* function is given 500 milliseconds to wait for an event to occur (if it  
27949 has not occurred prior to the *poll()* call).
- 27950 4. Checks the returned value of *ret*. If a positive value is returned, one of the following can  
27951 be done:
  - 27952 a. Priority data can be written to the open STREAM on priority bands greater than 0,  
27953 because the POLLWRBAND event occurred on the open STREAM (*fds[0]* or *fds[1]*).
  - 27954 b. Data can be written to the open STREAM on priority-band 0, because the POLLOUT  
27955 event occurred on the open STREAM (*fds[0]* or *fds[1]*).
- 27956 5. If the returned value is not a positive value, permission to write data to the open STREAM  
27957 (on any priority band) is denied.
- 27958 6. If the POLLHUP event occurs on the open STREAM (*fds[0]* or *fds[1]*), the device on the  
27959 open STREAM has disconnected.

```

27960 #include <stropts.h>
27961 #include <poll.h>
27962 ...
27963 struct pollfd fds[2];
27964 int timeout_msecs = 500;
27965 int ret;
27966     int i;

27967 /* Open STREAMS device. */
27968 fds[0].fd = open("/dev/dev0", ...);
27969 fds[1].fd = open("/dev/dev1", ...);
27970     fds[0].events = POLLOUT | POLLWRBAND;
27971     fds[1].events = POLLOUT | POLLWRBAND;

27972 ret = poll(fds, 2, timeout_msecs);

27973 if (ret > 0) {
27974     /* An event on one of the fds has occurred. */
27975     for (i=0; i<2; i++) {
27976         if (fds[i].revents & POLLWRBAND) {
27977             /* Priority data may be written on device number i. */
27978             ...
27979         }
27980         if (fds[i].revents & POLLOUT) {

```

```

27981          /* Data may be written on device number i. */
27982      ...
27983          }
27984      if (fds[i].revents & POLLHUP) {
27985          /* A hangup has occurred on device number i. */
27986      ...
27987          }
27988      }
27989  }

```

27990 **APPLICATION USAGE**

27991 None.

27992 **RATIONALE**

27993 None.

27994 **FUTURE DIRECTIONS**

27995 None.

27996 **SEE ALSO**

27997 *getmsg()*, *putmsg()*, *read()*, *select()*, *write()*, the Base Definitions volume of IEEE Std 1003.1-200x,  
27998 <poll.h>, <stropts.h>, Section 2.6 (on page 488)

27999 **CHANGE HISTORY**

28000 First released in Issue 4, Version 2.

28001 **Issue 5**

28002 Moved from X/OPEN UNIX extension to BASE.

28003 The description of POLLWRBAND is updated.

28004 **Issue 6**

28005 Text referring to sockets is added to the DESCRIPTION.

28006 Text relating to the XSI STREAMS Option Group is marked. |

28007 The Open Group Corrigendum Unnn/nn is applied, updating the DESCRIPTION of |

28008 POLLWRBAND. |

28009 **NAME**

28010 popen — initiate pipe streams to or from a process

28011 **SYNOPSIS**28012 cx `#include <stdio.h>`28013 `FILE *popen(const char *command, const char *mode);`

28014

28015 **DESCRIPTION**

28016 The *popen()* function shall execute the command specified by the string *command*. It shall create a  
 28017 pipe between the calling program and the executed command, and shall return a pointer to a  
 28018 stream that can be used to either read from or write to the pipe.

28019 The environment of the executed command shall be as if a child process were created within the  
 28020 *popen()* call using the *fork()* function, and the child invoked the *sh* utility using the call:

28021 `execl(shell_path, "sh", "-c", command, (char *)0);`28022 where *shell\_path* is an unspecified pathname for the *sh* utility.

28023 The *popen()* function shall ensure that any streams from previous *popen()* calls that remain open  
 28024 in the parent process are closed in the new child process.

28025 The *mode* argument to *popen()* is a string that specifies I/O mode:

- 28026 1. If *mode* is *r*, when the child process is started, its file descriptor `STDOUT_FILENO` shall be  
 28027 the writable end of the pipe, and the file descriptor *fileno(stream)* in the calling process,  
 28028 where *stream* is the stream pointer returned by *popen()*, shall be the readable end of the  
 28029 pipe.
- 28030 2. If *mode* is *w*, when the child process is started its file descriptor `STDIN_FILENO` shall be  
 28031 the readable end of the pipe, and the file descriptor *fileno(stream)* in the calling process,  
 28032 where *stream* is the stream pointer returned by *popen()*, shall be the writable end of the  
 28033 pipe.
- 28034 3. If *mode* is any other value, the result is undefined.

28035 After *popen()*, both the parent and the child process shall be capable of executing independently  
 28036 before either terminates.

28037 Pipe streams are byte-oriented.

28038 **RETURN VALUE**

28039 Upon successful completion, *popen()* shall return a pointer to an open stream that can be used to  
 28040 read or write to the pipe. Otherwise, it shall return a null pointer and may set *errno* to indicate  
 28041 the error.

28042 **ERRORS**28043 The *popen()* function may fail if:

28044 [EMFILE] {FOPEN\_MAX} or {STREAM\_MAX} streams are currently open in the calling  
 28045 process.

28046 [EINVAL] The *mode* argument is invalid.28047 The *popen()* function may also set *errno* values as described by *fork()* or *pipe()*.

28048 **EXAMPLES**

28049 None.

28050 **APPLICATION USAGE**

28051 Since open files are shared, a mode *r* command can be used as an input filter and a mode *w* |  
28052 command as an output filter.

28053 Buffered reading before opening an input filter may leave the standard input of that filter  
28054 mispositioned. Similar problems with an output filter may be prevented by careful buffer  
28055 flushing; for example, with *flush()*.

28056 A stream opened by *popen()* should be closed by *pclose()*.

28057 The behavior of *popen()* is specified for values of *mode* of *r* and *w*. Other modes such as *rb* and  
28058 *wb* might be supported by specific implementations, but these would not be portable features.  
28059 Note that historical implementations of *popen()* only check to see if the first character of *mode* is  
28060 *r*. Thus, a *mode* of *robert the robot* would be treated as *mode r*, and a *mode* of *anything else* would be  
28061 treated as *mode w*.

28062 If the application calls *waitpid()* or *waitid()* with a *pid* argument greater than 0, and it still has a  
28063 stream that was called with *popen()* open, it must ensure that *pid* does not refer to the process  
28064 started by *popen()*.

28065 To determine whether or not the environment specified in the Shell and Utilities volume of  
28066 IEEE Std 1003.1-200x is present, use the function call:

28067 `sysconf(_SC_2_VERSION)`

28068 (See *sysconf()*).

28069 **RATIONALE**

28070 The *popen()* function should not be used by programs that have set user (or group) ID privileges.  
28071 The *fork()* and *exec* family of functions (except *execlp()* and *execvp()*), should be used instead.  
28072 This prevents any unforeseen manipulation of the environment of the user that could cause  
28073 execution of commands not anticipated by the calling program.

28074 If the original and *popen()*ed processes both intend to read or write or read and write a common  
28075 file, and either will be using FILE-type C functions (*fread()*, *fwrite()*, and so on), the rules for  
28076 sharing file handles must be observed (see Section 2.5.1 (on page 485)).

28077 Since open files are shared, a mode *r* argument can be used as an input filter and a mode *w* |  
28078 argument as an output filter.

28079 The behavior of *popen()* is specified for modes of *r* and *w*. Other modes such as *rb* and *wb* might  
28080 be supported by specific implementations, but these would not be portable features. Note that  
28081 historical implementations of *popen()* only check to see if the first character of *mode* is '*r*'.  
28082 Thus, a *mode* of *robert the robot* would be treated as *mode r*, and a *mode* of *anything else* would be  
28083 treated as *mode w*.

28084 If the application calls *waitpid()* with a *pid* argument greater than zero, and it still has a  
28085 *popen()*ed stream open, it must ensure that *pid* does not refer to the process started by *popen()*.

28086 **FUTURE DIRECTIONS**

28087 None.

28088 **SEE ALSO**

28089 *pclose()*, *pipe()*, *sysconf()*, *system()*, the Base Definitions volume of IEEE Std 1003.1-200x,  
28090 `<stdio.h>`, the Shell and Utilities volume of IEEE Std 1003.1-200x, *sh* |

28091 **CHANGE HISTORY**

28092 First released in Issue 1. Derived from Issue 1 of the SVID.

28093 **Issue 5**

28094 A statement is added to the DESCRIPTION indicating that pipe streams are byte-oriented.

28095 **Issue 6**

28096 The following new requirements on POSIX implementations derive from alignment with the  
28097 Single UNIX Specification:

- 28098 • The optional [EMFILE] error condition is added.

28099 **NAME**

28100 posix\_fadvise — file advisory information (**ADVANCED REALTIME**)

28101 **SYNOPSIS**

28102 ADV #include <fcntl.h>

28103 int posix\_fadvise(int fd, off\_t offset, size\_t len, int advice);

28104

28105 **DESCRIPTION**

28106 The *posix\_fadvise()* function shall advise the implementation on the expected behavior of the  
 28107 application with respect to the data in the file associated with the open file descriptor, *fd*,  
 28108 starting at *offset* and continuing for *len* bytes. The specified range need not currently exist in the  
 28109 file. If *len* is zero, all data following *offset* is specified. The implementation may use this  
 28110 information to optimize handling of the specified data. The *posix\_fadvise()* function shall have no  
 28111 effect on the semantics of other operations on the specified data, although it may affect the  
 28112 performance of other operations.

28113 The advice to be applied to the data is specified by the *advice* parameter and may be one of the  
 28114 following values:

28115 POSIX\_FADV\_NORMAL

28116 Specifies that the application has no advice to give on its behavior with respect to the  
 28117 specified data. It is the default characteristic if no advice is given for an open file.

28118 POSIX\_FADV\_SEQUENTIAL

28119 Specifies that the application expects to access the specified data sequentially from lower  
 28120 offsets to higher offsets.

28121 POSIX\_FADV\_RANDOM

28122 Specifies that the application expects to access the specified data in a random order.

28123 POSIX\_FADV\_WILLNEED

28124 Specifies that the application expects to access the specified data in the near future.

28125 POSIX\_FADV\_DONTNEED

28126 Specifies that the application expects that it will not access the specified data in the near  
 28127 future.

28128 POSIX\_FADV\_NOREUSE

28129 Specifies that the application expects to access the specified data once and then not reuse it  
 28130 thereafter.

28131 These values are defined in <fcntl.h>.

28132 **RETURN VALUE**

28133 Upon successful completion, *posix\_fadvise()* shall return zero; otherwise, an error number shall  
 28134 be returned to indicate the error.

28135 **ERRORS**

28136 The *posix\_fadvise()* function shall fail if:

28137 [EBADF] The *fd* argument is not a valid file descriptor.

28138 [EINVAL] The value of *advice* is invalid.

28139 [ESPIPE] The *fd* argument is associated with a pipe or FIFO.

28140 **EXAMPLES**

28141           None.

28142 **APPLICATION USAGE**

28143           The *posix\_fadvise()* function is part of the Advisory Information option and need not be provided  
28144           on all implementations.

28145 **RATIONALE**

28146           None.

28147 **FUTURE DIRECTIONS**

28148           None.

28149 **SEE ALSO**28150           *posix\_madvise()*, the Base Definitions volume of IEEE Std 1003.1-200x, <fcntl.h>28151 **CHANGE HISTORY**

28152           First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

28153           In the SYNOPSIS, the inclusion of &lt;sys/types.h&gt; is no longer required.

28154 **NAME**

28155 posix\_fallocate — file space control (**ADVANCED REALTIME**)

28156 **SYNOPSIS**

28157 ADV #include <fcntl.h>

28158 int posix\_fallocate(int fd, off\_t offset, size\_t len);

28159

28160 **DESCRIPTION**

28161 The *posix\_fallocate()* function shall ensure that any required storage for regular file data starting  
 28162 at *offset* and continuing for *len* bytes is allocated on the file system storage media. If  
 28163 *posix\_fallocate()* returns successfully, subsequent writes to the specified file data shall not fail  
 28164 due to the lack of free space on the file system storage media.

28165 If the *offset+len* is beyond the current file size, then *posix\_fallocate()* shall adjust the file size to  
 28166 *offset+len*. Otherwise, the file size shall not be changed.

28167 It is implementation-defined whether a previous *posix\_fadvise()* call influences allocation  
 28168 strategy.

28169 Space allocated via *posix\_fallocate()* shall be freed by a successful call to *creat()* or *open()* that  
 28170 truncates the size of the file. Space allocated via *posix\_fallocate()* may be freed by a successful call  
 28171 to *ftruncate()* that reduces the file size to a size smaller than *offset+len*.

28172 **RETURN VALUE**

28173 Upon successful completion, *posix\_fallocate()* shall return zero; otherwise, an error number shall  
 28174 be returned to indicate the error.

28175 **ERRORS**

28176 The *posix\_fallocate()* function shall fail if:

- 28177 [EBADF] The *fd* argument is not a valid file descriptor.
- 28178 [EBADF] The *fd* argument references a file that was opened without write permission.
- 28179 [EFBIG] The value of *offset+len* is greater than the maximum file size.
- 28180 [EINTR] A signal was caught during execution.
- 28181 [EINVAL] The *len* argument was zero or the *offset* argument was less than zero.
- 28182 [EIO] An I/O error occurred while reading from or writing to a file system.
- 28183 [ENODEV] The *fd* argument does not refer to a regular file.
- 28184 [ENOSPC] There is insufficient free space remaining on the file system storage media.
- 28185 [ESPIPE] The *fd* argument is associated with a pipe or FIFO.

28186 **EXAMPLES**

28187 None.

28188 **APPLICATION USAGE**

28189 The *posix\_fallocate()* function is part of the Advisory Information option and need not be  
 28190 provided on all implementations.

28191 **RATIONALE**

28192 None.

28193 **FUTURE DIRECTIONS**

28194       None.

28195 **SEE ALSO**28196       *creat()*, *truncate()*, *open()*, *unlink()*, the Base Definitions volume of IEEE Std 1003.1-200x,  
28197       <fcntl.h>28198 **CHANGE HISTORY**

28199       First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

28200       In the SYNOPSIS, the inclusion of &lt;sys/types.h&gt; is no longer required.

28201 **NAME**

28202 posix\_madvise — memory advisory information and alignment control (**ADVANCED**  
28203 **REALTIME**)

28204 **SYNOPSIS**

28205 ADV #include <sys/mman.h>

28206 int posix\_madvise(void \*addr, size\_t len, int advice);  
28207

28208 **DESCRIPTION**

28209 MF|SHM The *posix\_madvise()* function need only be supported if either the Memory Mapped Files or the  
28210 Shared Memory Objects options are supported.

28211 The *posix\_madvise()* function shall advise the implementation on the expected behavior of the  
28212 application with respect to the data in the memory starting at address *addr*, and continuing for  
28213 *len* bytes. The implementation may use this information to optimize handling of the specified  
28214 data. The *posix\_madvise()* function shall have no effect on the semantics of access to memory in  
28215 the specified range, although it may affect the performance of access.

28216 The implementation may require that *addr* be a multiple of the page size, which is the value  
28217 returned by *sysconf()* when the name value *\_SC\_PAGESIZE* is used.

28218 The advice to be applied to the memory range is specified by the *advice* parameter and may be  
28219 one of the following values:

28220 POSIX\_MADV\_NORMAL

28221 Specifies that the application has no advice to give on its behavior with respect to the  
28222 specified range. It is the default characteristic if no advice is given for a range of memory.

28223 POSIX\_MADV\_SEQUENTIAL

28224 Specifies that the application expects to access the specified range sequentially from lower  
28225 addresses to higher addresses.

28226 POSIX\_MADV\_RANDOM

28227 Specifies that the application expects to access the specified range in a random order.

28228 POSIX\_MADV\_WILLNEED

28229 Specifies that the application expects to access the specified range in the near future.

28230 POSIX\_MADV\_DONTNEED

28231 Specifies that the application expects that it will not access the specified range in the near  
28232 future.

28233 These values are defined in the <sys/mman.h> header.

28234 **RETURN VALUE**

28235 Upon successful completion, *posix\_madvise()* shall return zero; otherwise, an error number shall  
28236 be returned to indicate the error.

28237 **ERRORS**

28238 The *posix\_madvise()* function shall fail if:

28239 [EINVAL] The value of *advice* is invalid.

28240 [ENOMEM] Addresses in the range starting at *addr* and continuing for *len* bytes are partly  
28241 or completely outside the range allowed for the address space of the calling  
28242 process.

28243 The *posix\_madvise()* function may fail if:

- 28244 [EINVAL] The value of *addr* is not a multiple of the value returned by *sysconf()* when the  
28245 name value `_SC_PAGESIZE` is used.
- 28246 [EINVAL] The value of *len* is zero.
- 28247 **EXAMPLES**
- 28248 None.
- 28249 **APPLICATION USAGE**
- 28250 The *posix\_madvise()* function is part of the Advisory Information option and need not be  
28251 provided on all implementations.
- 28252 **RATIONALE**
- 28253 None.
- 28254 **FUTURE DIRECTIONS**
- 28255 None.
- 28256 **SEE ALSO**
- 28257 *mmap()*, *posix\_fadvise()*, *sysconf()*, the Base Definitions volume of IEEE Std 1003.1-200x, |  
28258 `<sys/mman.h>` |
- 28259 **CHANGE HISTORY**
- 28260 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.
- 28261 IEEE PASC Interpretation 1003.1 #102 is applied. |

28262 **NAME**

28263 posix\_mem\_offset — find offset and length of a mapped typed memory block (**ADVANCED**  
28264 **REALTIME**)

28265 **SYNOPSIS**

28266 TYM #include <sys/mman.h>

```
28267 int posix_mem_offset(const void *restrict addr, size_t len,
28268 off_t *restrict off, size_t *restrict contig_len,
28269 int *restrict fildes);
28270
```

28271 **DESCRIPTION**

28272 The *posix\_mem\_offset()* function shall return in the variable pointed to by *off* a value that  
28273 identifies the offset (or location), within a memory object, of the memory block currently  
28274 mapped at *addr*. The function shall return in the variable pointed to by *fildes*, the descriptor used  
28275 (via *mmap()*) to establish the mapping which contains *addr*. If that descriptor was closed since  
28276 the mapping was established, the returned value of *fildes* shall be  $-1$ . The *len* argument specifies  
28277 the length of the block of the memory object the user wishes the offset for; upon return, the value  
28278 pointed to by *contig\_len* shall equal either *len*, or the length of the largest contiguous block of the  
28279 memory object that is currently mapped to the calling process starting at *addr*, whichever is  
28280 smaller.

28281 If the memory object mapped at *addr* is a typed memory object, then if the *off* and *contig\_len*  
28282 values obtained by calling *posix\_mem\_offset()* are used in a call to *mmap()* with a file descriptor  
28283 that refers to the same memory pool as *fildes* (either through the same port or through a different  
28284 port), and that was opened with neither the `POSIX_TYPED_MEM_ALLOCATE` nor the  
28285 `POSIX_TYPED_MEM_ALLOCATE_CONTIG` flag, the typed memory area that is mapped shall  
28286 be exactly the same area that was mapped at *addr* in the address space of the process that called  
28287 *posix\_mem\_offset()*.

28288 If the memory object specified by *fildes* is not a typed memory object, then the behavior of this  
28289 function is implementation-defined.

28290 **RETURN VALUE**

28291 Upon successful completion, the *posix\_mem\_offset()* function shall return zero; otherwise, the  
28292 corresponding error status value shall be returned.

28293 **ERRORS**

28294 The *posix\_mem\_offset()* function shall fail if:

28295 [EACCES] The process has not mapped a memory object supported by this function at  
28296 the given address *addr*.

28297 This function shall not return an error code of [EINTR].

28298 **EXAMPLES**

28299 None.

28300 **APPLICATION USAGE**

28301 None.

28302 **RATIONALE**

28303 None.

28304 **FUTURE DIRECTIONS**

28305           None.

28306 **SEE ALSO**

28307           *mmap()*, *posix\_typed\_mem\_open()*, the Base Definitions volume of IEEE Std 1003.1-200x,  
28308           <**sys/mman.h**>

28309 **CHANGE HISTORY**

28310           First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

28311 **NAME**

28312 posix\_memalign — aligned memory allocation (**ADVANCED REALTIME**)

28313 **SYNOPSIS**

28314 ADV `#include <stdlib.h>`

28315 `int posix_memalign(void **memptr, size_t alignment, size_t size);`

28316

28317 **DESCRIPTION**

28318 The *posix\_memalign()* function shall allocate *size* bytes aligned on a boundary specified by  
 28319 *alignment*, and shall return a pointer to the allocated memory in *memptr*. The value of *alignment*  
 28320 shall be a multiple of *sizeof(void \*)*, that is also a power of two. Upon successful completion, the  
 28321 value pointed to by *memptr* shall be a multiple of *alignment*.

28322 CX The *free()* function shall deallocate memory that has previously been allocated by  
 28323 *posix\_memalign()*.

28324 **RETURN VALUE**

28325 Upon successful completion, *posix\_memalign()* shall return zero; otherwise, an error number  
 28326 shall be returned to indicate the error.

28327 **ERRORS**

28328 The *posix\_memalign()* function shall fail if:

28329 [EINVAL] The value of the alignment parameter is not a power of two multiple of  
 28330 *sizeof(void \*)*.

28331 [ENOMEM] There is insufficient memory available with the requested alignment.

28332 **EXAMPLES**

28333 None.

28334 **APPLICATION USAGE**

28335 The *posix\_memalign()* function is part of the Advisory Information option and need not be  
 28336 provided on all implementations.

28337 **RATIONALE**

28338 None.

28339 **FUTURE DIRECTIONS**

28340 None.

28341 **SEE ALSO**

28342 *free()*, *malloc()*, the Base Definitions volume of IEEE Std 1003.1-200x, `<stdlib.h>`

28343 **CHANGE HISTORY**

28344 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

28345 In the SYNOPSIS, the inclusion of `<sys/types.h>` is no longer required.

28346 **NAME**

28347        posix\_openpt — open a pseudo terminal device

28348 **SYNOPSIS**

28349 XSI        #include &lt;stdlib.h&gt;

28350        #include &lt;fcntl.h&gt;

28351        int posix\_openpt(int oflag);

28352

28353 **DESCRIPTION**

28354        The *posix\_openpt()* function shall establish a connection between a master device for a pseudo-terminal and a file descriptor. The file descriptor is used by other I/O functions that refer to that pseudo-terminal.

28357        The file status flags and file access modes of the open file description shall be set according to the value of *oflag*.

28359        Values for *oflag* are constructed by a bitwise-inclusive OR of flags from the following list, defined in <fcntl.h>:

28361        O\_RDWR        Open for reading and writing.

28362        O\_NOCTTY       If set *posix\_openpt()* shall not cause the terminal device to become the controlling terminal for the process.

28364        The behavior of other values for the *oflag* argument is unspecified.

28365 **RETURN VALUE**

28366        Upon successful completion, the *posix\_openpt()* function shall open a master pseudo-terminal device and return a non-negative integer representing the lowest numbered unused file descriptor. Otherwise, -1 shall be returned and *errno* set to indicate the error.

28369 **ERRORS**

28370        The *posix\_openpt()* function shall fail if:

28371        [EMFILE]       {OPEN\_MAX} file descriptors are currently open in the calling process.

28372        [ENFILE]       The maximum allowable number of files is currently open in the system.

28373        The *posix\_openpt()* function may fail if:

28374        [EINVAL]       The value of *oflag* is not valid.

28375        [EAGAIN]       Out of pseudo-terminal resources.

28376 XSR        [ENOSR]       Out of STREAMS resources.

28377 **EXAMPLES**

28378        **Opening a Pseudo-Terminal and Returning the Name of the Slave Device and a File Descriptor**

28380        #include <fcntl.h>

28381        #include <stdio.h>

28382        int masterfd, slavefd;

28383        char \*slavedevice;

28384        masterfd = posix\_openpt(O\_RDWR|O\_NOCTTY);

28385        if (masterfd == -1

28386            || grantpt (masterfd) == -1

```

28387         || unlockpt (masterfd) == -1
28388         || (slavedevice = ptsname (masterfd)) == NULL)
28389         return -1;

28390     printf("slave device is: %s\n", slavedevice);

28391     slavefd = open(slave, O_RDWR|O_NOCTTY);
28392     if (slavefd < 0)
28393         return -1;

```

**28394 APPLICATION USAGE**

28395 This function is a method for portably obtaining a file descriptor of a master terminal device for  
 28396 a pseudo-terminal. The *grantpt()* and *ptsname()* functions can be used to manipulate mode and  
 28397 ownership permissions, and to obtain the name of the slave device, respectively.

**28398 RATIONALE**

28399 The standard developers considered the matter of adding a special device for cloning master  
 28400 pseudo-terminals: the */dev/ptmx* device. However, consensus could not be reached, and it was  
 28401 felt that adding a new function would permit other implementations. The *posix\_openpt()*  
 28402 function is designed to complement the *grantpt()*, *ptsname()*, and *unlockpt()* functions.

28403 On implementations supporting the */dev/ptmx* clone device, opening the master device of a  
 28404 pseudo-terminal is simply:

```

28405     mfdp = open("/dev/ptmx", oflag );
28406     if (mfdp < 0)
28407         return -1;

```

**28408 FUTURE DIRECTIONS**

28409 None.

**28410 SEE ALSO**

28411 *grantpt()*, *open()*, *ptsname()*, *unlockpt()*, the Base Definitions volume of IEEE Std 1003.1-200x,  
 28412 *<fcntl.h>*

**28413 CHANGE HISTORY**

28414 First released in Issue 6.

28415 **NAME**28416 posix\_spawn, posix\_spawnnp — spawn a process (**ADVANCED REALTIME**)28417 **SYNOPSIS**28418 SPN `#include <spawn.h>`

```

28419 int posix_spawn(pid_t *restrict pid, const char *restrict path,
28420               const posix_spawn_file_actions_t *file_actions,
28421               const posix_spawnattr_t *restrict attrp,
28422               char *const argv[restrict], char *const envp[restrict]);
28423 int posix_spawnnp(pid_t *restrict pid, const char *restrict file,
28424                  const posix_spawn_file_actions_t *file_actions,
28425                  const posix_spawnattr_t *restrict attrp,
28426                  char *const argv[restrict], char * const envp[restrict]);
28427

```

28428 **DESCRIPTION**

28429 The *posix\_spawn()* and *posix\_spawnnp()* functions shall create a new process (child process) from  
 28430 the specified process image. The new process image shall be constructed from a regular  
 28431 executable file called the new process image file.

28432 When a C program is executed as the result of this call, it shall be entered as a C language  
 28433 function call as follows:

```
28434 int main(int argc, char *argv[]);
```

28435 where *argc* is the argument count and *argv* is an array of character pointers to the arguments  
 28436 themselves. In addition, the following variable:

```
28437 extern char **environ;
```

28438 shall be initialized as a pointer to an array of character pointers to the environment strings.

28439 The argument *argv* is an array of character pointers to null-terminated strings. The last member  
 28440 of this array shall be a null pointer and is not counted in *argc*. These strings constitute the  
 28441 argument list available to the new process image. The value in *argv*[0] should point to a filename  
 28442 that is associated with the process image being started by the *posix\_spawn()* or *posix\_spawnnp()*  
 28443 function.

28444 The argument *envp* is an array of character pointers to null-terminated strings. These strings  
 28445 constitute the environment for the new process image. The environment array is terminated by a  
 28446 null pointer.

28447 The number of bytes available for the child process' combined argument and environment lists  
 28448 is {ARG\_MAX}. The implementation shall specify in the system documentation (see the Base  
 28449 Definitions volume of IEEE Std 1003.1-200x, Chapter 2, Conformance) whether any list  
 28450 overhead, such as length words, null terminators, pointers, or alignment bytes, is included in  
 28451 this total.

28452 The *path* argument to *posix\_spawn()* is a pathname that identifies the new process image file to  
 28453 execute.

28454 The *file* parameter to *posix\_spawnnp()* shall be used to construct a pathname that identifies the  
 28455 new process image file. If the *file* parameter contains a slash character, the *file* parameter shall be  
 28456 used as the pathname for the new process image file. Otherwise, the path prefix for this file shall  
 28457 be obtained by a search of the directories passed as the environment variable *PATH* (see the Base  
 28458 Definitions volume of IEEE Std 1003.1-200x, Chapter 8, Environment Variables). If this  
 28459 environment variable is not defined, the results of the search are implementation-defined.

28460 If *file\_actions* is a null pointer, then file descriptors open in the calling process shall remain open  
 28461 in the child process, except for those whose close-on-exec flag FD\_CLOEXEC is set (see *fcntl()*).  
 28462 For those file descriptors that remain open, all attributes of the corresponding open file  
 28463 descriptions, including file locks (see *fcntl()*), shall remain unchanged.

28464 If *file\_actions* is not NULL, then the file descriptors open in the child process shall be those open  
 28465 in the calling process as modified by the spawn file actions object pointed to by *file\_actions* and  
 28466 the FD\_CLOEXEC flag of each remaining open file descriptor after the spawn file actions have  
 28467 been processed. The effective order of processing the spawn file actions shall be:

- 28468 1. The set of open file descriptors for the child process shall initially be the same set as is  
 28469 open for the calling process. All attributes of the corresponding open file descriptions,  
 28470 including file locks (see *fcntl()*), shall remain unchanged.
- 28471 2. The signal mask, signal default actions, and the effective user and group IDs for the child  
 28472 process shall be changed as specified in the attributes object referenced by *attrp*.
- 28473 3. The file actions specified by the spawn file actions object shall be performed in the order in  
 28474 which they were added to the spawn file actions object.
- 28475 4. Any file descriptor that has its FD\_CLOEXEC flag set (see *fcntl()*) shall be closed.

28476 The **posix\_spawnattr\_t** spawn attributes object type is defined in **<spawn.h>**. It shall contain at  
 28477 least the attributes defined below.

28478 If the POSIX\_SPAWN\_SETPGROUP flag is set in the *spawn\_flags* attribute of the object  
 28479 referenced by *attrp*, and the *spawn-pgroup* attribute of the same object is non-zero, then the  
 28480 child's process group shall be as specified in the *spawn-pgroup* attribute of the object referenced  
 28481 by *attrp*.

28482 As a special case, if the POSIX\_SPAWN\_SETPGROUP flag is set in the *spawn\_flags* attribute of  
 28483 the object referenced by *attrp*, and the *spawn-pgroup* attribute of the same object is set to zero,  
 28484 then the child shall be in a new process group with a process group ID equal to its process ID.

28485 If the POSIX\_SPAWN\_SETPGROUP flag is not set in the *spawn\_flags* attribute of the object  
 28486 referenced by *attrp*, the new child process shall inherit the parent's process group.

28487 PS If the POSIX\_SPAWN\_SETSCHEDPARAM flag is set in the *spawn\_flags* attribute of the object  
 28488 referenced by *attrp*, but POSIX\_SPAWN\_SETSCHEDULER is not set, the new process image  
 28489 shall initially have the scheduling policy of the calling process with the scheduling parameters  
 28490 specified in the *spawn-schedparam* attribute of the object referenced by *attrp*.

28491 If the POSIX\_SPAWN\_SETSCHEDULER flag is set in *spawn\_flags* attribute of the object  
 28492 referenced by *attrp* (regardless of the setting of the POSIX\_SPAWN\_SETSCHEDPARAM flag),  
 28493 the new process image shall initially have the scheduling policy specified in the *spawn-*  
 28494 *schedpolicy* attribute of the object referenced by *attrp* and the scheduling parameters specified in  
 28495 the *spawn-schedparam* attribute of the same object.

28496 The POSIX\_SPAWN\_RESETEUIDS flag in the *spawn\_flags* attribute of the object referenced by *attrp* |  
 28497 governs the effective user ID of the child process. If this flag is not set, the child process shall |  
 28498 inherit the parent process' effective user ID. If this flag is set, the child process' effective user ID |  
 28499 shall be reset to the parent's real user ID. In either case, if the set-user-ID mode bit of the new |  
 28500 process image file is set, the effective user ID of the child process shall become that file's owner |  
 28501 ID before the new process image begins execution. |

28502 The POSIX\_SPAWN\_RESETEUIDS flag in the *spawn\_flags* attribute of the object referenced by *attrp* |  
 28503 also governs the effective group ID of the child process. If this flag is not set, the child process |  
 28504 shall inherit the parent process' effective group ID. If this flag is set, the child process' effective |  
 28505 group ID shall be reset to the parent's real group ID. In either case, if the set-group-ID mode bit |

28506 of the new process image file is set, the effective group ID of the child process shall become that  
 28507 file's group ID before the new process image begins execution.

28508 If the POSIX\_SPAWN\_SETSIGMASK flag is set in the *spawn\_flags* attribute of the object  
 28509 referenced by *attrp*, the child process shall initially have the signal mask specified in the *spawn-*  
 28510 *sigmask* attribute of the object referenced by *attrp*.

28511 If the POSIX\_SPAWN\_SETSIGDEF flag is set in the *spawn\_flags* attribute of the object referenced  
 28512 by *attrp*, the signals specified in the *spawn\_sigdefault* attribute of the same object shall be set to  
 28513 their default actions in the child process. Signals set to the default action in the parent process  
 28514 shall be set to the default action in the child process.

28515 Signals set to be caught by the calling process shall be set to the default action in the child  
 28516 process.

28517 Except for SIGCHLD, signals set to be ignored by the calling process image shall be set to be  
 28518 ignored by the child process, unless otherwise specified by the POSIX\_SPAWN\_SETSIGDEF flag  
 28519 being set in the *spawn\_flags* attribute of the object referenced by *attrp* and the signals being  
 28520 indicated in the *spawn\_sigdefault* attribute of the object referenced by *attrp*.

28521 If the SIGCHLD signal is set to be ignored by the calling process, it is unspecified whether the  
 28522 SIGCHLD signal is set to be ignored or to the default action in the child process, unless  
 28523 otherwise specified by the POSIX\_SPAWN\_SETSIGDEF flag being set in the *spawn\_flags*  
 28524 attribute of the object referenced by *attrp* and the SIGCHLD signal being indicated in the  
 28525 *spawn\_sigdefault* attribute of the object referenced by *attrp*.

28526 If the value of the *attrp* pointer is NULL, then the default values are used.

28527 All process attributes, other than those influenced by the attributes set in the object referenced  
 28528 by *attrp* as specified above or by the file descriptor manipulations specified in *file\_actions*, shall  
 28529 appear in the new process image as though *fork()* had been called to create a child process and  
 28530 then a member of the *exec* family of functions had been called by the child process to execute the  
 28531 new process image.

28532 THR It is implementation-defined whether the fork handlers are run when *posix\_spawn()* or  
 28533 *posix\_spawnnp()* is called.

28534 **RETURN VALUE**

28535 Upon successful completion, *posix\_spawn()* and *posix\_spawnnp()* shall return the process ID of the  
 28536 child process to the parent process, in the variable pointed to by a non-NULL *pid* argument, and  
 28537 shall return zero as the function return value. Otherwise, no child process shall be created, the  
 28538 value stored into the variable pointed to by a non-NULL *pid* is unspecified, and an error number  
 28539 shall be returned as the function return value to indicate the error. If the *pid* argument is a null  
 28540 pointer, the process ID of the child is not returned to the caller.

28541 **ERRORS**

28542 The *posix\_spawn()* and *posix\_spawnnp()* functions may fail if:

28543 [EINVAL] The value specified by *file\_actions* or *attrp* is invalid.

28544 If this error occurs after the calling process successfully returns from the *posix\_spawn()* or  
 28545 *posix\_spawnnp()* function, the child process may exit with exit status 127.

28546 If *posix\_spawn()* or *posix\_spawnnp()* fail for any of the reasons that would cause *fork()* or one of  
 28547 the *exec* family of functions to fail, an error value shall be returned as described by *fork()* and  
 28548 *exec*, respectively (or, if the error occurs after the calling process successfully returns, the child  
 28549 process shall exit with exit status 127).

28550 If POSIX\_SPAWN\_SETPGROUP is set in the *spawn-flags* attribute of the object referenced by  
 28551 *attrp*, and *posix\_spawn()* or *posix\_spawnnp()* fails while changing the child's process group, an  
 28552 error value shall be returned as described by *setpgid()* (or, if the error occurs after the calling  
 28553 process successfully returns, the child process shall exit with exit status 127).

28554 PS If POSIX\_SPAWN\_SETSCHEDPARAM is set and POSIX\_SPAWN\_SETSCHEDULER is not set  
 28555 in the *spawn-flags* attribute of the object referenced by *attrp*, then if *posix\_spawn()* or  
 28556 *posix\_spawnnp()* fails for any of the reasons that would cause *sched\_setparam()* to fail, an error  
 28557 value shall be returned as described by *sched\_setparam()* (or, if the error occurs after the calling  
 28558 process successfully returns, the child process shall exit with exit status 127).

28559 If POSIX\_SPAWN\_SETSCHEDULER is set in the *spawn-flags* attribute of the object referenced by  
 28560 *attrp*, and if *posix\_spawn()* or *posix\_spawnnp()* fails for any of the reasons that would cause  
 28561 *sched\_setscheduler()* to fail, an error value shall be returned as described by *sched\_setscheduler()*  
 28562 (or, if the error occurs after the calling process successfully returns, the child process shall exit  
 28563 with exit status 127).

28564 If the *file\_actions* argument is not NULL, and specifies any *close*, *dup2*, or *open* actions to be  
 28565 performed, and if *posix\_spawn()* or *posix\_spawnnp()* fails for any of the reasons that would cause  
 28566 *close()*, *dup2()*, or *open()* to fail, an error value shall be returned as described by *close()*,  
 28567 *dup2()*, and *open()*, respectively (or, if the error occurs after the calling process successfully returns, the  
 28568 child process shall exit with exit status 127). An open file action may, by itself, result in any of  
 28569 the errors described by *close()* or *dup2()*, in addition to those described by *open()*.

28570 **EXAMPLES**

28571 None.

28572 **APPLICATION USAGE**

28573 These functions are part of the Spawn option and need not be provided on all implementations.

28574 **RATIONALE**

28575 The *posix\_spawn()* function and its close relation *posix\_spawnnp()* have been introduced to |  
 28576 overcome the following perceived difficulties with *fork()*: the *fork()* function is difficult or |  
 28577 impossible to implement without swapping or dynamic address translation. |

- 28578 • Swapping is generally too slow for a realtime environment. |
- 28579 • Dynamic address translation is not available everywhere that POSIX might be useful. |
- 28580 • Processes are too useful to simply option out of POSIX whenever it must run without |  
 28581 address translation or other MMU services, |

28582 Thus, POSIX needs process creation and file execution primitives that can be efficiently |  
 28583 implemented without address translation or other MMU services. |

28584 The *posix\_spawn()* function is implementable as a library routine, but both *posix\_spawn()* and |  
 28585 *posix\_spawnnp()* are designed as kernel operations. Also, although they may be an efficient |  
 28586 replacement for many *fork()/exec* pairs, their goal is to provide useful process creation |  
 28587 primitives for systems that have difficulty with *fork()*, not to provide drop-in replacements for |  
 28588 *fork()/exec*.

28589 This view of the role of *posix\_spawn()* and *posix\_spawnnp()* influenced the design of their API. It  
 28590 does not attempt to provide the full functionality of *fork()/exec* in which arbitrary user-specified  
 28591 operations of any sort are permitted between the creation of the child process and the execution  
 28592 of the new process image; any attempt to reach that level would need to provide a programming  
 28593 language as parameters. Instead, *posix\_spawn()* and *posix\_spawnnp()* are process creation  
 28594 primitives like the *Start\_Process* and *Start\_Process\_Search* Ada language bindings package  
 28595 *POSIX\_Process\_Primitives* and also like those in many operating systems that are not UNIX

28596 systems, but with some POSIX-specific additions.

28597 To achieve its coverage goals, *posix\_spawn()* and *posix\_spawnp()* have control of six types of  
 28598 inheritance: file descriptors, process group ID, user and group ID, signal mask, scheduling, and  
 28599 whether each signal ignored in the parent will remain ignored in the child, or be reset to its  
 28600 default action in the child.

28601 Control of file descriptors is required to allow an independently written child process image to  
 28602 access data streams opened by and even generated or read by the parent process without being  
 28603 specifically coded to know which parent files and file descriptors are to be used. Control of the  
 28604 process group ID is required to control how the child process' job control relates to that of the  
 28605 parent.

28606 Control of the signal mask and signal defaulting is sufficient to support the implementation of  
 28607 *system()*. Although support for *system()* is not explicitly one of the goals for *posix\_spawn()* and  
 28608 *posix\_spawnp()*, it is covered under the "at least 50%" coverage goal.

28609 The intention is that the normal file descriptor inheritance across *fork()*, the subsequent effect of  
 28610 the specified spawn file actions, and the normal file descriptor inheritance across one of the *exec*  
 28611 family of functions should fully specify open file inheritance. The implementation need make no  
 28612 decisions regarding the set of open file descriptors when the child process image begins  
 28613 execution, those decisions having already been made by the caller and expressed as the set of  
 28614 open file descriptors and their *FD\_CLOEXEC* flags at the time of the call and the spawn file  
 28615 actions object specified in the call. We have been assured that in cases where the POSIX  
 28616 *Start\_Process* Ada primitives have been implemented in a library, this method of controlling file  
 28617 descriptor inheritance may be implemented very easily.

28618 We can identify several problems with *posix\_spawn()* and *posix\_spawnp()*, but there does not  
 28619 appear to be a solution that introduces fewer problems. Environment modification for child  
 28620 process attributes not specifiable via the *attrp* or *file\_actions* arguments must be done in the  
 28621 parent process, and since the parent generally wants to save its context, it is more costly than  
 28622 similar functionality with *fork()/exec*. It is also complicated to modify the environment of a  
 28623 multi-threaded process temporarily, since all threads must agree when it is safe for the  
 28624 environment to be changed. However, this cost is only borne by those invocations of  
 28625 *posix\_spawn()* and *posix\_spawnp()* that use the additional functionality. Since extensive  
 28626 modifications are not the usual case, and are particularly unlikely in time-critical code, keeping  
 28627 much of the environment control out of *posix\_spawn()* and *posix\_spawnp()* is appropriate design.

28628 The *posix\_spawn()* and *posix\_spawnp()* functions do not have all the power of *fork()/exec*. This is  
 28629 to be expected. The *fork()* function is a wonderfully powerful operation. We do not expect to  
 28630 duplicate its functionality in a simple, fast function with no special hardware requirements. It is  
 28631 worth noting that *posix\_spawn()* and *posix\_spawnp()* are very similar to the process creation  
 28632 operations on many operating systems that are not UNIX systems.

### 28633 **Requirements**

28634 The requirements for *posix\_spawn()* and *posix\_spawnp()* are:

- 28635 • They must be implementable without an MMU or unusual hardware.
- 28636 • They must be compatible with existing POSIX standards.

28637 Additional goals are:

- 28638 • They should be efficiently implementable.
- 28639 • They should be able to replace at least 50% of typical executions of *fork()*.

28640           • A system with *posix\_spawn()* and *posix\_spawnp()* and without *fork()* should be useful, at least  
28641           for realtime applications.

28642           • A system with *fork()* and the *exec* family should be able to implement *posix\_spawn()* and  
28643           *posix\_spawnp()* as library routines.

28644           **Two-Syntax**

28645           POSIX *exec* has several calling sequences with approximately the same functionality. These  
28646           appear to be required for compatibility with existing practice. Since the existing practice for the  
28647           *posix\_spawn\*()* functions is otherwise substantially unlike POSIX, we feel that simplicity  
28648           outweighs compatibility. There are, therefore, only two names for the *posix\_spawn\*()* functions.

28649           The parameter list does not differ between *posix\_spawn()* and *posix\_spawnp()*; *posix\_spawnp()*  
28650           interprets the second parameter more elaborately than *posix\_spawn()*.

28651           **Compatibility with POSIX.5 (Ada)**

28652           The *Start\_Process* and *Start\_Process\_Search* procedures from the POSIX\_Process\_Primitives  
28653           package from the Ada language binding to POSIX.1 encapsulate *fork()* and *exec* functionality in a  
28654           manner similar to that of *posix\_spawn()* and *posix\_spawnp()*. Originally, in keeping with our  
28655           simplicity goal, the standard developers had limited the capabilities of *posix\_spawn()* and  
28656           *posix\_spawnp()* to a subset of the capabilities of *Start\_Process* and *Start\_Process\_Search*; certain  
28657           non-default capabilities were not supported. However, based on suggestions by the ballot group  
28658           to improve file descriptor mapping or drop it, and on the advice of an Ada Language Bindings  
28659           working group member, the standard developers decided that *posix\_spawn()* and *posix\_spawnp()*  
28660           should be sufficiently powerful to implement *Start\_Process* and *Start\_Process\_Search*. The  
28661           rationale is that if the Ada language binding to such a primitive had already been approved as  
28662           an IEEE standard, there can be little justification for not approving the functionally-equivalent  
28663           parts of a C binding. The only three capabilities provided by *posix\_spawn()* and *posix\_spawnp()*  
28664           that are not provided by *Start\_Process* and *Start\_Process\_Search* are optionally specifying the  
28665           child's process group ID, the set of signals to be reset to default signal handling in the child  
28666           process, and the child's scheduling policy and parameters.

28667           For the Ada language binding for *Start\_Process* to be implemented with *posix\_spawn()*, that  
28668           binding would need to explicitly pass an empty signal mask and the parent's environment to  
28669           *posix\_spawn()* whenever the caller of *Start\_Process* allowed these arguments to default, since  
28670           *posix\_spawn()* does not provide such defaults. The ability of *Start\_Process* to mask user-specified  
28671           signals during its execution is functionally unique to the Ada language binding and must be  
28672           dealt with in the binding separately from the call to *posix\_spawn()*.

28673           **Process Group**

28674           The process group inheritance field can be used to join the child process with an existing process  
28675           group. By assigning a value of zero to the *spawn-pgroup* attribute of the object referenced by  
28676           *attrp*, the *setpgid()* mechanism will place the child process in a new process group.

28677

**Threads**

28678

28679

28680

28681

28682

28683

28684

28685

28686

Without the *posix\_spawn()* and *posix\_spawnp()* functions, systems without address translation can still use threads to give an abstraction of concurrency. In many cases, thread creation suffices, but it is not always a good substitute. The *posix\_spawn()* and *posix\_spawnp()* functions are considerably “heavier” than thread creation. Processes have several important attributes that threads do not. Even without address translation, a process may have base-and-bound memory protection. Each process has a process environment including security attributes and file capabilities, and powerful scheduling attributes. Processes abstract the behavior of non-uniform-memory-architecture multi-processors better than threads, and they are more convenient to use for activities that are not closely linked.

28687

28688

28689

28690

28691

28692

The *posix\_spawn()* and *posix\_spawnp()* functions may not bring support for multiple processes to every configuration. Process creation is not the only piece of operating system support required to support multiple processes. The total cost of support for multiple processes may be quite high in some circumstances. Existing practice shows that support for multiple processes is uncommon and threads are common among “tiny kernels”. There should, therefore, probably continue to be AEPs for operating systems with only one process.

28693

**Asynchronous Error Notification**

28694

28695

28696

28697

28698

A library implementation of *posix\_spawn()* or *posix\_spawnp()* may not be able to detect all possible errors before it forks the child process. IEEE Std 1003.1-200x provides for an error indication returned from a child process which could not successfully complete the spawn operation via a special exit status which may be detected using the status value returned by *wait()* and *waitpid()*.

28699

28700

28701

28702

28703

The *stat\_val* interface and the macros used to interpret it are not well suited to the purpose of returning API errors, but they are the only path available to a library implementation. Thus, an implementation may cause the child process to exit with exit status 127 for any error detected during the spawn process after the *posix\_spawn()* or *posix\_spawnp()* function has successfully returned.

28704

28705

28706

28707

28708

28709

28710

28711

The standard developers had proposed using two additional macros to interpret *stat\_val*. The first, WIFSPAWNFAIL, would have detected a status that indicated that the child exited because of an error detected during the *posix\_spawn()* or *posix\_spawnp()* operations rather than during actual execution of the child process image; the second, WSPAWNERRNO, would have extracted the error value if WIFSPAWNFAIL indicated a failure. Unfortunately, the ballot group strongly opposed this because it would make a library implementation of *posix\_spawn()* or *posix\_spawnp()* dependent on kernel modifications to *waitpid()* to be able to embed special information in *stat\_val* to indicate a spawn failure.

28712

28713

28714

28715

28716

28717

28718

28719

28720

28721

28722

The 8 bits of child process exit status that are guaranteed by IEEE Std 1003.1-200x to be accessible to the waiting parent process are insufficient to disambiguate a spawn error from any other kind of error that may be returned by an arbitrary process image. No other bits of the exit status are required to be visible in *stat\_val*, so these macros could not be strictly implemented at the library level. Reserving an exit status of 127 for such spawn errors is consistent with the use of this value by *system()* and *popen()* to signal failures in these operations that occur after the function has returned but before a shell is able to execute. The exit status of 127 does not uniquely identify this class of error, nor does it provide any detailed information on the nature of the failure. Note that a kernel implementation of *posix\_spawn()* or *posix\_spawnp()* is permitted (and encouraged) to return any possible error as the function value, thus providing more detailed failure information to the parent process.

28723

28724

Thus, no special macros are available to isolate asynchronous *posix\_spawn()* or *posix\_spawnp()* errors. Instead, errors detected by the *posix\_spawn()* or *posix\_spawnp()* operations in the context

28725 of the child process before the new process image executes are reported by setting the child's  
 28726 exit status to 127. The calling process may use the WIFEXITED and WEXITSTATUS macros on  
 28727 the *stat\_val* stored by the *wait()* or *waitpid()* functions to detect spawn failures to the extent that  
 28728 other status values with which the child process image may exit (before the parent can  
 28729 conclusively determine that the child process image has begun execution) are distinct from exit  
 28730 status 127.

28731 **FUTURE DIRECTIONS**

28732 None.

28733 **SEE ALSO**

28734 *alarm()*, *chmod()*, *close()*, *dup()*, *exec*, *exit()*, *fcntl()*, *fork()*, *kill()*, *open()*,  
 28735 *posix\_spawn\_file\_actions\_addclose()*, *posix\_spawn\_file\_actions\_adddup2()*,  
 28736 *posix\_spawn\_file\_actions\_addopen()*, *posix\_spawn\_file\_actions\_destroy()*,  
 28737 *posix\_spawn\_file\_actions\_init()*, *posix\_spawnattr\_destroy()*, *posix\_spawnattr\_init()*,  
 28738 *posix\_spawnattr\_getsigdefault()*, *posix\_spawnattr\_getflags()*, *posix\_spawnattr\_getpgroup()*,  
 28739 *posix\_spawnattr\_getschedparam()*, *posix\_spawnattr\_getschedpolicy()*, *posix\_spawnattr\_getsigmask()*,  
 28740 *posix\_spawnattr\_setsigdefault()*, *posix\_spawnattr\_setflags()*, *posix\_spawnattr\_setpgroup()*,  
 28741 *posix\_spawnattr\_setschedparam()*, *posix\_spawnattr\_setschedpolicy()*, *posix\_spawnattr\_setsigmask()*,  
 28742 *sched\_setparam()*, *sched\_setscheduler()*, *setpgid()*, *setuid()*, *stat()*, *times()*, *wait()*, the Base  
 28743 Definitions volume of IEEE Std 1003.1-200x, <spawn.h>

28744 **CHANGE HISTORY**

28745 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

28746 IEEE PASC Interpretation 1003.1 #103 is applied, noting that the signal default actions are |  
 28747 changed as well as the signal mask in step 2. |

28748 IEEE PASC Interpretation 1003.1 #132 is applied. |

28749 **NAME**

28750 posix\_spawn\_file\_actions\_addclose, posix\_spawn\_file\_actions\_addopen — add close or open  
 28751 action to spawn file actions object (**ADVANCED REALTIME**)

28752 **SYNOPSIS**

```
28753 SPN #include <spawn.h>
28754 int posix_spawn_file_actions_addclose(posix_spawn_file_actions_t *
28755     file_actions, int fildes);
28756 int posix_spawn_file_actions_addopen(posix_spawn_file_actions_t *restrict
28757     file_actions, int fildes, const char *restrict path,
28758     int oflag, mode_t mode);
28759
```

28760 **DESCRIPTION**

28761 These functions shall add or delete a close or open action to a spawn file actions object. |

28762 A spawn file actions object is of type **posix\_spawn\_file\_actions\_t** (defined in `<spawn.h>`) and is |  
 28763 used to specify a series of actions to be performed by a `posix_spawn()` or `posix_spawnp()` |  
 28764 operation in order to arrive at the set of open file descriptors for the child process given the set of |  
 28765 open file descriptors of the parent. IEEE Std 1003.1-200x does not define comparison or |  
 28766 assignment operators for the type **posix\_spawn\_file\_actions\_t**.

28767 A spawn file actions object, when passed to `posix_spawn()` or `posix_spawnp()`, shall specify how |  
 28768 the set of open file descriptors in the calling process is transformed into a set of potentially open |  
 28769 file descriptors for the spawned process. This transformation shall be as if the specified sequence |  
 28770 of actions was performed exactly once, in the context of the spawned process (prior to execution |  
 28771 of the new process image), in the order in which the actions were added to the object; |  
 28772 additionally, when the new process image is executed, any file descriptor (from this new set) |  
 28773 which has its `FD_CLOEXEC` flag set shall be closed (see `posix_spawn()`). |

28774 The `posix_spawn_file_actions_addclose()` function shall add a *close* action to the object referenced |  
 28775 by `file_actions` that shall cause the file descriptor `fildes` to be closed (as if `close(fildes)` had been |  
 28776 called) when a new process is spawned using this file actions object.

28777 The `posix_spawn_file_actions_addopen()` function shall add an *open* action to the object referenced |  
 28778 by `file_actions` that shall cause the file named by `path` to be opened (as if `open(path, oflag, mode)` |  
 28779 had been called, and the returned file descriptor, if not `fildes`, had been changed to `fildes`) when a |  
 28780 new process is spawned using this file actions object. If `fildes` was already an open file descriptor, |  
 28781 it shall be closed before the new file is opened.

28782 The string described by `path` shall be copied by the `posix_spawn_file_actions_addopen()` function. |

28783 **RETURN VALUE**

28784 Upon successful completion, these functions shall return zero; otherwise, an error number shall  
 28785 be returned to indicate the error.

28786 **ERRORS**

28787 These functions shall fail if:

28788 [EBADF] The value specified by `fildes` is negative or greater than or equal to  
 28789 {OPEN\_MAX}.

28790 These functions may fail if:

28791 [EINVAL] The value specified by `file_actions` is invalid.

28792 [ENOMEM] Insufficient memory exists to add to the spawn file actions object.

28793 It shall not be considered an error for the *fildef* argument passed to these functions to specify a  
 28794 file descriptor for which the specified operation could not be performed at the time of the call.  
 28795 Any such error will be detected when the associated file actions object is later used during a  
 28796 *posix\_spawn()* or *posix\_spawnnp()* operation.

28797 **EXAMPLES**

28798 None.

28799 **APPLICATION USAGE**

28800 These functions are part of the Spawn option and need not be provided on all implementations.

28801 **RATIONALE**

28802 A spawn file actions object may be initialized to contain an ordered sequence of *close()*, *dup2()*,  
 28803 and *open()* operations to be used by *posix\_spawn()* or *posix\_spawnnp()* to arrive at the set of open  
 28804 file descriptors inherited by the spawned process from the set of open file descriptors in the  
 28805 parent at the time of the *posix\_spawn()* or *posix\_spawnnp()* call. It had been suggested that the  
 28806 *close()* and *dup2()* operations alone are sufficient to rearrange file descriptors, and that files  
 28807 which need to be opened for use by the spawned process can be handled either by having the  
 28808 calling process open them before the *posix\_spawn()* or *posix\_spawnnp()* call (and close them after),  
 28809 or by passing filenames to the spawned process (in *argv*) so that it may open them itself. The  
 28810 standard developers recommend that applications use one of these two methods when practical,  
 28811 since detailed error status on a failed open operation is always available to the application this  
 28812 way. However, the standard developers feel that allowing a spawn file actions object to specify  
 28813 open operations is still appropriate because:

- 28814 1. It is consistent with equivalent POSIX.5 (Ada) functionality.
- 28815 2. It supports the I/O redirection paradigm commonly employed by POSIX programs  
 28816 designed to be invoked from a shell. When such a program is the child process, it may not  
 28817 be designed to open files on its own.
- 28818 3. It allows file opens that might otherwise fail or violate file ownership/access rights if  
 28819 executed by the parent process.

28820 Regarding 2. above, note that the spawn open file action provides to *posix\_spawn()* and  
 28821 *posix\_spawnnp()* the same capability that the shell redirection operators provide to *system()*, only  
 28822 without the intervening execution of a shell; for example:

```
28823 system ("myprog <file1 3<file2");
```

28824 Regarding 3. above, note that if the calling process needs to open one or more files for access by  
 28825 the spawned process, but has insufficient spare file descriptors, then the open action is necessary  
 28826 to allow the *open()* to occur in the context of the child process after other file descriptors have  
 28827 been closed (that must remain open in the parent).

28828 Additionally, if a parent is executed from a file having a “set-user-id” mode bit set and the  
 28829 POSIX\_SPAWN\_RESETIDS flag is set in the spawn attributes, a file created within the parent  
 28830 process will (possibly incorrectly) have the parent’s effective user ID as its owner, whereas a file  
 28831 created via an *open()* action during *posix\_spawn()* or *posix\_spawnnp()* will have the parent’s real  
 28832 ID as its owner; and an open by the parent process may successfully open a file to which the real  
 28833 user should not have access or fail to open a file to which the real user should have access.

28834 **File Descriptor Mapping**

28835 The standard developers had originally proposed using an array which specified the mapping of  
28836 child file descriptors back to those of the parent. It was pointed out by the ballot group that it is  
28837 not possible to reshuffle file descriptors arbitrarily in a library implementation of *posix\_spawn()*  
28838 or *posix\_spawnnp()* without provision for one or more spare file descriptor entries (which simply  
28839 may not be available). Such an array requires that an implementation develop a complex  
28840 strategy to achieve the desired mapping without inadvertently closing the wrong file descriptor  
28841 at the wrong time.

28842 It was noted by a member of the Ada Language Bindings working group that the approved Ada  
28843 Language *Start\_Process* family of POSIX process primitives use a caller-specified set of file  
28844 actions to alter the normal *fork()/exec* semantics for inheritance of file descriptors in a very  
28845 flexible way, yet no such problems exist because the burden of determining how to achieve the  
28846 final file descriptor mapping is completely on the application. Furthermore, although the file  
28847 actions interface appears frightening at first glance, it is actually quite simple to implement in  
28848 either a library or the kernel.

28849 **FUTURE DIRECTIONS**

28850 None.

28851 **SEE ALSO**

28852 *close()*, *dup()*, *open()*, *posix\_spawn()*, *posix\_spawn\_file\_actions\_adddup2()*,  
28853 *posix\_spawn\_file\_actions\_destroy()*, *posix\_spawnnp()*, the Base Definitions volume of  
28854 IEEE Std 1003.1-200x, <spawn.h>

28855 **CHANGE HISTORY**

28856 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

28857 IEEE PASC Interpretation 1003.1 #105 is applied, adding a note to the DESCRIPTION that the  
28858 string pointed to by *path* is copied by the *posix\_spawn\_file\_actions\_addopen()* function.

28859 **NAME**

28860 posix\_spawn\_file\_actions\_adddup2 — add dup2 action to spawn file actions object  
 28861 (ADVANCED REALTIME)

28862 **SYNOPSIS**

```
28863 SPN #include <spawn.h>
28864
28864 int posix_spawn_file_actions_adddup2(posix_spawn_file_actions_t *
28865     file_actions, int fildes, int newfildes);
28866
```

28867 **DESCRIPTION**

28868 The *posix\_spawn\_file\_actions\_adddup2()* function shall add a *dup2()* action to the object |  
 28869 referenced by *file\_actions* that shall cause the file descriptor *fildes* to be duplicated as *newfildes* (as |  
 28870 if *dup2(fildes, newfildes)* had been called) when a new process is spawned using this file actions |  
 28871 object. |

28872 A spawn file actions object is as defined in *posix\_spawn\_file\_actions\_addclose()*. |

28873 **RETURN VALUE**

28874 Upon successful completion, the *posix\_spawn\_file\_actions\_adddup2()* function shall return zero;  
 28875 otherwise, an error number shall be returned to indicate the error.

28876 **ERRORS**

28877 The *posix\_spawn\_file\_actions\_adddup2()* function shall fail if:

28878 [EBADF] The value specified by *fildes* or *newfildes* is negative or greater than or equal to  
 28879 {OPEN\_MAX}.

28880 [ENOMEM] Insufficient memory exists to add to the spawn file actions object.

28881 The *posix\_spawn\_file\_actions\_adddup2()* function may fail if:

28882 [EINVAL] The value specified by *file\_actions* is invalid.

28883 It shall not be considered an error for the *fildes* argument passed to the  
 28884 *posix\_spawn\_file\_actions\_adddup2()* function to specify a file descriptor for which the specified  
 28885 operation could not be performed at the time of the call. Any such error will be detected when  
 28886 the associated file actions object is later used during a *posix\_spawn()* or *posix\_spawnnp()*  
 28887 operation.

28888 **EXAMPLES**

28889 None.

28890 **APPLICATION USAGE**

28891 The *posix\_spawn\_file\_actions\_adddup2()* function is part of the Spawn option and need not be  
 28892 provided on all implementations.

28893 **RATIONALE**

28894 Refer to the RATIONALE in *posix\_spawn\_file\_actions\_addclose()*.

28895 **FUTURE DIRECTIONS**

28896 None.

28897 **SEE ALSO**

28898 *dup()*, *posix\_spawn()*, *posix\_spawn\_file\_actions\_addclose()*, *posix\_spawn\_file\_actions\_destroy()*,  
 28899 *posix\_spawnnp()*, the Base Definitions volume of IEEE Std 1003.1-200x, <spawn.h>

28900 **CHANGE HISTORY**

28901 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

28902 IEEE PASC Interpretation 1003.1 #104 is applied, noting that the [EBADF] error can apply to the |

28903 *newfildes* argument in addition to *fildes*.

28904 **NAME**

28905 posix\_spawn\_file\_actions\_addopen — add open action to spawn file actions object  
28906 (ADVANCED REALTIME)

28907 **SYNOPSIS**

```
28908 SPN #include <spawn.h>  
28909 int posix_spawn_file_actions_addopen(posix_spawn_file_actions_t *restrict  
28910     file_actions, int fildes, const char *restrict path,  
28911     int oflag, mode_t mode);  
28912
```

28913 **DESCRIPTION**

28914 Refer to *posix\_spawn\_file\_actions\_addclose()*.

28915 **NAME**

28916 posix\_spawn\_file\_actions\_destroy, posix\_spawn\_file\_actions\_init — destroy and initialize  
 28917 spawn file actions object (**ADVANCED REALTIME**)

28918 **SYNOPSIS**

```
28919 SPN    #include <spawn.h>
28920
28921    int posix_spawn_file_actions_destroy(posix_spawn_file_actions_t *
28922        file_actions);
28923    int posix_spawn_file_actions_init(posix_spawn_file_actions_t *
28924        file_actions);
```

28925 **DESCRIPTION**

28926 The *posix\_spawn\_file\_actions\_destroy()* function shall destroy the object referenced by *file\_actions*; |  
 28927 the object becomes, in effect, uninitialized. An implementation may cause  
 28928 *posix\_spawn\_file\_actions\_destroy()* to set the object referenced by *file\_actions* to an invalid value. A  
 28929 destroyed spawn file actions object can be reinitialized using *posix\_spawn\_file\_actions\_init()*; the  
 28930 results of otherwise referencing the object after it has been destroyed are undefined.

28931 The *posix\_spawn\_file\_actions\_init()* function shall initialize the object referenced by *file\_actions* to  
 28932 contain no file actions for *posix\_spawn()* or *posix\_spawnnp()* to perform.

28933 A spawn file actions object is as defined in *posix\_spawn\_file\_actions\_addclose()*. |

28934 The effect of initializing an already initialized spawn file actions object is undefined. |

28935 **RETURN VALUE**

28936 Upon successful completion, these functions shall return zero; otherwise, an error number shall  
 28937 be returned to indicate the error.

28938 **ERRORS**

28939 The *posix\_spawn\_file\_actions\_init()* function shall fail if:

28940 [ENOMEM] Insufficient memory exists to initialize the spawn file actions object.

28941 The *posix\_spawn\_file\_actions\_destroy()* function may fail if:

28942 [EINVAL] The value specified by *file\_actions* is invalid.

28943 **EXAMPLES**

28944 None.

28945 **APPLICATION USAGE**

28946 These functions are part of the Spawn option and need not be provided on all implementations.

28947 **RATIONALE**

28948 Refer to the RATIONALE in *posix\_spawn\_file\_actions\_addclose()*.

28949 **FUTURE DIRECTIONS**

28950 None.

28951 **SEE ALSO**

28952 *posix\_spawn()*, *posix\_spawnnp()*, the Base Definitions volume of IEEE Std 1003.1-200x, <spawn.h>

28953 **CHANGE HISTORY**

28954 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

28955 In the SYNOPSIS, the inclusion of <sys/types.h> is no longer required.

28956 **NAME**

28957 posix\_spawn\_file\_actions\_init — initialize spawn file actions object (**ADVANCED REALTIME**)

28958 **SYNOPSIS**

28959 SPN `#include <spawn.h>`

28960 `int posix_spawn_file_actions_init(posix_spawn_file_actions_t *`  
28961 `file_actions);`

28962

28963 **DESCRIPTION**

28964 Refer to *posix\_spawn\_file\_actions\_destroy()*.

28965 **NAME**

28966 `posix_spawnattr_destroy`, `posix_spawnattr_init` — destroy and initialize spawn attributes object  
 28967 (**ADVANCED REALTIME**)

28968 **SYNOPSIS**

28969 SPN `#include <spawn.h>`

28970 `int posix_spawnattr_destroy(posix_spawnattr_t *attr);`

28971 `int posix_spawnattr_init(posix_spawnattr_t *attr);`

28972

28973 **DESCRIPTION**

28974 The `posix_spawnattr_destroy()` function shall destroy a spawn attributes object. A destroyed `attr` |  
 28975 attributes object can be reinitialized using `posix_spawnattr_init()`; the results of otherwise |  
 28976 referencing the object after it has been destroyed are undefined. An implementation may cause |  
 28977 `posix_spawnattr_destroy()` to set the object referenced by `attr` to an invalid value.

28978 The `posix_spawnattr_init()` function shall initialize a spawn attributes object `attr` with the default |  
 28979 value for all of the individual attributes used by the implementation. Results are undefined if |  
 28980 `posix_spawnattr_init()` is called specifying an already initialized `attr` attributes object.

28981 A spawn attributes object is of type **posix\_spawnattr\_t** (defined in `<spawn.h>`) and is used to |  
 28982 specify the inheritance of process attributes across a spawn operation. IEEE Std 1003.1-200x does |  
 28983 not define comparison or assignment operators for the type **posix\_spawnattr\_t**.

28984 Each implementation shall document the individual attributes it uses and their default values |  
 28985 unless these values are defined by IEEE Std 1003.1-200x. Attributes not defined by |  
 28986 IEEE Std 1003.1-200x, their default values, and the names of the associated functions to get and |  
 28987 set those attribute values are implementation-defined.

28988 The resulting spawn attributes object (possibly modified by setting individual attribute values), |  
 28989 is used to modify the behavior of `posix_spawn()` or `posix_spawnp()`. After a spawn attributes |  
 28990 object has been used to spawn a process by a call to a `posix_spawn()` or `posix_spawnp()`, any |  
 28991 function affecting the attributes object (including destruction) shall not affect any process that |  
 28992 has been spawned in this way.

28993 **RETURN VALUE**

28994 Upon successful completion, `posix_spawnattr_destroy()` and `posix_spawnattr_init()` shall return |  
 28995 zero; otherwise, an error number shall be returned to indicate the error.

28996 **ERRORS**

28997 The `posix_spawnattr_init()` function shall fail if:

28998 [ENOMEM] Insufficient memory exists to initialize the spawn attributes object.

28999 The `posix_spawnattr_destroy()` function may fail if:

29000 [EINVAL] The value specified by `attr` is invalid.

29001 **EXAMPLES**

29002 None.

29003 **APPLICATION USAGE**

29004 These functions are part of the Spawn option and need not be provided on all implementations.

29005 **RATIONALE**

29006 The original spawn interface proposed in IEEE Std 1003.1-200x defined the attributes that specify |  
 29007 the inheritance of process attributes across a spawn operation as a structure. In order to be able |  
 29008 to separate optional individual attributes under their appropriate options (that is, the `spawn-` |  
 29009 `schedparam` and `spawn-schedpolicy` attributes depending upon the Process Scheduling option), and

29010 also for extensibility and consistency with the newer POSIX interfaces, the attributes interface  
29011 has been changed to an opaque data type. This interface now consists of the type  
29012 **posix\_spawnattr\_t**, representing a spawn attributes object, together with associated functions to  
29013 initialize or destroy the attributes object, and to set or get each individual attribute. Although the  
29014 new object-oriented interface is more verbose than the original structure, it is simple to use,  
29015 more extensible, and easy to implement.

## 29016 FUTURE DIRECTIONS

29017 None.

## 29018 SEE ALSO

29019 *posix\_spawn()*, *posix\_spawnattr\_getsigdefault()*, *posix\_spawnattr\_getflags()*,  
29020 *posix\_spawnattr\_getpgroup()*, *posix\_spawnattr\_getschedparam()*, *posix\_spawnattr\_getschedpolicy()*,  
29021 *posix\_spawnattr\_getsigmask()*, *posix\_spawnattr\_setsigdefault()*, *posix\_spawnattr\_setflags()*,  
29022 *posix\_spawnattr\_setpgroup()*, *posix\_spawnattr\_setsigmask()*, *posix\_spawnattr\_setschedpolicy()*,  
29023 *posix\_spawnattr\_setschedparam()*, *posix\_spawn()*, the Base Definitions volume of  
29024 IEEE Std 1003.1-200x, <spawn.h>

## 29025 CHANGE HISTORY

29026 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

29027 IEEE PASC Interpretation 1003.1 #106 is applied, noting that the effect of initializing an already |  
29028 initialized spawn attributes option is undefined.

29029 **NAME**

29030 `posix_spawnattr_getflags`, `posix_spawnattr_setflags` — get and set spawn-flags attribute of  
 29031 spawn attributes object (**ADVANCED REALTIME**)

29032 **SYNOPSIS**

```
29033 SPN #include <spawn.h>
29034
29034 int posix_spawnattr_getflags(const posix_spawnattr_t *restrict attr,
29035 short *restrict flags);
29036 int posix_spawnattr_setflags(posix_spawnattr_t *attr, short flags);
29037
```

29038 **DESCRIPTION**

29039 The `posix_spawnattr_getflags()` function shall obtain the value of the *spawn-flags* attribute from |  
 29040 the attributes object referenced by *attr*. |

29041 The `posix_spawnattr_setflags()` function shall set the *spawn-flags* attribute in an initialized |  
 29042 attributes object referenced by *attr*. |

29043 The *spawn-flags* attribute is used to indicate which process attributes are to be changed in the |  
 29044 new process image when invoking `posix_spawn()` or `posix_spawnp()`. It is the bitwise-inclusive |  
 29045 OR of zero or more of the following flags:

```
29046 POSIX_SPAWN_RESETEIDS
29047 POSIX_SPAWN_SETPGROUP
29048 POSIX_SPAWN_SETSIGDEF
29049 POSIX_SPAWN_SETSIGMASK
29050 PS POSIX_SPAWN_SETSCHEDPARAM
29051 POSIX_SPAWN_SETSCHEDULER
29052
```

29053 These flags are defined in `<spawn.h>`. The default value of this attribute shall be as if no flags |  
 29054 were set. |

29055 **RETURN VALUE**

29056 Upon successful completion, `posix_spawnattr_getflags()` shall return zero and store the value of  
 29057 the *spawn-flags* attribute of *attr* into the object referenced by the *flags* parameter; otherwise, an  
 29058 error number shall be returned to indicate the error.

29059 Upon successful completion, `posix_spawnattr_setflags()` shall return zero; otherwise, an error  
 29060 number shall be returned to indicate the error.

29061 **ERRORS**

29062 These functions may fail if:

29063 [EINVAL] The value specified by *attr* is invalid.

29064 The `posix_spawnattr_setflags()` function may fail if:

29065 [EINVAL] The value of the attribute being set is not valid.

29066 **EXAMPLES**

29067           None.

29068 **APPLICATION USAGE**

29069           These functions are part of the Spawn option and need not be provided on all implementations.

29070 **RATIONALE**

29071           None.

29072 **FUTURE DIRECTIONS**

29073           None.

29074 **SEE ALSO**

29075           *posix\_spawn()*, *posix\_spawnattr\_destroy()*, *posix\_spawnattr\_init()*, *posix\_spawnattr\_getsigdefault()*,  
29076           *posix\_spawnattr\_getpgroup()*, *posix\_spawnattr\_getschedparam()*, *posix\_spawnattr\_getschedpolicy()*,  
29077           *posix\_spawnattr\_getsigmask()*, *posix\_spawnattr\_setsigdefault()*, *posix\_spawnattr\_setpgroup()*,  
29078           *posix\_spawnattr\_setschedparam()*, *posix\_spawnattr\_setschedpolicy()*, *posix\_spawnattr\_setsigmask()*,  
29079           *posix\_spawnnp()*, the Base Definitions volume of IEEE Std 1003.1-200x, <**spawn.h**>

29080 **CHANGE HISTORY**

29081           First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

29082 **NAME**

29083 posix\_spawnattr\_getpgroup, posix\_spawnattr\_setpgroup — get and set spawn-pgroup attribute  
 29084 of spawn attributes object (**ADVANCED REALTIME**)

29085 **SYNOPSIS**

```
29086 SPN      #include <spawn.h>
29087
29087      int posix_spawnattr_getpgroup(const posix_spawnattr_t *restrict attr,
29088                                  pid_t *restrict pgroup);
29089      int posix_spawnattr_setpgroup(posix_spawnattr_t *attr, pid_t pgroup);
29090
```

29091 **DESCRIPTION**

29092 The *posix\_spawnattr\_getpgroup()* function shall obtain the value of the *spawn-pgroup* attribute |  
 29093 from the attributes object referenced by *attr*.

29094 The *posix\_spawnattr\_setpgroup()* function shall set the *spawn-pgroup* attribute in an initialized |  
 29095 attributes object referenced by *attr*.

29096 The *spawn-pgroup* attribute represents the process group to be joined by the new process image |  
 29097 in a spawn operation (if POSIX\_SPAWN\_SETPGROUP is set in the *spawn-flags* attribute). The |  
 29098 default value of this attribute shall be zero.

29099 **RETURN VALUE**

29100 Upon successful completion, *posix\_spawnattr\_getpgroup()* shall return zero and store the value of  
 29101 the *spawn-pgroup* attribute of *attr* into the object referenced by the *pgroup* parameter; otherwise,  
 29102 an error number shall be returned to indicate the error.

29103 Upon successful completion, *posix\_spawnattr\_setpgroup()* shall return zero; otherwise, an error  
 29104 number shall be returned to indicate the error.

29105 **ERRORS**

29106 These functions may fail if:

29107 [EINVAL] The value specified by *attr* is invalid.

29108 The *posix\_spawnattr\_setpgroup()* function may fail if:

29109 [EINVAL] The value of the attribute being set is not valid.

29110 **EXAMPLES**

29111 None.

29112 **APPLICATION USAGE**

29113 These functions are part of the Spawn option and need not be provided on all implementations.

29114 **RATIONALE**

29115 None.

29116 **FUTURE DIRECTIONS**

29117 None.

29118 **SEE ALSO**

29119 *posix\_spawn()*, *posix\_spawnattr\_destroy()*, *posix\_spawnattr\_init()*, *posix\_spawnattr\_getsigdefault()*,  
 29120 *posix\_spawnattr\_getflags()*, *posix\_spawnattr\_getschedparam()*, *posix\_spawnattr\_getschedpolicy()*,  
 29121 *posix\_spawnattr\_getsigmask()*, *posix\_spawnattr\_setsigdefault()*, *posix\_spawnattr\_setflags()*,  
 29122 *posix\_spawnattr\_setschedparam()*, *posix\_spawnattr\_setschedpolicy()*, *posix\_spawnattr\_setsigmask()*,  
 29123 *posix\_spawnnp()*, the Base Definitions volume of IEEE Std 1003.1-200x, <spawn.h>

29124 **CHANGE HISTORY**

29125 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

29126 **NAME**

29127 `posix_spawnattr_getschedparam`, `posix_spawnattr_setschedparam` — get and set spawn-  
 29128 `schedparam` attribute of spawn attributes object (**ADVANCED REALTIME**)

29129 **SYNOPSIS**

```
29130 SPN PS #include <spawn.h>
29131         #include <sched.h>

29132         int posix_spawnattr_getschedparam(
29133             const posix_spawnattr_t *restrict attr,
29134             struct sched_param *restrict schedparam);
29135         int posix_spawnattr_setschedparam(posix_spawnattr_t *restrict attr,
29136             const struct sched_param *restrict schedparam);
29137
```

29138 **DESCRIPTION**

29139 The `posix_spawnattr_getschedparam()` function shall obtain the value of the `spawn-schedparam` |  
 29140 attribute from the attributes object referenced by `attr`.

29141 The `posix_spawnattr_setschedparam()` function shall set the `spawn-schedparam` attribute in an |  
 29142 initialized attributes object referenced by `attr`.

29143 The `spawn-schedparam` attribute represents the scheduling parameters to be assigned to the new |  
 29144 process image in a spawn operation (if `POSIX_SPAWN_SETSCHEDULER` or |  
 29145 `POSIX_SPAWN_SETSCHEDPARAM` is set in the `spawn-flags` attribute). The default value of this |  
 29146 attribute is unspecified.

29147 **RETURN VALUE**

29148 Upon successful completion, `posix_spawnattr_getschedparam()` shall return zero and store the  
 29149 value of the `spawn-schedparam` attribute of `attr` into the object referenced by the `schedparam`  
 29150 parameter; otherwise, an error number shall be returned to indicate the error.

29151 Upon successful completion, `posix_spawnattr_setschedparam()` shall return zero; otherwise, an  
 29152 error number shall be returned to indicate the error.

29153 **ERRORS**

29154 These functions may fail if:

29155 [EINVAL] The value specified by `attr` is invalid.

29156 The `posix_spawnattr_setschedparam()` function may fail if:

29157 [EINVAL] The value of the attribute being set is not valid.

29158 **EXAMPLES**

29159 None.

29160 **APPLICATION USAGE**

29161 These functions are part of the Spawn and Process Scheduling options and need not be provided  
 29162 on all implementations.

29163 **RATIONALE**

29164 None.

29165 **FUTURE DIRECTIONS**

29166 None.

29167 **SEE ALSO**

29168 *posix\_spawn()*, *posix\_spawnattr\_destroy()*, *posix\_spawnattr\_init()*, *posix\_spawnattr\_getsigdefault()*,  
29169 *posix\_spawnattr\_getflags()*, *posix\_spawnattr\_getpgroup()*, *posix\_spawnattr\_getschedpolicy()*,  
29170 *posix\_spawnattr\_getsigmask()*, *posix\_spawnattr\_setsigdefault()*, *posix\_spawnattr\_setflags()*,  
29171 *posix\_spawnattr\_setpgroup()*, *posix\_spawnattr\_setschedpolicy()*, *posix\_spawnattr\_setsigmask()*,  
29172 *posix\_spawnp()*, the Base Definitions volume of IEEE Std 1003.1-200x, <**sched.h**>, <**spawn.h**>

29173 **CHANGE HISTORY**

29174 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

29175 **NAME**

29176 `posix_spawnattr_getschedpolicy`, `posix_spawnattr_setschedpolicy` — get and set spawn-  
 29177 `schedpolicy` attribute of spawn attributes object (**ADVANCED REALTIME**)

29178 **SYNOPSIS**

```
29179 SPN PS #include <spawn.h>
```

```
29180 #include <sched.h>
```

```
29181 int posix_spawnattr_getschedpolicy(
```

```
29182     const posix_spawnattr_t *restrict attr,
```

```
29183     int *restrict schedpolicy);
```

```
29184 int posix_spawnattr_setschedpolicy(posix_spawnattr_t *attr,
```

```
29185     int schedpolicy);
```

```
29186
```

29187 **DESCRIPTION**

29188 The `posix_spawnattr_getschedpolicy()` function shall obtain the value of the *spawn-schedpolicy* |  
 29189 attribute from the attributes object referenced by *attr*.

29190 The `posix_spawnattr_setschedpolicy()` function shall set the *spawn-schedpolicy* attribute in an |  
 29191 initialized attributes object referenced by *attr*.

29192 The *spawn-schedpolicy* attribute represents the scheduling policy to be assigned to the new |  
 29193 process image in a spawn operation (if `POSIX_SPAWN_SETSCHEDULER` is set in the *spawn-* |  
 29194 *flags* attribute). The default value of this attribute is unspecified.

29195 **RETURN VALUE**

29196 Upon successful completion, `posix_spawnattr_getschedpolicy()` shall return zero and store the  
 29197 value of the *spawn-schedpolicy* attribute of *attr* into the object referenced by the *schedpolicy*  
 29198 parameter; otherwise, an error number shall be returned to indicate the error.

29199 Upon successful completion, `posix_spawnattr_setschedpolicy()` shall return zero; otherwise, an  
 29200 error number shall be returned to indicate the error.

29201 **ERRORS**

29202 These functions may fail if:

29203 [EINVAL] The value specified by *attr* is invalid.

29204 The `posix_spawnattr_setschedpolicy()` function may fail if:

29205 [EINVAL] The value of the attribute being set is not valid.

29206 **EXAMPLES**

29207 None.

29208 **APPLICATION USAGE**

29209 These functions are part of the Spawn and Process Scheduling options and need not be provided  
 29210 on all implementations.

29211 **RATIONALE**

29212 None.

29213 **FUTURE DIRECTIONS**

29214 None.

29215 **SEE ALSO**

29216 `posix_spawn()`, `posix_spawnattr_destroy()`, `posix_spawnattr_init()`, `posix_spawnattr_getsigdefault()`,

29217 `posix_spawnattr_getflags()`, `posix_spawnattr_getpgroup()`, `posix_spawnattr_getschedparam()`,

29218 `posix_spawnattr_getsigmask()`, `posix_spawnattr_setsigdefault()`, `posix_spawnattr_setflags()`,

29219            *posix\_spawnattr\_setpgroup()*, *posix\_spawnattr\_setschedparam()*, *posix\_spawnattr\_setsigmask()*,  
29220            *posix\_spawnnp()*, the Base Definitions volume of IEEE Std 1003.1-200x, <**sched.h**>, <**spawn.h**>

29221 **CHANGE HISTORY**

29222            First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

**29223 NAME**

29224 `posix_spawnattr_getsigdefault`, `posix_spawnattr_setsigdefault` — get and set spawn-sigdefault  
 29225 attribute of spawn attributes object (**ADVANCED REALTIME**)

**29226 SYNOPSIS**

```
29227 SPN #include <signal.h>
29228 #include <spawn.h>

29229 int posix_spawnattr_getsigdefault(
29230     const posix_spawnattr_t *restrict attr,
29231     sigset_t *restrict sigdefault);
29232 int posix_spawnattr_setsigdefault(posix_spawnattr_t *restrict attr,
29233     const sigset_t *restrict sigdefault);
29234
```

**29235 DESCRIPTION**

29236 The `posix_spawnattr_getsigdefault()` function shall obtain the value of the `spawn-sigdefault` |  
 29237 attribute from the attributes object referenced by `attr`.

29238 The `posix_spawnattr_setsigdefault()` function shall set the `spawn-sigdefault` attribute in an |  
 29239 initialized attributes object referenced by `attr`.

29240 The `spawn-sigdefault` attribute represents the set of signals to be forced to default signal handling |  
 29241 in the new process image (if `POSIX_SPAWN_SETSIGDEF` is set in the `spawn-flags` attribute) by a |  
 29242 spawn operation. The default value of this attribute shall be an empty signal set. |

**29243 RETURN VALUE**

29244 Upon successful completion, `posix_spawnattr_getsigdefault()` shall return zero and store the value  
 29245 of the `spawn-sigdefault` attribute of `attr` into the object referenced by the `sigdefault` parameter;  
 29246 otherwise, an error number shall be returned to indicate the error.

29247 Upon successful completion, `posix_spawnattr_setsigdefault()` shall return zero; otherwise, an error  
 29248 number shall be returned to indicate the error.

**29249 ERRORS**

29250 These functions may fail if:

29251 [EINVAL] The value specified by `attr` is invalid.

29252 The `posix_spawnattr_setsigdefault()` function may fail if:

29253 [EINVAL] The value of the attribute being set is not valid.

**29254 EXAMPLES**

29255 None.

**29256 APPLICATION USAGE**

29257 These functions are part of the Spawn option and need not be provided on all implementations.

**29258 RATIONALE**

29259 None.

**29260 FUTURE DIRECTIONS**

29261 None.

**29262 SEE ALSO**

29263 `posix_spawn()`, `posix_spawnattr_destroy()`, `posix_spawnattr_init()`, `posix_spawnattr_getflags()`,  
 29264 `posix_spawnattr_getpgroup()`, `posix_spawnattr_getschedparam()`, `posix_spawnattr_getschedpolicy()`,  
 29265 `posix_spawnattr_getsigmask()`, `posix_spawnattr_setflags()`, `posix_spawnattr_setpgroup()`,  
 29266 `posix_spawnattr_setschedparam()`, `posix_spawnattr_setschedpolicy()`, `posix_spawnattr_setsigmask()`,

- 29267            *posix\_spawnp()*, the Base Definitions volume of IEEE Std 1003.1-200x, <signal.h>, <spawn.h>
- 29268 **CHANGE HISTORY**
- 29269            First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

29270 **NAME**

29271 posix\_spawnattr\_getsigmask, posix\_spawnattr\_setsigmask — get and set spawn-sigmask  
 29272 attribute of spawn attributes object (**ADVANCED REALTIME**)

29273 **SYNOPSIS**

29274 SPN #include <signal.h>

29275 #include <spawn.h>

```
29276 int posix_spawnattr_getsigmask(const posix_spawnattr_t *restrict attr,
29277 sigset_t *restrict sigmask);
```

```
29278 int posix_spawnattr_setsigmask(posix_spawnattr_t *restrict attr,
29279 const sigset_t *restrict sigmask);
```

29280

29281 **DESCRIPTION**

29282 The *posix\_spawnattr\_getsigmask()* function shall obtain the value of the *spawn-sigmask* attribute  
 29283 from the attributes object referenced by *attr*.

29284 The *posix\_spawnattr\_setsigmask()* function shall set the *spawn-sigmask* attribute in an initialized  
 29285 attributes object referenced by *attr*.

29286 The *spawn-sigmask* attribute represents the signal mask in effect in the new process image of a  
 29287 spawn operation (if *POSIX\_SPAWN\_SETSIGMASK* is set in the *spawn-flags* attribute). The  
 29288 default value of this attribute is unspecified.

29289 **RETURN VALUE**

29290 Upon successful completion, *posix\_spawnattr\_getsigmask()* shall return zero and store the value  
 29291 of the *spawn-sigmask* attribute of *attr* into the object referenced by the *sigmask* parameter;  
 29292 otherwise, an error number shall be returned to indicate the error.

29293 Upon successful completion, *posix\_spawnattr\_setsigmask()* shall return zero; otherwise, an error  
 29294 number shall be returned to indicate the error.

29295 **ERRORS**

29296 These functions may fail if:

29297 [EINVAL] The value specified by *attr* is invalid.

29298 The *posix\_spawnattr\_setsigmask()* function may fail if:

29299 [EINVAL] The value of the attribute being set is not valid.

29300 **EXAMPLES**

29301 None.

29302 **APPLICATION USAGE**

29303 These functions are part of the Spawn option and need not be provided on all implementations.

29304 **RATIONALE**

29305 None.

29306 **FUTURE DIRECTIONS**

29307 None.

29308 **SEE ALSO**

29309 *posix\_spawn()*, *posix\_spawnattr\_destroy()*, *posix\_spawnattr\_init()*, *posix\_spawnattr\_getsigdefault()*,  
 29310 *posix\_spawnattr\_getflags()*, *posix\_spawnattr\_getpgroup()*, *posix\_spawnattr\_getschedparam()*,  
 29311 *posix\_spawnattr\_getschedpolicy()*, *posix\_spawnattr\_setsigdefault()*, *posix\_spawnattr\_setflags()*,  
 29312 *posix\_spawnattr\_setpgroup()*, *posix\_spawnattr\_setschedparam()*, *posix\_spawnattr\_setschedpolicy()*,  
 29313 *posix\_spawnnp()*, the Base Definitions volume of IEEE Std 1003.1-200x, <signal.h>, <spawn.h>

29314 **CHANGE HISTORY**

29315 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

29316 **NAME**

29317 `posix_spawnattr_init` — initialize spawn attributes object (**ADVANCED REALTIME**)

29318 **SYNOPSIS**

29319 SPN `#include <spawn.h>`

29320 `int posix_spawnattr_init(posix_spawnattr_t *attr);`

29321

29322 **DESCRIPTION**

29323 Refer to `posix_spawnattr_destroy()`.

29324 **NAME**

29325        posix\_spawnattr\_setflags — set spawn-flags attribute of spawn attributes object (**ADVANCED**  
29326        **REALTIME**)

29327 **SYNOPSIS**

29328 SPN    #include <spawn.h>

29329        int posix\_spawnattr\_setflags(posix\_spawnattr\_t \*attr, short flags);

29330

29331 **DESCRIPTION**

29332        Refer to *posix\_spawnattr\_getflags()*.

29333 **NAME**

29334 `posix_spawnattr_setpgroup` — set spawn-pgroup attribute of spawn attributes object  
29335 (**ADVANCED REALTIME**)

29336 **SYNOPSIS**

29337 SPN `#include <spawn.h>`

29338 `int posix_spawnattr_setpgroup(posix_spawnattr_t *attr, pid_t pgroup);`  
29339

29340 **DESCRIPTION**

29341 Refer to `posix_spawnattr_getpgroup()`.

29342 **NAME**

29343 posix\_spawnattr\_setschedparam — set spawn-schedparam attribute of spawn attributes object  
29344 (**ADVANCED REALTIME**)

29345 **SYNOPSIS**

29346 SPN PS #include <sched.h>

29347 #include <spawn.h>

```
29348 int posix_spawnattr_setschedparam(posix_spawnattr_t *restrict attr,  
29349     const struct sched_param *restrict schedparam);
```

29350

29351 **DESCRIPTION**

29352 Refer to *posix\_spawnattr\_getschedparam()*.

29353 **NAME**

29354 `posix_spawnattr_setschedpolicy` — set spawn-schedpolicy attribute of spawn attributes object  
29355 (**ADVANCED REALTIME**)

29356 **SYNOPSIS**

29357 SPN PS `#include <sched.h>`

29358 `#include <spawn.h>`

```
29359 int posix_spawnattr_setschedpolicy(posix_spawnattr_t *attr,  
29360 int schedpolicy);
```

29361

29362 **DESCRIPTION**

29363 Refer to `posix_spawnattr_getschedpolicy()`.

29364 **NAME**

29365        posix\_spawnattr\_setsigdefault — set spawn-sigdefault attribute of spawn attributes object  
29366        (ADVANCED REALTIME)

29367 **SYNOPSIS**

29368 SPN    #include <signal.h>

29369        #include <spawn.h>

```
29370        int posix_spawnattr_setsigdefault(posix_spawnattr_t *restrict attr,  
29371                                        const sigset_t *restrict sigdefault);
```

29372

29373 **DESCRIPTION**

29374        Refer to *posix\_spawnattr\_getsigdefault()*.

29375 **NAME**

29376 `posix_spawnattr_setsigmask` — set spawn-sigmask attribute of spawn attributes object  
29377 (**ADVANCED REALTIME**)

29378 **SYNOPSIS**

29379 SPN `#include <signal.h>`

29380 `#include <spawn.h>`

```
29381 int posix_spawnattr_setsigmask(posix_spawnattr_t *restrict attr,  
29382     const sigset_t *restrict sigmask);
```

29383

29384 **DESCRIPTION**

29385 Refer to `posix_spawnattr_getsigmask()`.

29386 **NAME**

29387 posix\_spawnnp — spawn a process (**ADVANCED REALTIME**)

29388 **SYNOPSIS**

29389 SPN `#include <spawn.h>`

```
29390 int posix_spawnnp(pid_t *restrict pid, const char *restrict file,  
29391                  const posix_spawn_file_actions_t *file_actions,  
29392                  const posix_spawnattr_t *restrict attrp,  
29393                  char *const argv[restrict], char *const envp[restrict]);  
29394
```

29395 **DESCRIPTION**

29396 Refer to *posix\_spawn()*.

29397 **NAME**

29398        *posix\_trace\_attr\_destroy*, *posix\_trace\_attr\_init* — trace stream attributes object destroy and  
 29399        initialization (**TRACING**)

29400 **SYNOPSIS**

```
29401 TRC    #include <trace.h>
```

```
29402        int posix_trace_attr_destroy(trace_attr_t *attr);
```

```
29403        int posix_trace_attr_init(trace_attr_t *attr);
```

29404

29405 **DESCRIPTION**

29406        The *posix\_trace\_attr\_destroy()* function shall destroy an initialized trace attributes object. A  
 29407        destroyed *attr* attributes object can be reinitialized using *posix\_trace\_attr\_init()*; the results of  
 29408        otherwise referencing the object after it has been destroyed are undefined.

29409        The *posix\_trace\_attr\_init()* function shall initialize a trace attributes object *attr* with the default  
 29410        value for all of the individual attributes used by a given implementation. The read-only  
 29411        *generation-version* and *clock-resolution* attributes of the newly initialized trace attributes object  
 29412        shall be set to their appropriate values (see Section 2.11.1.2 (on page 525)).

29413        Results are undefined if *posix\_trace\_attr\_init()* is called specifying an already initialized *attr*  
 29414        attributes object.

29415        Implementations may add extensions to the trace attributes object structure as permitted in the  
 29416        Base Definitions volume of IEEE Std 1003.1-200x, Chapter 2, Conformance.

29417        The resulting attributes object (possibly modified by setting individual attributes values), when  
 29418        used by *posix\_trace\_create()*, defines the attributes of the trace stream created. A single attributes  
 29419        object can be used in multiple calls to *posix\_trace\_create()*. After one or more trace streams have  
 29420        been created using an attributes object, any function affecting that attributes object, including  
 29421        destruction, shall not affect any trace stream previously created. An initialized attributes object  
 29422        also serves to receive the attributes of an existing trace stream or trace log when calling the  
 29423        *posix\_trace\_get\_attr()* function.

29424 **RETURN VALUE**

29425        Upon successful completion, these functions shall return a value of zero. Otherwise, they shall  
 29426        return the corresponding error number.

29427 **ERRORS**

29428        The *posix\_trace\_attr\_destroy()* function may fail if:

29429        [EINVAL]        The value of *attr* is invalid.

29430        The *posix\_trace\_attr\_init()* function shall fail if:

29431        [ENOMEM]       Insufficient memory exists to initialize the trace attributes object.

29432 **EXAMPLES**

29433        None.

29434 **APPLICATION USAGE**

29435        None.

29436 **RATIONALE**

29437        None.

29438 **FUTURE DIRECTIONS**

29439       None.

29440 **SEE ALSO**

29441       *posix\_trace\_create()*, *posix\_trace\_get\_attr()*, *uname()*, the Base Definitions volume of  
29442       IEEE Std 1003.1-200x, <**trace.h**>

29443 **CHANGE HISTORY**

29444       First released in Issue 6. Derived from IEEE Std 1003.1q-2000. |

29445       IEEE PASC Interpretation 1003.1 #123 is applied. |

29446 **NAME**

29447 `posix_trace_attr_getclockres`, `posix_trace_attr_getcreatetime`, `posix_trace_attr_getgenversion`,  
 29448 `posix_trace_attr_getname`, `posix_trace_attr_setname` — retrieve and set information about a  
 29449 trace stream (**TRACING**)

29450 **SYNOPSIS**

```
29451 TRC #include <time.h>
29452 #include <trace.h>

29453 int posix_trace_attr_getclockres(const trace_attr_t *attr,
29454     struct timespec *resolution);
29455 int posix_trace_attr_getcreatetime(const trace_attr_t *attr,
29456     struct timespec *createtime);

29457 #include <trace.h>

29458 int posix_trace_attr_getgenversion(const trace_attr_t *attr,
29459     char *genversion);
29460 int posix_trace_attr_getname(const trace_attr_t *attr,
29461     char *tracename);
29462 int posix_trace_attr_setname(trace_attr_t *attr,
29463     const char *tracename);
29464
```

29465 **DESCRIPTION**

29466 The `posix_trace_attr_getclockres()` function shall copy the clock resolution of the clock used to  
 29467 generate timestamps from the *clock-resolution* attribute of the attributes object pointed to by the  
 29468 *attr* argument into the structure pointed to by the *resolution* argument.

29469 The `posix_trace_attr_getcreatetime()` function shall copy the trace stream creation time from the  
 29470 *creation-time* attribute of the attributes object pointed to by the *attr* argument into the structure  
 29471 pointed to by the *createtime* argument. The *creation-time* attribute shall represent the time of  
 29472 creation of the trace stream.

29473 The `posix_trace_attr_getgenversion()` function shall copy the string containing version information  
 29474 from the *generation-version* attribute of the attributes object pointed to by the *attr* argument into  
 29475 the string pointed to by the *genversion* argument. The *genversion* argument shall be the address of  
 29476 a character array which can store at least {TRACE\_NAME\_MAX} characters.

29477 The `posix_trace_attr_getname()` function shall copy the string containing the trace name from the  
 29478 *trace-name* attribute of the attributes object pointed to by the *attr* argument into the string  
 29479 pointed to by the *tracename* argument. The *tracename* argument shall be the address of a character  
 29480 array which can store at least {TRACE\_NAME\_MAX} characters.

29481 The `posix_trace_attr_setname()` function shall set the name in the *trace-name* attribute of the  
 29482 attributes object pointed to by the *attr* argument, using the trace name string supplied by the  
 29483 *tracename* argument. If the supplied string contains more than {TRACE\_NAME\_MAX}  
 29484 characters, the name copied into the *trace-name* attribute may be truncated to one less than the  
 29485 length of {TRACE\_NAME\_MAX} characters. The default value is a null string.

29486 **RETURN VALUE**

29487 Upon successful completion, these functions shall return a value of zero. Otherwise, they shall  
 29488 return the corresponding error number.

29489 If successful, the `posix_trace_attr_getclockres()` function stores the *clock-resolution* attribute value  
 29490 in the object pointed to by *resolution*. Otherwise, the content of this object is unspecified.

29491 If successful, the *posix\_trace\_attr\_getcreatetime()* function stores the trace stream creation time in  
 29492 the object pointed to by *createtime*. Otherwise, the content of this object is unspecified.

29493 If successful, the *posix\_trace\_attr\_getgenversion()* function stores the trace version information in  
 29494 the string pointed to by *genversion*. Otherwise, the content of this string is unspecified.

29495 If successful, the *posix\_trace\_attr\_getname()* function stores the trace name in the string pointed  
 29496 to by *tracename*. Otherwise, the content of this string is unspecified.

29497 **ERRORS**

29498 The *posix\_trace\_attr\_getclockres()*, *posix\_trace\_attr\_getcreatetime()*, *posix\_trace\_attr\_getgenversion()*,  
 29499 and *posix\_trace\_attr\_getname()* functions may fail if:

29500 [EINVAL] The value specified by one of the arguments is invalid.

29501 **EXAMPLES**

29502 None.

29503 **APPLICATION USAGE**

29504 None.

29505 **RATIONALE**

29506 None.

29507 **FUTURE DIRECTIONS**

29508 None.

29509 **SEE ALSO**

29510 *posix\_trace\_attr\_init()*, *posix\_trace\_create()*, *posix\_trace\_get\_attr()*, *uname()*, the Base Definitions  
 29511 volume of IEEE Std 1003.1-200x, <**time.h**>, <**trace.h**>

29512 **CHANGE HISTORY**

29513 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

29514 **NAME**

29515        posix\_trace\_attr\_getcreatetime — retrieve and set information about a trace stream (**TRACING**)

29516 **SYNOPSIS**

29517 TRC     #include <time.h>

29518        #include <trace.h>

```
29519        int posix_trace_attr_getcreatetime(const trace_attr_t *attr,  
29520                                          struct timespec *createtime);
```

29521

29522 **DESCRIPTION**

29523        Refer to *posix\_trace\_attr\_getclockres()*.

29524 **NAME**

29525        posix\_trace\_attr\_getgenversion — retrieve and set information about a trace stream  
29526        **(TRACING)**

29527 **SYNOPSIS**

29528 TRC    #include <trace.h>

```
29529        int posix_trace_attr_getgenversion(const trace_attr_t *attr,  
29530        char *genversion);
```

29531

29532 **DESCRIPTION**

29533        Refer to *posix\_trace\_attr\_getclockres()*.

## 29534 NAME

29535 posix\_trace\_attr\_getinherited, posix\_trace\_attr\_getlogfullpolicy,  
 29536 posix\_trace\_attr\_getstreamfullpolicy, posix\_trace\_attr\_setinherited,  
 29537 posix\_trace\_attr\_setlogfullpolicy, posix\_trace\_attr\_setstreamfullpolicy — retrieve and set the  
 29538 behavior of a trace stream (**TRACING**)

## 29539 SYNOPSIS

```
29540 TRC #include <trace.h>
29541 TRC TRI int posix_trace_attr_getinherited(const trace_attr_t *restrict attr,
29542 int *restrict inheritancepolicy);
29543 TRC TRL int posix_trace_attr_getlogfullpolicy(const trace_attr_t *restrict attr,
29544 int *restrict logpolicy);
29545 TRC int posix_trace_attr_getstreamfullpolicy(const trace_attr_t *attr,
29546 int *streampolicy);
29547 TRC TRI int posix_trace_attr_setinherited(trace_attr_t *attr,
29548 int inheritancepolicy);
29549 TRC TRL int posix_trace_attr_setlogfullpolicy(trace_attr_t *attr,
29550 int logpolicy);
29551 TRC int posix_trace_attr_setstreamfullpolicy(trace_attr_t *attr,
29552 int streampolicy);
29553
```

## 29554 DESCRIPTION

29555 TRI The *posix\_trace\_attr\_getinherited()* and *posix\_trace\_attr\_setinherited()* functions, respectively, shall  
 29556 get and set the inheritance policy stored in the *inheritance* attribute for traced processes across  
 29557 the *fork()* and *spawn()* operations. The *inheritance* attribute of the attributes object pointed to by  
 29558 the *attr* argument shall be set to one of the following values defined by manifest constants in the  
 29559 **<trace.h>** header:

## 29560 POSIX\_TRACE\_CLOSE\_FOR\_CHILD

29561 After a *fork()* or *spawn()* operation, the child shall not be traced, and tracing of the parent  
 29562 shall continue.

## 29563 POSIX\_TRACE\_INHERITED

29564 After a *fork()* or *spawn()* operation, if the parent is being traced, its child shall be  
 29565 concurrently traced using the same trace stream.

29566 The default value for the *inheritance* attribute is **POSIX\_TRACE\_CLOSE\_FOR\_CHILD**.

29567 TRL The *posix\_trace\_attr\_getlogfullpolicy()* and *posix\_trace\_attr\_setlogfullpolicy()* functions, |  
 29568 respectively, shall get and set the trace log full policy stored in the *log-full-policy* attribute of the |  
 29569 attributes object pointed to by the *attr* argument.

29570 The *log-full-policy* attribute shall be set to one of the following values defined by manifest  
 29571 constants in the **<trace.h>** header:

## 29572 POSIX\_TRACE\_LOOP

29573 The trace log shall loop until the associated trace stream is stopped. This policy means that  
 29574 when the trace log gets full, the file system shall reuse the resources allocated to the oldest  
 29575 trace events that were recorded. In this way, the trace log will always contain the most  
 29576 recent trace events flushed.

## 29577 POSIX\_TRACE\_UNTIL\_FULL

29578 The trace stream shall be flushed to the trace log until the trace log is full. This condition can  
 29579 be deduced from the *posix\_log\_full\_status* member status (see the *posix\_trace\_status\_info()*  
 29580 function). The last recorded trace event shall be the **POSIX\_TRACE\_STOP** trace event.

29581 POSIX\_TRACE\_APPEND  
 29582 The associated trace stream shall be flushed to the trace log without log size limitation. If  
 29583 the application specifies POSIX\_TRACE\_APPEND, the implementation shall ignore the  
 29584 *log-max-size* attribute.

29585 The default value for the *log-full-policy* attribute is POSIX\_TRACE\_LOOP.

29586 The *posix\_trace\_attr\_getstreamfullpolicy()* and *posix\_trace\_attr\_setstreamfullpolicy()* functions,  
 29587 respectively, shall get and set the trace stream full policy stored in the *stream-full-policy* attribute  
 29588 of the attributes object pointed to by the *attr* argument.

29589 The *stream-full-policy* attribute shall be set to one of the following values defined by manifest  
 29590 constants in the <trace.h> header:

29591 POSIX\_TRACE\_LOOP  
 29592 The trace stream shall loop until explicitly stopped by the *posix\_trace\_stop()* function. This  
 29593 policy means that when the trace stream is full, the trace system shall reuse the resources  
 29594 allocated to the oldest trace events recorded. In this way, the trace stream will always  
 29595 contain the most recent trace events recorded.

29596 POSIX\_TRACE\_UNTIL\_FULL  
 29597 The trace stream will run until the trace stream resources are exhausted. Then the trace  
 29598 stream will stop. This condition can be deduced from *posix\_stream\_status* and  
 29599 *posix\_stream\_full\_status* statuses (see the *posix\_trace\_status\_info()* function). When this trace  
 29600 stream is read, a POSIX\_TRACE\_STOP trace event shall be reported after reporting the last  
 29601 recorded trace event. The trace system shall reuse the resources allocated to any trace  
 29602 events already reported—see the *posix\_trace\_getnext\_event()*, *posix\_trace\_trygetnext\_event()*,  
 29603 and *posix\_trace\_timedgetnext\_event()* functions—or already flushed for an active trace stream  
 29604 with log if the Trace Log option is supported; see the *posix\_trace\_flush()* function. The trace  
 29605 system shall restart the trace stream when it is empty and may restart it sooner. A  
 29606 POSIX\_TRACE\_START trace event shall be reported before reporting the next recorded  
 29607 trace event.

29608 TRL POSIX\_TRACE\_FLUSH  
 29609 If the Trace Log option is supported, this policy is identical to the  
 29610 POSIX\_TRACE\_UNTIL\_FULL trace stream full policy except that the trace stream shall be  
 29611 flushed regularly as if *posix\_trace\_flush()* had been explicitly called. Defining this policy for  
 29612 an active trace stream without log shall be invalid.

29613 The default value for the *stream-full-policy* attribute shall be POSIX\_TRACE\_LOOP for an active  
 29614 trace stream without log.

29615 TRL If the Trace Log option is supported, the default value for the *stream-full-policy* attribute shall be  
 29616 POSIX\_TRACE\_FLUSH for an active trace stream with log.

29617 **RETURN VALUE**

29618 Upon successful completion, these functions shall return a value of zero. Otherwise, they shall  
 29619 return the corresponding error number.

29620 TRI If successful, the *posix\_trace\_attr\_getinherited()* function shall store the *inheritance* attribute value  
 29621 in the object pointed to by *inheritancepolicy*. Otherwise, the content of this object is undefined.

29622 TRL If successful, the *posix\_trace\_attr\_getlogfullpolicy()* function shall store the *log-full-policy* attribute  
 29623 value in the object pointed to by *logpolicy*. Otherwise, the content of this object is undefined.

29624 If successful, the *posix\_trace\_attr\_getstreamfullpolicy()* function shall store the *stream-full-policy*  
 29625 attribute value in the object pointed to by *streampolicy*. Otherwise, the content of this object is  
 29626 undefined.

29627 **ERRORS**

29628           These functions may fail if:

29629           [EINVAL]           The value specified by at least one of the arguments is invalid.

29630 **EXAMPLES**

29631           None.

29632 **APPLICATION USAGE**

29633           None.

29634 **RATIONALE**

29635           None.

29636 **FUTURE DIRECTIONS**

29637           None.

29638 **SEE ALSO**

29639           *fork()*, *posix\_trace\_attr\_init()*, *posix\_trace\_create()*, *posix\_trace\_flush()*, *posix\_trace\_get\_attr()*,  
29640           *posix\_trace\_getnext\_event()*, *posix\_trace\_start()*, **posix\_trace\_status\_info Structure**,  
29641           *posix\_trace\_timedgetnext\_event()*, the Base Definitions volume of IEEE Std 1003.1-200x, <**trace.h**>

29642 **CHANGE HISTORY**

29643           First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

29644 **NAME**

29645           posix\_trace\_attr\_getlogfullpolicy — retrieve and set the behavior of a trace stream (**TRACING**)

29646 **SYNOPSIS**

29647 TRC       #include <trace.h>

29648 TRC TRL   int posix\_trace\_attr\_getlogfullpolicy(const trace\_attr\_t \*restrict attr,  
29649   int \*restrict logpolicy);

29650

29651 **DESCRIPTION**

29652           Refer to *posix\_trace\_attr\_getinherited()*.

## 29653 NAME

29654 posix\_trace\_attr\_getlogsize, posix\_trace\_attr\_getmaxdatasize,  
 29655 posix\_trace\_attr\_getmaxsystemeventsize, posix\_trace\_attr\_getmaxusereventsize,  
 29656 posix\_trace\_attr\_getstreamsize, posix\_trace\_attr\_setlogsize, posix\_trace\_attr\_setmaxdatasize,  
 29657 posix\_trace\_attr\_setstreamsize — retrieve and set trace stream size attributes (TRACING)

## 29658 SYNOPSIS

```
29659 TRC    #include <sys/types.h>
29660        #include <trace.h>

29661 TRC TRL int posix_trace_attr_getlogsize(const trace_attr_t *restrict attr,
29662        size_t *restrict logsize);
29663 TRC    int posix_trace_attr_getmaxdatasize(const trace_attr_t *restrict attr,
29664        size_t *restrict maxdatasize);
29665        int posix_trace_attr_getmaxsystemeventsize(
29666        const trace_attr_t *restrict attr,
29667        size_t *restrict eventsize);
29668        int posix_trace_attr_getmaxusereventsize(
29669        const trace_attr_t *restrict attr,
29670        size_t data_len, size_t *restrict eventsize);
29671        int posix_trace_attr_getstreamsize(const trace_attr_t *restrict attr,
29672        size_t *restrict streamsize);
29673 TRC TRL int posix_trace_attr_setlogsize(trace_attr_t *attr,
29674        size_t logsize);
29675 TRC    int posix_trace_attr_setmaxdatasize(trace_attr_t *attr,
29676        size_t maxdatasize);
29677        int posix_trace_attr_setstreamsize(trace_attr_t *attr,
29678        size_t streamsize);
29679
```

## 29680 DESCRIPTION

29681 TRL The *posix\_trace\_attr\_getlogsize()* function shall copy the log size, in bytes, from the *log-max-size*  
 29682 attribute of the attributes object pointed to by the *attr* argument into the variable pointed to by  
 29683 the *logsize* argument. This log size is the maximum total of bytes that shall be allocated for  
 29684 system and user trace events in the trace log. The default value for the *log-max-size* attribute is  
 29685 implementation-defined.

29686 The *posix\_trace\_attr\_setlogsize()* function shall set the maximum allowed size, in bytes, in the  
 29687 *log-max-size* attribute of the attributes object pointed to by the *attr* argument, using the size value  
 29688 supplied by the *logsize* argument.

29689 The trace log size shall be used if the *log-full-policy* attribute is set to `POSIX_TRACE_LOOP` or  
 29690 `POSIX_TRACE_UNTIL_FULL`. If the *log-full-policy* attribute is set to `POSIX_TRACE_APPEND`,  
 29691 the implementation shall ignore the *log-max-size* attribute.

29692 The *posix\_trace\_attr\_getmaxdatasize()* function shall copy the maximum user trace event data  
 29693 size, in bytes, from the *max-data-size* attribute of the attributes object pointed to by the *attr*  
 29694 argument into the variable pointed to by the *maxdatasize* argument. The default value for the  
 29695 *max-data-size* attribute is implementation-defined.

29696 The *posix\_trace\_attr\_getmaxsystemeventsize()* function shall calculate the maximum memory size,  
 29697 in bytes, required to store a single system trace event. This value is calculated for the trace  
 29698 stream attributes object pointed to by the *attr* argument and is returned in the variable pointed  
 29699 to by the *eventsize* argument.

29700 The values returned as the maximum memory sizes of the user and system trace events shall be  
 29701 such that if the sum of the maximum memory sizes of a set of the trace events that may be  
 29702 recorded in a trace stream is less than or equal to the *stream-min-size* attribute of that trace  
 29703 stream, the system provides the necessary resources for recording all those trace events, without  
 29704 loss.

29705 The *posix\_trace\_attr\_getmaxusereventsize()* function shall calculate the maximum memory size, in  
 29706 bytes, required to store a single user trace event generated by a call to *posix\_trace\_event()* with a  
 29707 *data\_len* parameter equal to the *data\_len* value specified in this call. This value is calculated for  
 29708 the trace stream attributes object pointed to by the *attr* argument and is returned in the variable  
 29709 pointed to by the *eventsize* argument.

29710 The *posix\_trace\_attr\_getstreamsize()* function shall copy the stream size, in bytes, from the  
 29711 *stream-min-size* attribute of the attributes object pointed to by the *attr* argument into the variable  
 29712 pointed to by the *streamsize* argument.

29713 This stream size is the current total memory size reserved for system and user trace events in the  
 29714 trace stream. The default value for the *stream-min-size* attribute is implementation-defined. The  
 29715 stream size refers to memory used to store trace event records. Other stream data (for example,  
 29716 trace attribute values) shall not be included in this size.

29717 The *posix\_trace\_attr\_setmaxdatasize()* function shall set the maximum allowed size, in bytes, in  
 29718 the *max-data-size* attribute of the attributes object pointed to by the *attr* argument, using the size  
 29719 value supplied by the *maxdatasize* argument. This maximum size is the maximum allowed size  
 29720 for the user data argument which may be passed to *posix\_trace\_event()*. The implementation  
 29721 shall be allowed to truncate data passed to *trace\_user\_event* which is longer than *maxdatasize*.

29722 The *posix\_trace\_attr\_setstreamsize()* function shall set the minimum allowed size, in bytes, in the  
 29723 *stream-min-size* attribute of the attributes object pointed to by the *attr* argument, using the size  
 29724 value supplied by the *streamsize* argument.

29725 **RETURN VALUE**

29726 Upon successful completion, these functions shall return a value of zero. Otherwise, they shall  
 29727 return the corresponding error number.

29728 TRL The *posix\_trace\_attr\_getlogsize()* function stores the maximum trace log allowed size in the object  
 29729 pointed to by *logsize*, if successful.

29730 The *posix\_trace\_attr\_getmaxdatasize()* function stores the maximum trace event record memory  
 29731 size in the object pointed to by *maxdatasize*, if successful.

29732 The *posix\_trace\_attr\_getmaxsystemeventsize()* function stores the maximum memory size to store  
 29733 a single system trace event in the object pointed to by *eventsize*, if successful.

29734 The *posix\_trace\_attr\_getmaxusereventsize()* function stores the maximum memory size to store a  
 29735 single user trace event in the object pointed to by *eventsize*, if successful.

29736 The *posix\_trace\_attr\_getstreamsize()* function stores the maximum trace stream allowed size in  
 29737 the object pointed to by *streamsize*, if successful.

29738 **ERRORS**

29739 These functions may fail if:

29740 [EINVAL] The value specified by one of the arguments is invalid.

29741 **EXAMPLES**

29742 None.

29743 **APPLICATION USAGE**

29744 None.

29745 **RATIONALE**

29746 None.

29747 **FUTURE DIRECTIONS**

29748 None.

29749 **SEE ALSO**29750 *posix\_trace\_attr\_init()*, *posix\_trace\_create()*, *posix\_trace\_event()*, *posix\_trace\_get\_attr()*, the Base

29751 Definitions volume of IEEE Std 1003.1-200x, &lt;sys/types.h&gt;, &lt;trace.h&gt;

29752 **CHANGE HISTORY**

29753 First released in Issue 6. Derived from the IEEE Std 1003.1q-2000.

29754 **NAME**

29755 posix\_trace\_attr\_getmaxdatasize, posix\_trace\_attr\_getmaxsystemeventsize,  
29756 posix\_trace\_attr\_getmaxusereventsize — retrieve and set trace stream size attributes  
29757 **(TRACING)**

29758 **SYNOPSIS**

```
29759 TRC #include <sys/types.h>  
29760 #include <trace.h>
```

```
29761 TRC int posix_trace_attr_getmaxdatasize(const trace_attr_t *restrict attr,  
29762 size_t *restrict maxdatasize);  
29763 int posix_trace_attr_getmaxsystemeventsize(  
29764 const trace_attr_t *restrict attr,  
29765 size_t *restrict eventsize);  
29766 int posix_trace_attr_getmaxusereventsize(  
29767 const trace_attr_t *restrict attr,  
29768 size_t data_len, size_t *restrict eventsize);  
29769
```

29770 **DESCRIPTION**

29771 Refer to *posix\_trace\_attr\_getlogsize()*.

29772 **NAME**

29773 `posix_trace_attr_getname` — retrieve and set information about a trace stream (**TRACING**)

29774 **SYNOPSIS**

29775 TRC `#include <trace.h>`

```
29776 int posix_trace_attr_getname(const trace_attr_t *attr,  
29777 char *tracename);
```

29778

29779 **DESCRIPTION**

29780 Refer to `posix_trace_attr_getclockres()`.

29781 **NAME**

29782        posix\_trace\_attr\_getstreamfullpolicy — retrieve and set the behavior of a trace stream  
29783        **(TRACING)**

29784 **SYNOPSIS**

```
29785 TRC     #include <trace.h>
29786           int posix_trace_attr_getstreamfullpolicy(const trace_attr_t *attr,
29787                                                    int *streampolicy);
29788
```

29789 **DESCRIPTION**

29790        Refer to *posix\_trace\_attr\_getinherited()*.

29791 **NAME**29792        **posix\_trace\_attr\_getstreamsize** — retrieve and set trace stream size attributes (**TRACING**)29793 **SYNOPSIS**

29794 TRC     #include &lt;sys/types.h&gt;

29795        #include &lt;trace.h&gt;

29796        int posix\_trace\_attr\_getstreamsize(const trace\_attr\_t \*restrict attr,  
29797    size\_t \*restrict streamsize);

29798

29799 **DESCRIPTION**29800        Refer to *posix\_trace\_attr\_getlogsize()*.

29801 **NAME**

29802        posix\_trace\_attr\_init — trace stream attributes object initialization (**TRACING**)

29803 **SYNOPSIS**

29804 TRC     #include <trace.h>

29805        int posix\_trace\_attr\_init(trace\_attr\_t \*attr);

29806

29807 **DESCRIPTION**

29808        Refer to *posix\_trace\_attr\_destroy()*.

29809 **NAME**

29810 `posix_trace_attr_setinherited` — retrieve and set the behavior of a trace stream (**TRACING**)

29811 **SYNOPSIS**

29812 TRC `#include <trace.h>`

29813 TRC TRI `int posix_trace_attr_setinherited(trace_attr_t *attr,`  
29814 `int inheritancepolicy);`

29815

29816 **DESCRIPTION**

29817 Refer to `posix_trace_attr_getinherited()`.

29818 **NAME**

29819        posix\_trace\_attr\_setlogfullpolicy — retrieve and set the behavior of a trace stream (**TRACING**)

29820 **SYNOPSIS**

29821 TRC     #include <trace.h>

29822 TRC TRL int posix\_trace\_attr\_setlogfullpolicy(trace\_attr\_t \*attr,  
29823           int logpolicy);

29824

29825 **DESCRIPTION**

29826        Refer to *posix\_trace\_attr\_getinherited()*.

29827 **NAME**29828        posix\_trace\_attr\_setlogsize — retrieve and set trace stream size attributes (**TRACING**)29829 **SYNOPSIS**

29830 TRC     #include &lt;sys/types.h&gt;

29831        #include &lt;trace.h&gt;

29832 TRC TRL int posix\_trace\_attr\_setlogsize(trace\_attr\_t \*attr,  
29833                                   size\_t logsize);

29834

29835 **DESCRIPTION**29836        Refer to *posix\_trace\_attr\_getlogsize()*.

29837 **NAME**

29838        posix\_trace\_attr\_setmaxdatasize — retrieve and set trace stream size attributes (**TRACING**)

29839 **SYNOPSIS**

29840 TRC     #include <sys/types.h>

29841        #include <trace.h>

```
29842        int posix_trace_attr_setmaxdatasize(trace_attr_t *attr,  
29843                                   size_t maxdatasize);
```

29844

29845 **DESCRIPTION**

29846        Refer to *posix\_trace\_attr\_getlogsize()*.

29847 **NAME**

29848        posix\_trace\_attr\_setname — retrieve and set information about a trace stream (**TRACING**)

29849 **SYNOPSIS**

29850 TRC     #include <trace.h>

```
29851        int posix_trace_attr_setname(trace_attr_t *attr,  
29852                                    const char *tracename);
```

29853

29854 **DESCRIPTION**

29855        Refer to *posix\_trace\_attr\_getclockres()*.

29856 **NAME**

29857        posix\_trace\_attr\_setstreamfullpolicy — retrieve and set the behavior of a trace stream  
29858        **(TRACING)**

29859 **SYNOPSIS**

29860 TRC     #include <trace.h>

29861 TRC TRL int posix\_trace\_attr\_setlogfullpolicy(trace\_attr\_t \*attr,  
29862           int logpolicy);

29863

29864 **DESCRIPTION**

29865        Refer to *posix\_trace\_attr\_getinherited()*.

29866 **NAME**

29867        posix\_trace\_attr\_setstreamsize — retrieve and set trace stream size attributes (**TRACING**)

29868 **SYNOPSIS**

29869 TRC     #include <sys/types.h>

29870        #include <trace.h>

29871        int posix\_trace\_attr\_setstreamsize(trace\_attr\_t \*attr,  
29872    size\_t streamsize);

29873

29874 **DESCRIPTION**

29875        Refer to *posix\_trace\_attr\_getlogsize()*.

29876 **NAME**

29877 posix\_trace\_clear — clear trace stream and trace log (**TRACING**)

29878 **SYNOPSIS**

29879 TRC #include <sys/types.h>

29880 #include <trace.h>

29881 int posix\_trace\_clear(trace\_id\_t trid);

29882

29883 **DESCRIPTION**

29884 The *posix\_trace\_clear()* function shall reinitialize the trace stream identified by the argument *trid* as if it were returning from the *posix\_trace\_create()* function, except that the same allocated resources shall be reused, the mapping of trace event type identifiers to trace event names shall be unchanged, and the trace stream status shall remain unchanged (that is, if it was running, it remains running and if it was suspended, it remains suspended).

29889 All trace events in the trace stream recorded before the call to *posix\_trace\_clear()* shall be lost. The *posix\_stream\_full\_status* status shall be set to `POSIX_TRACE_NOT_FULL`. There is no guarantee that all trace events that occurred during the *posix\_trace\_clear()* call are recorded; the behavior with respect to trace points that may occur during this call, is unspecified.

29893 TRL If the Trace Log option is supported and the trace stream has been created with a log, the *posix\_trace\_clear()* function shall reinitialize the trace stream with the same behavior as if the trace stream was created without the log, plus it shall reinitialize the trace log associated with the trace stream identified by the argument *trid* as if it were returning from the *posix\_trace\_create\_withlog()* function, except that the same allocated resources, for the trace log, may be reused and the associated trace stream status remains unchanged. The first trace event recorded in the trace log after the call to *posix\_trace\_clear()* shall be the same as the first trace event recorded in the active trace stream after the call to *posix\_trace\_clear()*. The *posix\_log\_full\_status* status shall be set to `POSIX_TRACE_NOT_FULL`. There is no guarantee that all trace events that occurred during the *posix\_trace\_clear()* call are recorded in the trace log; the behavior with respect to trace points that may occur during this call is unspecified. If the log full policy is `POSIX_TRACE_APPEND`, the effect of a call to this function is unspecified for the trace log associated with the trace stream identified by the *trid* argument.

29906 **RETURN VALUE**

29907 Upon successful completion, the *posix\_trace\_clear()* function shall return a value of zero. 29908 Otherwise, it shall return the corresponding error number.

29909 **ERRORS**

29910 The *posix\_trace\_clear()* function shall fail if:

29911 [EINVAL] The value of the *trid* argument does not correspond to an active trace stream.

29912 **EXAMPLES**

29913 None.

29914 **APPLICATION USAGE**

29915 None.

29916 **RATIONALE**

29917 None.

29918 **FUTURE DIRECTIONS**

29919 None.

29920 **SEE ALSO**

29921 *posix\_trace\_attr\_init()*, *posix\_trace\_create()*, *posix\_trace\_flush()*, *posix\_trace\_get\_attr()*, the Base  
29922 Definitions volume of IEEE Std 1003.1-200x, <sys/types.h>, <trace.h>

29923 **CHANGE HISTORY**

29924 First released in Issue 6. Derived from IEEE Std 1003.1q-2000. |

29925 IEEE PASC Interpretation 1003.1 #123 is applied. |

29926 **NAME**

29927 posix\_trace\_close, posix\_trace\_open, posix\_trace\_rewind — trace log management (**TRACING**)

29928 **SYNOPSIS**

29929 TRC TRL #include <trace.h>

```
29930 int posix_trace_close(trace_id_t trid);
29931 int posix_trace_open(int file_desc, trace_id_t *trid);
29932 int posix_trace_rewind(trace_id_t trid);
```

29934 **DESCRIPTION**

29935 The *posix\_trace\_close()* function shall deallocate the trace log identifier indicated by *trid*, and all  
 29936 of its associated resources. If there is no valid trace log pointed to by the *trid*, this function shall  
 29937 fail.

29938 The *posix\_trace\_open()* function shall allocate the necessary resources and establishes the  
 29939 connection between a trace log identified by the *file\_desc* argument and a trace stream identifier  
 29940 identified by the object pointed to by the *trid* argument. The *file\_desc* argument should be a valid  
 29941 open file descriptor that corresponds to a trace log. The *file\_desc* argument shall be open for  
 29942 reading. The current trace event timestamp, which specifies the timestamp of the trace event  
 29943 that will be read by the next call to *posix\_trace\_getnext\_event()*, shall be set to the timestamp of  
 29944 the oldest trace event recorded in the trace log identified by *trid*.

29945 The *posix\_trace\_open()* function shall return a trace stream identifier in the variable pointed to by  
 29946 the *trid* argument, that may only be used by the following functions:

29947	<i>posix_trace_close()</i>	<i>posix_trace_get_attr()</i>
29948	<i>posix_trace_eventid_equal()</i>	<i>posix_trace_get_status()</i>
29949	<i>posix_trace_eventid_get_name()</i>	<i>posix_trace_getnext_event()</i>
29950	<i>posix_trace_eventtypelist_getnext_id()</i>	<i>posix_trace_rewind()</i>
29951	<i>posix_trace_eventtypelist_rewind()</i>	

29952 In particular, notice that the operations normally used by a trace controller process, such as  
 29953 *posix\_trace\_start()*, *posix\_trace\_stop()*, or *posix\_trace\_shutdown()*, cannot be invoked using the  
 29954 trace stream identifier returned by the *posix\_trace\_open()* function.

29955 The *posix\_trace\_rewind()* function shall reset the current trace event timestamp, which specifies  
 29956 the timestamp of the trace event that will be read by the next call to *posix\_trace\_getnext\_event()*,  
 29957 to the timestamp of the oldest trace event recorded in the trace log identified by *trid*.

29958 **RETURN VALUE**

29959 Upon successful completion, these functions shall return a value of zero. Otherwise, they shall  
 29960 return the corresponding error number.

29961 If successful, the *posix\_trace\_open()* function stores the trace stream identifier value in the object  
 29962 pointed to by *trid*.

29963 **ERRORS**

- 29964 The *posix\_trace\_open()* function shall fail if:
- 29965 [EINTR] The operation was interrupted by a signal and thus no trace log was opened.
  - 29966 [EINVAL] The object pointed to by *file\_desc* does not correspond to a valid trace log.
- 29967 The *posix\_trace\_close()* and *posix\_trace\_rewind()* functions may fail if:
- 29968 [EINVAL] The object pointed to by *trid* does not correspond to a valid trace log.

29969 **EXAMPLES**

29970 None.

29971 **APPLICATION USAGE**

29972 None.

29973 **RATIONALE**

29974 None.

29975 **FUTURE DIRECTIONS**

29976 None.

29977 **SEE ALSO**29978 *posix\_trace\_get\_attr()*, *posix\_trace\_get\_filter()*, *posix\_trace\_getnext\_event()*, the Base Definitions29979 volume of IEEE Std 1003.1-200x, <**trace.h**>29980 **CHANGE HISTORY**

29981 First released in Issue 6. Derived from IEEE Std 1003.1q-2000. |

29982 IEEE PASC Interpretation 1003.1 #123 is applied. |

29983 **NAME**

29984 posix\_trace\_create, posix\_trace\_create\_withlog, posix\_trace\_flush, posix\_trace\_shutdown —  
 29985 trace stream initialization, flush, and shutdown from a process (**TRACING**)

29986 **SYNOPSIS**

```

29987 TRC #include <sys/types.h>
29988 #include <trace.h>

29989 int posix_trace_create(pid_t pid,
29990 const trace_attr_t *restrict attr,
29991 trace_id_t *restrict trid);
29992 TRC TRL int posix_trace_create_withlog(pid_t pid,
29993 const trace_attr_t *restrict attr, int file_desc,
29994 trace_id_t *restrict trid);
29995 int posix_trace_flush(trace_id_t trid);
29996 TRC int posix_trace_shutdown(trace_id_t trid);
29997
    
```

29998 **DESCRIPTION**

29999 The *posix\_trace\_create()* function shall create an active trace stream. It allocates all the resources  
 30000 needed by the trace stream being created for tracing the process specified by *pid* in accordance  
 30001 with the *attr* argument. The *attr* argument represents the initial attributes of the trace stream and  
 30002 shall have been initialized by the function *posix\_trace\_attr\_init()* prior the *posix\_trace\_create()*  
 30003 call. If the argument *attr* is NULL, the default attributes shall be used. The *attr* attributes object  
 30004 shall be manipulated through a set of functions described in the *posix\_trace\_attr* family of  
 30005 functions. If the attributes of the object pointed to by *attr* are modified later, the attributes of the  
 30006 trace stream shall not be affected. The *creation-time* attribute of the newly created trace stream  
 30007 shall be set to the value of the system clock, if the Timers option is not supported, or to the value  
 30008 of the CLOCK\_REALTIME clock, if the Timers option is supported.

30009 The *pid* argument represents the target process to be traced. If the process executing this  
 30010 function does not have appropriate privileges to trace the process identified by *pid*, an error shall  
 30011 be returned. If the *pid* argument is zero, the calling process shall be traced.

30012 The *posix\_trace\_create()* function shall store the trace stream identifier of the new trace stream in  
 30013 the object pointed to by the *trid* argument. This trace stream identifier shall be used in  
 30014 subsequent calls to control tracing. The *trid* argument may only be used by the following  
 30015 functions:

30016	<i>posix_trace_clear()</i>	<i>posix_trace_getnext_event()</i>
30017	<i>posix_trace_eventid_equal()</i>	<i>posix_trace_shutdown()</i>
30018	<i>posix_trace_eventid_get_name()</i>	<i>posix_trace_start()</i>
30019	<i>posix_trace_eventtypelist_getnext_id()</i>	<i>posix_trace_stop()</i>
30020	<i>posix_trace_eventtypelist_rewind()</i>	<i>posix_trace_timedgetnext_event()</i>
30021	<i>posix_trace_get_attr()</i>	<i>posix_trace_trid_eventid_open()</i>
30022	<i>posix_trace_get_status()</i>	<i>posix_trace_trygetnext_event()</i>

30023 TEF If the Trace Event Filter option is supported, the following additional functions may use the *trid*  
 30024 argument:

```

30025 posix_trace_get_filter() posix_trace_set_filter()
    
```

30026

30027 In particular, notice that the operations normally used by a trace analyzer process, such as  
 30028 *posix\_trace\_rewind()* or *posix\_trace\_close()*, cannot be invoked using the trace stream identifier  
 30029 returned by the *posix\_trace\_create()* function.

30030 TEF A trace stream shall be created in a suspended state. If the Trace Event Filter option is  
 30031 supported, its trace event type filter shall be empty.

30032 The *posix\_trace\_create()* function may be called multiple times from the same or different  
 30033 processes, with the system-wide limit indicated by the runtime invariant value  
 30034 {TRACE\_SYS\_MAX}, which has the minimum value {\_POSIX\_TRACE\_SYS\_MAX}.

30035 The trace stream identifier returned by the *posix\_trace\_create()* function in the argument pointed  
 30036 to by *trid* is valid only in the process that made the function call. If it is used from another  
 30037 process, that is a child process, in functions defined in IEEE Std 1003.1-200x, these functions shall  
 30038 return with the error [EINVAL].

30039 TRL The *posix\_trace\_create\_withlog()* function shall be equivalent to *posix\_trace\_create()*, except that it |  
 30040 associates a trace log with this stream. The *file\_desc* argument shall be the file descriptor |  
 30041 designating the trace log destination. The function shall fail if this file descriptor refers to a file |  
 30042 with a file type that is not compatible with the log policy associated with the trace log. The list of |  
 30043 the appropriate file types that are compatible with each log policy is implementation-defined. |

30044 The *posix\_trace\_create\_withlog()* function shall return in the parameter pointed to by *trid* the trace  
 30045 stream identifier, which uniquely identifies the newly created trace stream, and shall be used in  
 30046 subsequent calls to control tracing. The *trid* argument may only be used by the following  
 30047 functions:

30048	<i>posix_trace_clear()</i>	<i>posix_trace_getnext_event()</i>
30049	<i>posix_trace_eventid_equal()</i>	<i>posix_trace_shutdown()</i>
30050	<i>posix_trace_eventid_get_name()</i>	<i>posix_trace_start()</i>
30051	<i>posix_trace_eventtypelist_getnext_id()</i>	<i>posix_trace_stop()</i>
30052	<i>posix_trace_eventtypelist_rewind()</i>	<i>posix_trace_timedgetnext_event()</i>
30053	<i>posix_trace_flush()</i>	<i>posix_trace_trid_eventid_open()</i>
30054	<i>posix_trace_get_attr()</i>	<i>posix_trace_trygetnext_event()</i>
30055	<i>posix_trace_get_status()</i>	

30056

30057 TRL TEF If the Trace Event Filter option is supported, the following additional functions may use the *trid*  
 30058 argument:

30059 *posix\_trace\_get\_filter()*    *posix\_trace\_set\_filter()*

30060

30061 TRL In particular, notice that the operations normally used by a trace analyzer process, such as  
 30062 *posix\_trace\_rewind()* or *posix\_trace\_close()*, cannot be invoked using the trace stream identifier  
 30063 returned by the *posix\_trace\_create\_withlog()* function.

30064 The *posix\_trace\_flush()* function shall initiate a flush operation which copies the contents of the |  
 30065 trace stream identified by the argument *trid* into the trace log associated with the trace stream at |  
 30066 the creation time. If no trace log has been associated with the trace stream pointed to by *trid*, this |  
 30067 function shall return an error. The termination of the flush operation can be polled by the |  
 30068 *posix\_trace\_get\_status()* function. During the flush operation, it shall be possible to trace new |  
 30069 trace events up to the point when the trace stream becomes full. After flushing is completed, the |  
 30070 space used by the flushed trace events shall be available for tracing new trace events.

30071 If flushing the trace stream causes the resulting trace log to become full, the trace log full policy  
 30072 shall be applied. If the trace *log-full-policy* attribute is set, the following occurs:

30073 POSIX\_TRACE\_UNTIL\_FULL  
 30074 The trace events that have not yet been flushed shall be discarded. |

30075 POSIX\_TRACE\_LOOP  
 30076 The trace events that have not yet been flushed shall be written to the beginning of the trace |  
 30077 log, overwriting previous trace events stored there.

30078 POSIX\_TRACE\_APPEND  
 30079 The trace events that had not yet been flushed shall be appended to the trace log.  
 30080

30081 The *posix\_trace\_shutdown()* function shall stop the tracing of trace events in the trace stream  
 30082 identified by *trid*, as if *posix\_trace\_stop()* had been invoked. The *posix\_trace\_shutdown()* function  
 30083 shall free all the resources associated with the trace stream.

30084 The *posix\_trace\_shutdown()* function shall not return until all the resources associated with the  
 30085 trace stream have been freed. When the *posix\_trace\_shutdown()* function returns, the *trid*  
 30086 argument becomes an invalid trace stream identifier. A call to this function shall unconditionally  
 30087 deallocate the resources regardless of whether all trace events have been retrieved by the  
 30088 analyzer process. Any thread blocked on one of the *trace\_getnext\_event()* functions (which  
 30089 specified this *trid*) before this call is unblocked with the error [EINVAL].

30090 If the process exits, invokes an *exec()* call, or is terminated, the trace streams that the process had  
 30091 created and that have not yet been shut down, shall be automatically shut down as if an explicit  
 30092 call were made to the *posix\_trace\_shutdown()* function.

30093 TRL For an active trace stream with log, when the *posix\_trace\_shutdown()* function is called, all trace  
 30094 events that have not yet been flushed to the trace log shall be flushed, as in the  
 30095 *posix\_trace\_flush()* function, and the trace log shall be closed.

30096 When a trace log is closed, all the information that may be retrieved later from the trace log |  
 30097 through the trace interface shall have been written to the trace log. This information includes the |  
 30098 trace attributes, the list of trace event types (with the mapping between trace event names and |  
 30099 trace event type identifiers), and the trace status.

30100 In addition, unspecified information shall be written to the trace log to allow detection of a valid |  
 30101 trace log during the *posix\_trace\_open()* operation. |

30102 The *posix\_trace\_shutdown()* function shall not return until all trace events have been flushed.

30103 **RETURN VALUE**

30104 Upon successful completion, these functions shall return a value of zero. Otherwise, they shall  
 30105 return the corresponding error number.

30106 TRL The *posix\_trace\_create()* and *posix\_trace\_create\_withlog()* functions store the trace stream identifier  
 30107 value in the object pointed to by *trid*, if successful.

30108 **ERRORS**

30109 TRL The *posix\_trace\_create()* and *posix\_trace\_create\_withlog()* functions shall fail if:

30110 [EAGAIN] No more trace streams can be started now. {TRACE\_SYS\_MAX} has been  
 30111 exceeded.

30112 [EINTR] The operation was interrupted by a signal. No trace stream was created.

30113 [EINVAL] One or more of the trace parameters specified by the *attr* parameter is invalid.

30114	[ENOMEM]	The implementation does not currently have sufficient memory to create the trace stream with the specified parameters.
30115		
30116	[EPERM]	The caller does not have appropriate privilege to trace the process specified by <i>pid</i> .
30117		
30118	[ESRCH]	The <i>pid</i> argument does not refer to an existing process.
30119	TRL	The <i>posix_trace_create_withlog()</i> function shall fail if:
30120	[EBADF]	The <i>file_desc</i> argument is not a valid file descriptor open for writing.
30121	[EINVAL]	The <i>file_desc</i> argument refers to a file with a file type that does not support the log policy associated with the trace log.
30122		
30123	[ENOSPC]	No space left on device. The device corresponding to the argument <i>file_desc</i> does not contain the space required to create this trace log.
30124		
30125		
30126	TRL	The <i>posix_trace_flush()</i> and <i>posix_trace_shutdown()</i> functions shall fail if:
30127	[EINVAL]	The value of the <i>trid</i> argument does not correspond to an active trace stream with log.
30128		
30129	[EFBIG]	The trace log file has attempted to exceed an implementation-defined maximum file size.
30130		
30131	[ENOSPC]	No space left on device.
30132		

**30133 EXAMPLES**

30134 None.

**30135 APPLICATION USAGE**

30136 None.

**30137 RATIONALE**

30138 None.

**30139 FUTURE DIRECTIONS**

30140 None.

**30141 SEE ALSO**

30142 *clock\_getres()*, *exec*, *posix\_trace\_attr\_init()*, *posix\_trace\_clear()*, *posix\_trace\_close()*,  
 30143 *posix\_trace\_eventid\_equal()*, *posix\_trace\_eventtypelist\_getnext\_id()*, *posix\_trace\_flush()*,  
 30144 *posix\_trace\_get\_attr()*, *posix\_trace\_get\_filter()*, *posix\_trace\_get\_status()*, *posix\_trace\_getnext\_event()*,  
 30145 *posix\_trace\_open()*, *posix\_trace\_rewind()*, *posix\_trace\_set\_filter()*, *posix\_trace\_shutdown()*,  
 30146 *posix\_trace\_start()*, *posix\_trace\_timedgetnext\_event()*, *posix\_trace\_trid\_eventid\_open()*,  
 30147 *posix\_trace\_start()*, *time()*, the Base Definitions volume of IEEE Std 1003.1-200x, <sys/types.h>,  
 30148 <trace.h>

**30149 CHANGE HISTORY**

30150 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

30151 NAME

30152 posix\_trace\_event, posix\_trace\_eventid\_open — trace functions for instrumenting application  
 30153 code (TRACING)

30154 SYNOPSIS

```
30155 TRC #include <sys/types.h>
30156 #include <trace.h>

30157 void posix_trace_event(trace_event_id_t event_id,
30158     const void *restrictdata_ptr, size_t data_len);
30159 int posix_trace_eventid_open(const char *restrict event_name,
30160     trace_event_id_t *restrict event_id);
30161
```

30162 DESCRIPTION

30163 The *posix\_trace\_event()* function shall record the *event\_id* and the user data pointed to by *data\_ptr*  
 30164 in the trace stream into which the calling process is being traced and in which *event\_id* is not  
 30165 filtered out. If the total size of the user trace event data represented by *data\_len* is not greater  
 30166 than the declared maximum size for user trace event data, then the *truncation-status* attribute of  
 30167 the trace event recorded is POSIX\_TRACE\_NOT\_TRUNCATED. Otherwise, the user trace event  
 30168 data is truncated to this declared maximum size and the *truncation-status* attribute of the trace  
 30169 event recorded is POSIX\_TRACE\_TRUNCATED\_RECORD.

30170 If there is no trace stream created for the process or if the created trace stream is not running or if  
 30171 the trace event specified by *event\_id* is filtered out in the trace stream, the *posix\_trace\_event()*  
 30172 function shall have no effect.

30173 The *posix\_trace\_eventid\_open()* function shall associate a user trace event name with a trace event  
 30174 type identifier for the calling process. The trace event name is the string pointed to by the  
 30175 argument *event\_name*. It shall have a maximum of {TRACE\_EVENT\_NAME\_MAX} characters  
 30176 (which has the minimum value {POSIX\_TRACE\_EVENT\_NAME\_MAX}). The number of user  
 30177 trace event type identifiers that can be defined for any given process is limited by the maximum  
 30178 value {TRACE\_USER\_EVENT\_MAX}, which has the minimum value  
 30179 {POSIX\_TRACE\_USER\_EVENT\_MAX}.

30180 If the Trace Inherit option is not supported, the *posix\_trace\_eventid\_open()* function shall  
 30181 associate the user trace event name pointed to by the *event\_name* argument with a trace event  
 30182 type identifier that is unique for the traced process, and is returned in the variable pointed to by  
 30183 the *event\_id* argument. If the user trace event name has already been mapped for the traced  
 30184 process, then the previously assigned trace event type identifier shall be returned. If the per-  
 30185 process user trace event name limit represented by {TRACE\_USER\_EVENT\_MAX} has been  
 30186 reached, the pre-defined POSIX\_TRACE\_UNNAMED\_USEREVENT (see Table 2-7 (on page  
 30187 529)) user trace event shall be returned.

30188 TRI If the Trace Inherit option is supported, the *posix\_trace\_eventid\_open()* function shall associate the  
 30189 user trace event name pointed to by the *event\_name* argument with a trace event type identifier  
 30190 that is unique for all the processes being traced in this same trace stream, and is returned in the  
 30191 variable pointed to by the *event\_id* argument. If the user trace event name has already been  
 30192 mapped for the traced processes, then the previously assigned trace event type identifier shall be  
 30193 returned. If the per-process user trace event name limit represented by  
 30194 {TRACE\_USER\_EVENT\_MAX} has been reached, the pre-defined  
 30195 POSIX\_TRACE\_UNNAMED\_USEREVENT (Table 2-7 (on page 529)) user trace event shall be  
 30196 returned.

30197 **Note:** The above procedure, together with the fact that multiple processes can only be traced into the  
 30198 same trace stream by inheritance, ensure that all the processes that are traced into a trace  
 30199 stream have the same mapping of trace event names to trace event type identifiers.

30200

30201 If there is no trace stream created, the *posix\_trace\_eventid\_open()* function shall store this  
 30202 information for future trace streams created for this process.

**30203 RETURN VALUE**

30204 No return value is defined for the *posix\_trace\_event()* function.

30205 Upon successful completion, the *posix\_trace\_eventid\_open()* function shall return a value of zero.  
 30206 Otherwise, it shall return the corresponding error number. The *posix\_trace\_eventid\_open()*  
 30207 function stores the trace event type identifier value in the object pointed to by *event\_id*, if  
 30208 successful.

**30209 ERRORS**

30210 The *posix\_trace\_eventid\_open()* function shall fail if:

30211 [ENAMETOOLONG]

30212 The size of the name pointed to by *event\_name* argument was longer than the  
 30213 implementation-defined value {TRACE\_EVENT\_NAME\_MAX}.

**30214 EXAMPLES**

30215 None.

**30216 APPLICATION USAGE**

30217 None.

**30218 RATIONALE**

30219 None.

**30220 FUTURE DIRECTIONS**

30221 None.

**30222 SEE ALSO**

30223 *posix\_trace\_start()*, *posix\_trace\_trid\_eventid\_open()*, the Base Definitions volume of  
 30224 IEEE Std 1003.1-200x, <sys/types.h>, <trace.h>

**30225 CHANGE HISTORY**

30226 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

30227 IEEE PASC Interpretation 1003.1 #123 is applied.

30228 IEEE PASC Interpretation 1003.1 #127 is applied, correcting some editorial errors in the names of  
 30229 the *posix\_trace\_eventid\_open()* function and the *event\_id* argument.

30230 NAME

30231 posix\_trace\_eventid\_equal, posix\_trace\_eventid\_get\_name, posix\_trace\_trid\_eventid\_open —  
 30232 manipulate trace event type identifier (TRACING)

30233 SYNOPSIS

```
30234 TRC #include <trace.h>

30235 int posix_trace_eventid_equal(trace_id_t trid, trace_event_id_t event1, |
30236 trace_event_id_t event2); |
30237 int posix_trace_eventid_get_name(trace_id_t trid, |
30238 trace_event_id_t event, char *event_name); |
30239 TRC TEF int posix_trace_trid_eventid_open(trace_id_t trid, |
30240 const char *restrict event_name, |
30241 trace_event_id_t *restrict event); |
30242
```

30243 DESCRIPTION

30244 The *posix\_trace\_eventid\_equal()* function shall compare the trace event type identifiers *event1* and  
 30245 *event2* from the same trace stream or the same trace log identified by the *trid* argument. If the  
 30246 trace event type identifiers *event1* and *event2* are from different trace streams, the return value  
 30247 shall be unspecified.

30248 The *posix\_trace\_eventid\_get\_name()* function shall return in the argument pointed to by  
 30249 *event\_name*, the trace event name associated with the trace event type identifier identified by the  
 30250 argument *event*, for the trace stream or for the trace log identified by the *trid* argument. The  
 30251 name of the trace event shall have a maximum of {TRACE\_EVENT\_NAME\_MAX} characters  
 30252 (which has the minimum value {\_POSIX\_TRACE\_EVENT\_NAME\_MAX}). Successive calls to  
 30253 this function with the same trace event type identifier and the same trace stream identifier shall  
 30254 return the same event name.

30255 TEF The *posix\_trace\_trid\_eventid\_open()* function shall associate a user trace event name with a trace  
 30256 event type identifier for a given trace stream. The trace stream is identified by the *trid* argument,  
 30257 and it shall be an active trace stream. The trace event name is the string pointed to by the  
 30258 argument *event\_name*. It shall have a maximum of {TRACE\_EVENT\_NAME\_MAX} characters  
 30259 (which has the minimum value {\_POSIX\_TRACE\_EVENT\_NAME\_MAX}). The number of user  
 30260 trace event type identifiers that can be defined for any given process is limited by the maximum  
 30261 value {TRACE\_USER\_EVENT\_MAX}, which has the minimum value  
 30262 {\_POSIX\_TRACE\_USER\_EVENT\_MAX}.

30263 If the Trace Inherit option is not supported, the *posix\_trace\_trid\_eventid\_open()* function shall  
 30264 associate the user trace event name pointed to by the *event\_name* argument with a trace event  
 30265 type identifier that is unique for the process being traced in the trace stream identified by the *trid*  
 30266 argument, and is returned in the variable pointed to by the *event* argument. If the user trace  
 30267 event name has already been mapped for the traced process, then the previously assigned trace  
 30268 event type identifier shall be returned. If the per-process user trace event name limit represented  
 30269 by {TRACE\_USER\_EVENT\_MAX} has been reached, the pre-defined  
 30270 POSIX\_TRACE\_UNNAMED\_USEREVENT (see Table 2-7 (on page 529)) user trace event shall  
 30271 be returned.

30272 TEF TRI If the Trace Inherit option is supported, the *posix\_trace\_trid\_eventid\_open()* function shall  
 30273 associate the user trace event name pointed to by the *event\_name* argument with a trace event  
 30274 type identifier that is unique for all the processes being traced in the trace stream identified by  
 30275 the *trid* argument, and is returned in the variable pointed to by the *event* argument. If the user  
 30276 trace event name has already been mapped for the traced processes, then the previously  
 30277 assigned trace event type identifier shall be returned. If the per-process user trace event name  
 30278 limit represented by {TRACE\_USER\_EVENT\_MAX} has been reached, the pre-defined

30279 POSIX\_TRACE\_UNNAMED\_USEREVENT (see Table 2-7 (on page 529)) user trace event shall  
30280 be returned.

### 30281 RETURN VALUE

30282 TEF Upon successful completion, the *posix\_trace\_eventid\_get\_name()* and  
30283 *posix\_trace\_trid\_eventid\_open()* functions shall return a value of zero. Otherwise, they shall return  
30284 the corresponding error number.

30285 The *posix\_trace\_eventid\_equal()* function shall return a non-zero value if *event1* and *event2* are  
30286 equal; otherwise, a value of zero shall be returned. No errors are defined. If either *event1* or  
30287 *event2* are not valid trace event type identifiers for the trace stream specified by *trid* or if the *trid*  
30288 is invalid, the behavior shall be unspecified.

30289 The *posix\_trace\_eventid\_get\_name()* function stores the trace event name value in the object  
30290 pointed to by *event\_name*, if successful.

30291 TEF The *posix\_trace\_trid\_eventid\_open()* function stores the trace event type identifier value in the  
30292 object pointed to by *event*, if successful.

### 30293 ERRORS

30294 TEF The *posix\_trace\_eventid\_get\_name()* and *posix\_trace\_trid\_eventid\_open()* functions shall fail if:

30295 [EINVAL] The *trid* argument was not a valid trace stream identifier.

30296 TEF The *posix\_trace\_trid\_eventid\_open()* function shall fail if:

30297 TEF [ENAMETOOLONG]

30298 The size of the name pointed to by *event\_name* argument was longer than the  
30299 implementation-defined value {TRACE\_EVENT\_NAME\_MAX}.

30300 The *posix\_trace\_eventid\_get\_name()* function shall fail if:

30301 [EINVAL] The trace event type identifier *event* was not associated with any name.

### 30302 EXAMPLES

30303 None.

### 30304 APPLICATION USAGE

30305 None.

### 30306 RATIONALE

30307 None.

### 30308 FUTURE DIRECTIONS

30309 None.

### 30310 SEE ALSO

30311 *posix\_trace\_event()*, *posix\_trace\_getnext\_event()*, the Base Definitions volume of  
30312 IEEE Std 1003.1-200x, <trace.h>

### 30313 CHANGE HISTORY

30314 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

30315 IEEE PASC Interpretations 1003.1 #123 and #129 are applied.

30316 **NAME**

30317        posix\_trace\_eventid\_get\_name — manipulate trace event type identifier (**TRACING**)

30318 **SYNOPSIS**

30319 TRC     #include <trace.h>

```
30320        int posix_trace_eventid_get_name(trace_id_t trid,  
30321                                        trace_event_id_t event, char *event_name);
```

30322

30323 **DESCRIPTION**

30324        Refer to *posix\_trace\_eventid\_equal()*.

30325 **NAME**30326        posix\_trace\_eventid\_open — trace functions for instrumenting application code (**TRACING**)30327 **SYNOPSIS**

30328 TRC     #include &lt;sys/types.h&gt;

30329        #include &lt;trace.h&gt;

30330        int posix\_trace\_eventid\_open(const char \*restrict event\_name,

30331                                    trace\_event\_id\_t \*restrict event\_id);

30332

30333 **DESCRIPTION**30334        Refer to *posix\_trace\_event()*.

30335 NAME

30336 posix\_trace\_eventset\_add, posix\_trace\_eventset\_del, posix\_trace\_eventset\_empty,  
 30337 posix\_trace\_eventset\_fill, posix\_trace\_eventset\_ismember — manipulate trace event type sets  
 30338 (TRACING)

30339 SYNOPSIS

```
30340 TRC TEF #include <trace.h>

30341 int posix_trace_eventset_add(trace_event_id_t event_id,
30342     trace_event_set_t *set);
30343 int posix_trace_eventset_del(trace_event_id_t event_id,
30344     trace_event_set_t *set);
30345 int posix_trace_eventset_empty(trace_event_set_t *set);
30346 int posix_trace_eventset_fill(trace_event_set_t *set, int what);
30347 int posix_trace_eventset_ismember(trace_event_id_t event_id,
30348     const trace_event_set_t *restrict set,
30349     int *restrict ismember);
30350
```

30351 DESCRIPTION

30352 These primitives manipulate sets of trace event types. They operate on data objects addressable  
 30353 by the application, not on the current trace event filter of any trace stream.

30354 The *posix\_trace\_eventset\_add()* and *posix\_trace\_eventset\_del()* functions, respectively, shall add or  
 30355 delete the individual trace event type specified by the value of the argument *event\_id* to or from  
 30356 the trace event type set pointed to by the argument *set*. Adding a trace event type already in the  
 30357 set or deleting a trace event type not in the set shall not be considered an error.

30358 The *posix\_trace\_eventset\_empty()* function shall initialize the trace event type set pointed to by  
 30359 the *set* argument such that all trace event types defined, both system and user, shall be excluded  
 30360 from the set.

30361 The *posix\_trace\_eventset\_fill()* function shall initialize the trace event type set pointed to by the  
 30362 argument *set*, such that the set of trace event types defined by the argument *what* shall be  
 30363 included in the set. The value of the argument *what* shall consist of one of the following values,  
 30364 as defined in the **<trace.h>** header:

30365 POSIX\_TRACE\_WOPID\_EVENTS  
 30366 All the process-independent implementation-defined system trace event types are included  
 30367 in the set.

30368 POSIX\_TRACE\_SYSTEM\_EVENTS All the implementation-defined system trace event types are  
 30369 included in the set, as are those defined in IEEE Std 1003.1-200x.

30370 POSIX\_TRACE\_ALL\_EVENTS All trace event types defined, both system and user, are included  
 30371 in the set.

30372 Applications shall call either *posix\_trace\_eventset\_empty()* or *posix\_trace\_eventset\_fill()* at least  
 30373 once for each object of type **trace\_event\_set\_t** prior to any other use of that object. If such an  
 30374 object is not initialized in this way, but is nonetheless supplied as an argument to any of the  
 30375 *posix\_trace\_eventset\_add()*, *posix\_trace\_eventset\_del()*, or *posix\_trace\_eventset\_ismember()* functions,  
 30376 the results are undefined.

30377 The *posix\_trace\_eventset\_ismember()* function shall test whether the trace event type specified by  
 30378 the value of the argument *event\_id* is a member of the set pointed to by the argument *set*. The  
 30379 value returned in the object pointed to by *ismember* argument is zero if the trace event type  
 30380 identifier is not a member of the set and a value different from zero if it is a member of the set.

**30381 RETURN VALUE**

30382       Upon successful completion, these functions shall return a value of zero. Otherwise, they shall  
30383       return the corresponding error number.

**30384 ERRORS**

30385       These functions may fail if:

30386       [EINVAL]       The value of one of the arguments is invalid.

**30387 EXAMPLES**

30388       None.

**30389 APPLICATION USAGE**

30390       None.

**30391 RATIONALE**

30392       None.

**30393 FUTURE DIRECTIONS**

30394       None.

**30395 SEE ALSO**

30396       *posix\_trace\_set\_filter()*, *posix\_trace\_trid\_eventid\_open()*, the Base Definitions volume of  
30397       IEEE Std 1003.1-200x, <trace.h>

**30398 CHANGE HISTORY**

30399       First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

30400 **NAME**

30401 posix\_trace\_eventtypelist\_getnext\_id, posix\_trace\_eventtypelist\_rewind — iterate over a  
30402 mapping of trace event types (**TRACING**)

30403 **SYNOPSIS**

```
30404 TRC #include <trace.h>
30405 int posix_trace_eventtypelist_getnext_id(trace_id_t trid,
30406     trace_event_id_t *restrict event, int *restrict unavailable);
30407 int posix_trace_eventtypelist_rewind(trace_id_t trid);
```

30409 **DESCRIPTION**

30410 The first time *posix\_trace\_eventtypelist\_getnext\_id()* is called, the function shall return in the  
30411 variable pointed to by *event* the first trace event type identifier of the list of trace events of the  
30412 trace stream identified by the *trid* argument. Successive calls to  
30413 *posix\_trace\_eventtypelist\_getnext\_id()* return in the variable pointed to by *event* the next trace  
30414 event type identifier in that same list. Each time a trace event type identifier is successfully  
30415 written into the variable pointed to by the *event* argument, the variable pointed to by the  
30416 *unavailable* argument shall be set to zero. When no more trace event type identifiers are  
30417 available, and so none is returned, the variable pointed to by the *unavailable* argument shall be  
30418 set to a value different from zero.

30419 The *posix\_trace\_eventtypelist\_rewind()* function shall reset the next trace event type identifier to  
30420 be read to the first trace event type identifier from the list of trace events used in the trace stream  
30421 identified by *trid*.

30422 **RETURN VALUE**

30423 Upon successful completion, these functions shall return a value of zero. Otherwise, they shall  
30424 return the corresponding error number.

30425 The *posix\_trace\_eventtypelist\_getnext\_id()* function stores the trace event type identifier value in  
30426 the object pointed to by *event*, if successful.

30427 **ERRORS**

30428 These functions shall fail if:  
30429 [EINVAL] The *trid* argument was not a valid trace stream identifier.

30430 **EXAMPLES**

30431 None.

30432 **APPLICATION USAGE**

30433 None.

30434 **RATIONALE**

30435 None.

30436 **FUTURE DIRECTIONS**

30437 None.

30438 **SEE ALSO**

30439 *posix\_trace\_event()*, *posix\_trace\_getnext\_event()*, *posix\_trace\_trid\_eventid\_open()*, the Base  
30440 Definitions volume of IEEE Std 1003.1-200x, <trace.h>

30441 **CHANGE HISTORY**

30442 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

30443 IEEE PASC Interpretations 1003.1 #123 and #129 are applied.

30444 **NAME**30445        posix\_trace\_flush — trace stream flush from a process (**TRACING**)30446 **SYNOPSIS**

30447 TRC     #include &lt;sys/types.h&gt;

30448        #include &lt;trace.h&gt;

30449        int posix\_trace\_flush(trace\_id\_t trid);

30450

30451 **DESCRIPTION**30452        Refer to *posix\_trace\_create()*.

30453 **NAME**

30454 posix\_trace\_get\_attr, posix\_trace\_get\_status — retrieve the trace attributes or trace statuses  
 30455 (TRACING)

30456 **SYNOPSIS**

```
30457 TRC #include <trace.h>
30458
30458 int posix_trace_get_attr(trace_id_t trid, trace_attr_t *attr);
30459 int posix_trace_get_status(trace_id_t trid,
30460 struct posix_trace_status_info *statusinfo);
30461
```

30462 **DESCRIPTION**

30463 The *posix\_trace\_get\_attr()* function shall copy the attributes of the active trace stream identified  
 30464 TRL by *trid* into the object pointed to by the *attr* argument. If the Trace Log option is supported, *trid*  
 30465 may represent a pre-recorded trace log.

30466 The *posix\_trace\_get\_status()* function shall return, in the structure pointed to by the *statusinfo*  
 30467 argument, the current trace status for the trace stream identified by the *trid* argument. These  
 30468 status values returned in the structure pointed to by *statusinfo* shall have been appropriately  
 30469 TRL read to ensure that the returned values are consistent. If the Trace Log option is supported and  
 30470 the *trid* argument refers to a pre-recorded trace stream, the status shall be the status of the  
 30471 completed trace stream.

30472 Each time the *posix\_trace\_get\_status()* function is used, the overrun status of the trace stream  
 30473 TRL shall be reset to POSIX\_TRACE\_NO\_OVERRUN immediately after the call completes. If the  
 30474 Trace Log option is supported, the *posix\_trace\_get\_status()* function shall behave the same as  
 30475 when the option is not supported except for the following differences:

- 30476 • If the *trid* argument refers to a trace stream with log, each time the *posix\_trace\_get\_status()*  
 30477 function is used, the log overrun status of the trace stream shall be reset to  
 30478 POSIX\_TRACE\_NO\_OVERRUN and the *flush\_error* status shall be reset to zero immediately  
 30479 after the call completes.
- 30480 • If the *trid* argument refers to a pre-recorded trace stream, the status returned shall be the  
 30481 status of the completed trace stream and the status values of the trace stream shall not be  
 30482 reset.  
 30483

30484 **RETURN VALUE**

30485 Upon successful completion, these functions shall return a value of zero. Otherwise, they shall  
 30486 return the corresponding error number.

30487 The *posix\_trace\_get\_attr()* function stores the trace attributes in the object pointed to by *attr*, if  
 30488 successful.

30489 The *posix\_trace\_get\_status()* function stores the trace status in the object pointed to by *statusinfo*,  
 30490 if successful.

30491 **ERRORS**

30492 These functions shall fail if:

30493 [EINVAL] The trace stream argument *trid* does not correspond to a valid active trace  
 30494 stream or a valid trace log.

30495 **EXAMPLES**

30496 None.

30497 **APPLICATION USAGE**

30498 None.

30499 **RATIONALE**

30500 None.

30501 **FUTURE DIRECTIONS**

30502 None.

30503 **SEE ALSO**30504 *posix\_trace\_attr\_destroy()*, *posix\_trace\_attr\_init()*, *posix\_trace\_create()*, *posix\_trace\_open()*, the Base

30505 Definitions volume of IEEE Std 1003.1-200x, &lt;trace.h&gt;

30506 **CHANGE HISTORY**

30507 First released in Issue 6. Derived from IEEE Std 1003.1q-2000. |

30508 IEEE PASC Interpretation 1003.1 #123 is applied. |

30509 **NAME**

30510 posix\_trace\_get\_filter, posix\_trace\_set\_filter — retrieve and set filter of an initialized trace  
30511 stream (**TRACING**)

30512 **SYNOPSIS**

```
30513 TRC TEF #include <trace.h>
30514
30514 int posix_trace_get_filter(trace_id_t trid, trace_event_set_t *set);
30515 int posix_trace_set_filter(trace_id_t trid,
30516     const trace_event_set_t *set, int how);
30517
```

30518 **DESCRIPTION**

30519 The *posix\_trace\_get\_filter()* function shall retrieve, into the argument pointed to by *set*, the actual  
30520 trace event filter from the trace stream specified by *trid*.

30521 The *posix\_trace\_set\_filter()* function shall change the set of filtered trace event types after a trace  
30522 stream identified by the *trid* argument is created. This function may be called prior to starting  
30523 the trace stream, or while the trace stream is active. By default, if no call is made to  
30524 *posix\_trace\_set\_filter()*, all trace events shall be recorded (that is, none of the trace event types are  
30525 filtered out).

30526 If this function is called while the trace is in progress, a special system trace event,  
30527 POSIX\_TRACE\_FILTER, shall be recorded in the trace indicating both the old and the new sets  
30528 of filtered trace event types (see Table 2-4 (on page 528) and Table 2-6 (on page 529)).

30529 If the *posix\_trace\_set\_filter()* function is interrupted by a signal, an error shall be returned and the  
30530 filter shall not be changed. In this case, the state of the trace stream shall not be changed.

30531 The value of the argument *how* indicates the manner in which the set is to be changed and shall  
30532 have one of the following values, as defined in the **<trace.h>** header:

30533 POSIX\_TRACE\_SET\_EVENTSET  
30534 The resulting set of trace event types to be filtered shall be the trace event type set pointed  
30535 to by the argument *set*.

30536 POSIX\_TRACE\_ADD\_EVENTSET  
30537 The resulting set of trace event types to be filtered shall be the union of the current set and  
30538 the trace event type set pointed to by the argument *set*.

30539 POSIX\_TRACE\_SUB\_EVENTSET  
30540 The resulting set of trace event types to be filtered shall be all trace event types in the  
30541 current set that are not in the set pointed to by the argument *set*; that is, remove each  
30542 element of the specified set from the current filter.

30543 **RETURN VALUE**

30544 Upon successful completion, these functions shall return a value of zero. Otherwise, they shall  
30545 return the corresponding error number.

30546 The *posix\_trace\_get\_filter()* function stores the set of filtered trace event types in *set*, if successful.

30547 **ERRORS**

30548 These functions shall fail if:

30549 [EINVAL] The value of the *trid* argument does not correspond to an active trace stream  
30550 or the value of the argument pointed to by *set* is invalid.

30551 [EINTR] The operation was interrupted by a signal.

30552 **EXAMPLES**

30553 None.

30554 **APPLICATION USAGE**

30555 None.

30556 **RATIONALE**

30557 None.

30558 **FUTURE DIRECTIONS**

30559 None.

30560 **SEE ALSO**30561 *posix\_trace\_eventset\_add()*, the Base Definitions volume of IEEE Std 1003.1-200x, <trace.h>30562 **CHANGE HISTORY**

30563 First released in Issue 6. Derived from IEEE Std 1003.1q-2000. |

30564 IEEE PASC Interpretation 1003.1 #123 is applied. |

30565 **NAME**

30566        posix\_trace\_get\_status — retrieve the trace statuses (**TRACING**)

30567 **SYNOPSIS**

30568 TRC     #include <trace.h>

```
30569        int posix_trace_get_status(trace_id_t trid,  
30570                                    struct posix_trace_status_info *statusinfo);
```

30571

30572 **DESCRIPTION**

30573        Refer to *posix\_trace\_get\_attr()*.

## 30574 NAME

30575 posix\_trace\_getnext\_event, posix\_trace\_timedgetnext\_event, posix\_trace\_trygetnext\_event —  
 30576 retrieve a trace event (**TRACING**)

## 30577 SYNOPSIS

```
30578 TRC #include <sys/types.h>
30579 #include <trace.h>

30580 int posix_trace_getnext_event(trace_id_t trid,
30581 struct posix_trace_event_info *restrict event,
30582 void *restrict data, size_t num_bytes,
30583 size_t *restrict data_len, int *restrict unavailable);
30584 TRC TMO int posix_trace_timedgetnext_event(trace_id_t trid,
30585 struct posix_trace_event_info *restrict event,
30586 void *restrict data, size_t num_bytes,
30587 size_t *restrict data_len, int *restrict unavailable,
30588 const struct timespec *restrict abs_timeout);
30589 TRC int posix_trace_trygetnext_event(trace_id_t trid,
30590 struct posix_trace_event_info *restrict event,
30591 void *restrict data, size_t num_bytes,
30592 size_t *restrict data_len, int *restrict unavailable);
30593
```

## 30594 DESCRIPTION

30595 The *posix\_trace\_getnext\_event()* function shall report a recorded trace event either from an active |  
 30596 TRL trace stream without log or a pre-recorded trace stream identified by the *trid* argument. The |  
 30597 *posix\_trace\_trygetnext\_event()* function shall report a recorded trace event from an active trace |  
 30598 stream without log identified by the *trid* argument.

30599 The trace event information associated with the recorded trace event shall be copied by the  
 30600 function into the structure pointed to by the argument *event* and the data associated with the  
 30601 trace event shall be copied into the buffer pointed to by the *data* argument.

30602 The *posix\_trace\_getnext\_event()* function shall block if the *trid* argument identifies an active trace  
 30603 stream and there is currently no trace event ready to be retrieved. When returning, if a recorded  
 30604 trace event was reported, the variable pointed to by the *unavailable* argument shall be set to zero.  
 30605 Otherwise, the variable pointed to by the *unavailable* argument shall be set to a value different  
 30606 from zero.

30607 TMO The *posix\_trace\_timedgetnext\_event()* function shall attempt to get another trace event from an |  
 30608 active trace stream without log, as in the *posix\_trace\_getnext\_event()* function. However, if no |  
 30609 trace event is available from the trace stream, the implied wait shall be terminated when the |  
 30610 timeout specified by the argument *abs\_timeout* expires, and the function shall return the error |  
 30611 [ETIMEDOUT].

30612 The timeout shall expire when the absolute time specified by *abs\_timeout* passes, as measured by |  
 30613 the clock upon which timeouts are based (that is, when the value of that clock equals or exceeds  
 30614 *abs\_timeout*), or if the absolute time specified by *abs\_timeout* has already passed at the time of the  
 30615 call.

30616 TMO TMR If the Timers option is supported, the timeout shall be based on the CLOCK\_REALTIME clock; |  
 30617 if the Timers option is not supported, the timeout shall be based on the system clock as returned |  
 30618 by the *time()* function. The resolution of the timeout shall be the resolution of the clock on which |  
 30619 it is based. The **timespec** data type is defined in the **<time.h>** header. |

30620 TMO Under no circumstance shall the function fail with a timeout if a trace event is immediately |  
 30621 available from the trace stream. The validity of the *abs\_timeout* argument need not be checked if

30622 a trace event is immediately available from the trace stream.

30623 The behavior of this function for a pre-recorded trace stream is unspecified.

30624 TRL The *posix\_trace\_trygetnext\_event()* function shall not block. This function shall return an error if  
 30625 the *trid* argument identifies a pre-recorded trace stream. If a recorded trace event was reported,  
 30626 the variable pointed to by the *unavailable* argument shall be set to zero. Otherwise, if no trace  
 30627 event was reported, the variable pointed to by the *unavailable* argument shall be set to a value  
 30628 different from zero.

30629 The argument *num\_bytes* shall be the size of the buffer pointed to by the *data* argument. The  
 30630 argument *data\_len* reports to the application the length in bytes of the data record just  
 30631 transferred. If *num\_bytes* is greater than or equal to the size of the data associated with the trace  
 30632 event pointed to by the *event* argument, all the recorded data shall be transferred. In this case, the  
 30633 *truncation-status* member of the trace event structure shall be either  
 30634 POSIX\_TRACE\_NOT\_TRUNCATED, if the trace event data was recorded without truncation  
 30635 while tracing, or POSIX\_TRACE\_TRUNCATED\_RECORD, if the trace event data was truncated  
 30636 when it was recorded. If the *num\_bytes* argument is less than the length of recorded trace event  
 30637 data, the data transferred shall be truncated to a length of *num\_bytes*, the value stored in the  
 30638 variable pointed to by *data\_len* shall be equal to *num\_bytes*, and the *truncation-status* member of  
 30639 the *event* structure argument shall be set to POSIX\_TRACE\_TRUNCATED\_READ (see the  
 30640 *posix\_trace\_event\_info()* function).

30641 The report of a trace event shall be sequential starting from the oldest recorded trace event. Trace  
 30642 events shall be reported in the order in which they were generated, up to an implementation-  
 30643 defined time resolution that causes the ordering of trace events occurring very close to each  
 30644 other to be unknown. Once reported, a trace event cannot be reported again from an active trace  
 30645 stream. Once a trace event is reported from an active trace stream without log, the trace stream  
 30646 shall make the resources associated with that trace event available to record future generated  
 30647 trace events.

30648 **RETURN VALUE**

30649 Upon successful completion, these functions shall return a value of zero. Otherwise, they shall  
 30650 return the corresponding error number.

30651 If successful, these functions store:

- 30652 • The recorded trace event in the object pointed to by *event*
- 30653 • The trace event information associated with the recorded trace event in the object pointed to  
 30654 by *data*
- 30655 • The length of this trace event information in the object pointed to by *data\_len*
- 30656 • The value of zero in the object pointed to by *unavailable*

30657 **ERRORS**

30658 These functions shall fail if:

30659 [EINVAL] The trace stream identifier argument *trid* is invalid.

30660 The *posix\_trace\_getnext\_event()* and *posix\_trace\_timedgetnext\_event()* functions shall fail if:

30661 [EINTR] The operation was interrupted by a signal, and so the call had no effect.

30662 The *posix\_trace\_trygetnext\_event()* function shall fail if:

30663 [EINVAL] The trace stream identifier argument *trid* does not correspond to an active  
 30664 trace stream.

30665 TMO The *posix\_trace\_timedgetnext\_event()* function shall fail if: |

30666 [EINVAL] There is no trace event immediately available from the trace stream, and the  
30667 *timeout* argument is invalid.

30668 [ETIMEDOUT] No trace event was available from the trace stream before the specified  
30669 *timeout* *timeout* expired.  
30670

30671 **EXAMPLES**

30672 None.

30673 **APPLICATION USAGE**

30674 None.

30675 **RATIONALE**

30676 None.

30677 **FUTURE DIRECTIONS**

30678 None.

30679 **SEE ALSO**

30680 *posix\_trace\_create()*, **posix\_trace\_event\_info Structure**, *posix\_trace\_open()*, the Base Definitions  
30681 volume of IEEE Std 1003.1-200x, <sys/types.h>, <trace.h>

30682 **CHANGE HISTORY**

30683 First released in Issue 6. Derived from IEEE Std 1003.1q-2000. |

30684 IEEE PASC Interpretation 1003.1 #123 is applied. |

30685 **NAME**

30686        posix\_trace\_open — trace log management (**TRACING**)

30687 **SYNOPSIS**

30688 TCT TRL   #include <trace.h>

30689        int posix\_trace\_open(int *file\_desc*, trace\_id\_t \**trid*);

30690

30691 **DESCRIPTION**

30692        Refer to *posix\_trace\_close()*.

30693 **NAME**30694        posix\_trace\_rewind — trace log management (**TRACING**)30695 **SYNOPSIS**

30696 TCT TRL #include &lt;trace.h&gt;

30697        int posix\_trace\_rewind(trace\_id\_t *trid*);

30698

30699 **DESCRIPTION**30700        Refer to *posix\_trace\_close()*.

30701 **NAME**

30702        posix\_trace\_set\_filter — set filter of an initialized trace stream (**TRACING**)

30703 **SYNOPSIS**

30704 TRC TEF #include <trace.h>

```
30705     int posix_trace_set_filter(trace_id_t trid,  
30706                             const trace_event_set_t *set, int how);
```

30707

30708 **DESCRIPTION**

30709        Refer to *posix\_trace\_get\_filter()*.

30710 **NAME**30711        posix\_trace\_shutdown — trace stream shutdown from a process (**TRACING**)30712 **SYNOPSIS**

30713 TRC     #include &lt;sys/types.h&gt;

30714        #include &lt;trace.h&gt;

30715        int posix\_trace\_shutdown(trace\_id\_t *trid*);

30716

30717 **DESCRIPTION**30718        Refer to *posix\_trace\_create()*.

30719 **NAME**

30720 `posix_trace_start`, `posix_trace_stop` — trace start and stop (**TRACING**)

30721 **SYNOPSIS**

30722 TRC `#include <trace.h>`

30723 `int posix_trace_start(trace_id_t trid);`

30724 `int posix_trace_stop (trace_id_t trid);`

30725

30726 **DESCRIPTION**

30727 The `posix_trace_start()` and `posix_trace_stop()` functions, respectively, shall start and stop the  
 30728 trace stream identified by the argument *trid*.

30729 The effect of calling the `posix_trace_start()` function shall be recorded in the trace stream as the  
 30730 POSIX\_TRACE\_START system trace event and the status of the trace stream shall become  
 30731 POSIX\_TRACE\_RUNNING. If the trace stream is in progress when this function is called, the  
 30732 POSIX\_TRACE\_START system trace event shall not be recorded and the trace stream shall  
 30733 continue to run. If the trace stream is full, the POSIX\_TRACE\_START system trace event shall  
 30734 not be recorded and the status of the trace stream shall not be changed.

30735 The effect of calling the `posix_trace_stop()` function shall be recorded in the trace stream as the  
 30736 POSIX\_TRACE\_STOP system trace event and the status of the trace stream shall become  
 30737 POSIX\_TRACE\_SUSPENDED. If the trace stream is suspended when this function is called, the  
 30738 POSIX\_TRACE\_STOP system trace event shall not be recorded and the trace stream shall remain  
 30739 suspended. If the trace stream is full, the POSIX\_TRACE\_STOP system trace event shall not be  
 30740 recorded and the status of the trace stream shall not be changed.

30741 **RETURN VALUE**

30742 Upon successful completion, these functions shall return a value of zero. Otherwise, they shall  
 30743 return the corresponding error number.

30744 **ERRORS**

30745 These functions shall fail if:

30746 [EINVAL] The value of the argument *trid* does not correspond to an active trace stream  
 30747 and thus no trace stream was started or stopped.

30748 [EINTR] The operation was interrupted by a signal and thus the trace stream was not  
 30749 necessarily started or stopped.

30750 **EXAMPLES**

30751 None.

30752 **APPLICATION USAGE**

30753 None.

30754 **RATIONALE**

30755 None.

30756 **FUTURE DIRECTIONS**

30757 None.

30758 **SEE ALSO**

30759 `posix_trace_create()`, the Base Definitions volume of IEEE Std 1003.1-200x, `<trace.h>`

30760 **CHANGE HISTORY**

30761 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

30762 IEEE PASC Interpretation 1003.1 #123 is applied.

30763 **NAME**

30764        posix\_trace\_timedgetnext\_event, — retrieve a trace event (**TRACING**)

30765 **SYNOPSIS**

```
30766 TRC TMO #include <sys/types.h>
30767        #include <trace.h>
```

```
30768        int posix_trace_timedgetnext_event(trace_id_t trid,
30769        struct posix_trace_event_info *restrict event,
30770        void *restrict data, size_t num_bytes,
30771        size_t *restrict data_len, int *restrict unavailable,
30772        const struct timespec *restrict abs_timeout);
30773
```

30774 **DESCRIPTION**

30775        Refer to *posix\_trace\_getnext\_event()*.

30776 **NAME**30777 `posix_trace_trid_eventid_open` — manipulate trace event type identifier (**TRACING**)30778 **SYNOPSIS**30779 TRC TEF `#include <trace.h>`

```
30780 int posix_trace_trid_eventid_open(trace_id_t trid,  
30781     const char *restrict event_name,  
30782     trace_event_id_t *restrict event);
```

30783

30784 **DESCRIPTION**30785 Refer to `posix_trace_eventid_equal()`.

30786 **NAME**

30787        posix\_trace\_trygetnext\_event — retrieve a trace event (**TRACING**)

30788 **SYNOPSIS**

30789 TRC     #include <sys/types.h>

30790        #include <trace.h>

```
30791        int posix_trace_trygetnext_event(trace_id_t trid,  
30792                                        struct posix_trace_event_info *restrict event,  
30793                                        void *restrict data, size_t num_bytes,  
30794                                        size_t *restrict data_len, int *restrict unavailable);
```

30795

30796 **DESCRIPTION**

30797        Refer to *posix\_trace\_getnext\_event()*.

30798 **NAME**30799 `posix_typed_mem_get_info` — query typed memory information (**ADVANCED REALTIME**)30800 **SYNOPSIS**30801 `TYM` `#include <sys/mman.h>`30802 `int posix_typed_mem_get_info(int fildes,`  
30803 `struct posix_typed_mem_info *info);`

30804

30805 **DESCRIPTION**

30806 The `posix_typed_mem_get_info()` function shall return, in the `posix_tmi_length` field of the  
 30807 **posix\_typed\_mem\_info** structure pointed to by `info`, the maximum length which may be  
 30808 successfully allocated by the typed memory object designated by `fildes`. This maximum length  
 30809 shall take into account the flag `POSIX_TYPED_MEM_ALLOCATE` or  
 30810 `POSIX_TYPED_MEM_ALLOCATE_CONTIG` specified when the typed memory object  
 30811 represented by `fildes` was opened. The maximum length is dynamic; therefore, the value returned  
 30812 is valid only while the current mapping of the corresponding typed memory pool remains  
 30813 unchanged.

30814 If `fildes` represents a typed memory object opened with neither the  
 30815 `POSIX_TYPED_MEM_ALLOCATE` flag nor the `POSIX_TYPED_MEM_ALLOCATE_CONTIG`  
 30816 flag specified, the returned value of `info->posix_tmi_length` is unspecified.

30817 The `posix_typed_mem_get_info()` function may return additional implementation-defined  
 30818 information in other fields of the **posix\_typed\_mem\_info** structure pointed to by `info`.

30819 If the memory object specified by `fildes` is not a typed memory object, then the behavior of this  
 30820 function is undefined.

30821 **RETURN VALUE**

30822 Upon successful completion, the `posix_typed_mem_get_info()` function shall return zero;  
 30823 otherwise, the corresponding error status value shall be returned.

30824 **ERRORS**

30825 The `posix_typed_mem_get_info()` function shall fail if:

30826 [EBADF] The `fildes` argument is not a valid open file descriptor.

30827 [ENODEV] The `fildes` argument is not connected to a memory object supported by this  
 30828 function.

30829 This function shall not return an error code of [EINTR].

30830 **EXAMPLES**

30831 None.

30832 **APPLICATION USAGE**

30833 None.

30834 **RATIONALE**

30835 An application that needs to allocate a block of typed memory with length dependent upon the  
 30836 amount of memory currently available must either query the typed memory object to obtain the  
 30837 amount available, or repeatedly invoke `mmap()` attempting to guess an appropriate length.  
 30838 While the latter method is existing practice with `malloc()`, it is awkward and imprecise. The  
 30839 `posix_typed_mem_get_info()` function allows an application to immediately determine available  
 30840 memory. This is particularly important for typed memory objects that may in some cases be  
 30841 scarce resources. Note that when a typed memory pool is a shared resource, some form of  
 30842 mutual exclusion or synchronization may be required while typed memory is being queried and

30843 allocated to prevent race conditions.

30844 The existing *fstat()* function is not suitable for this purpose. We realize that implementations  
30845 may wish to provide other attributes of typed memory objects (for example, alignment  
30846 requirements, page size, and so on). The *fstat()* function returns a structure which is not  
30847 extensible and, furthermore, contains substantial information that is inappropriate for typed  
30848 memory objects.

30849 **FUTURE DIRECTIONS**

30850 None.

30851 **SEE ALSO**

30852 *fstat()*, *mmap()*, *posix\_typed\_mem\_open()*, the Base Definitions volume of IEEE Std 1003.1-200x,  
30853 <sys/mman.h>

30854 **CHANGE HISTORY**

30855 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

30856 **NAME**30857 posix\_typed\_mem\_open — open a typed memory object (**ADVANCED REALTIME**)30858 **SYNOPSIS**

30859 TYM #include &lt;sys/mman.h&gt;

30860 int posix\_typed\_mem\_open(const char \*name, int oflag, int tflag);

30861

30862 **DESCRIPTION**

30863 The *posix\_typed\_mem\_open()* function shall establish a connection between the typed memory  
 30864 object specified by the string pointed to by *name* and a file descriptor. It shall create an open file  
 30865 description that refers to the typed memory object and a file descriptor that refers to that open  
 30866 file description. The file descriptor is used by other functions to refer to that typed memory  
 30867 object. It is unspecified whether the name appears in the file system and is visible to other  
 30868 functions that take pathnames as arguments. The *name* argument shall conform to the  
 30869 construction rules for a pathname. If *name* begins with the slash character, then processes calling  
 30870 *posix\_typed\_mem\_open()* with the same value of *name* shall refer to the same typed memory  
 30871 object. If *name* does not begin with the slash character, the effect is implementation-defined. The  
 30872 interpretation of slash characters other than the leading slash character in *name* is  
 30873 implementation-defined.

30874 Each typed memory object supported in a system shall be identified by a name which specifies  
 30875 not only its associated typed memory pool, but also the path or port by which it is accessed. That  
 30876 is, the same typed memory pool accessed via several different ports shall have several different  
 30877 corresponding names. The binding between names and typed memory objects is established in  
 30878 an implementation-defined manner. Unlike shared memory objects, there is no way within  
 30879 IEEE Std 1003.1-200x for a program to create a typed memory object.

30880 The value of *tflag* shall determine how the typed memory object behaves when subsequently  
 30881 mapped by calls to *mmap()*. At most, one of the following flags defined in <sys/mman.h> may  
 30882 be specified:

30883 POSIX\_TYPED\_MEM\_ALLOCATE

30884 Allocate on *mmap()*.

30885 POSIX\_TYPED\_MEM\_ALLOCATE\_CONTIG

30886 Allocate contiguously on *mmap()*.

30887 POSIX\_TYPED\_MEM\_MAP\_ALLOCATABLE

30888 Map on *mmap()*, without affecting allocatability.

30889 If *tflag* has the flag POSIX\_TYPED\_MEM\_ALLOCATE specified, any subsequent call to *mmap()*  
 30890 using the returned file descriptor shall result in allocation and mapping of typed memory from  
 30891 the specified typed memory pool. The allocated memory may be a contiguous previously  
 30892 unallocated area of the typed memory pool or several non-contiguous previously unallocated  
 30893 areas (mapped to a contiguous portion of the process address space). If *tflag* has the flag  
 30894 POSIX\_TYPED\_MEM\_ALLOCATE\_CONTIG specified, any subsequent call to *mmap()* using the  
 30895 returned file descriptor shall result in allocation and mapping of a single contiguous previously  
 30896 unallocated area of the typed memory pool (also mapped to a contiguous portion of the process  
 30897 address space). If *tflag* has none of the flags POSIX\_TYPED\_MEM\_ALLOCATE or  
 30898 POSIX\_TYPED\_MEM\_ALLOCATE\_CONTIG specified, any subsequent call to *mmap()* using the  
 30899 returned file descriptor shall map an application-chosen area from the specified typed memory  
 30900 pool such that this mapped area becomes unavailable for allocation until unmapped by all  
 30901 processes. If *tflag* has the flag POSIX\_TYPED\_MEM\_MAP\_ALLOCATABLE specified, any  
 30902 subsequent call to *mmap()* using the returned file descriptor shall map an application-chosen  
 30903 area from the specified typed memory pool without an effect on the availability of that area for

30904 allocation; that is, mapping such an object leaves each byte of the mapped area unallocated if it  
 30905 was unallocated prior to the mapping or allocated if it was allocated prior to the mapping. The  
 30906 appropriate privilege to specify the POSIX\_TYPED\_MEM\_MAP\_ALLOCATABLE flag is  
 30907 implementation-defined.

30908 If successful, *posix\_typed\_mem\_open()* shall return a file descriptor for the typed memory object  
 30909 that is the lowest numbered file descriptor not currently open for that process. The open file  
 30910 description is new, and therefore the file descriptor shall not share it with any other processes. It  
 30911 is unspecified whether the file offset is set. The FD\_CLOEXEC file descriptor flag associated  
 30912 with the new file descriptor shall be cleared.

30913 The behavior of *msync()*, *ftruncate()*, and all file operations other than *mmap()*,  
 30914 *posix\_mem\_offset()*, *posix\_typed\_mem\_get\_info()*, *fstat()*, *dup()*, *dup2()*, and *close()*, is unspecified  
 30915 when passed a file descriptor connected to a typed memory object by this function.

30916 The file status flags of the open file description shall be set according to the value of *oflag*.  
 30917 Applications shall specify exactly one of the three access mode values described below and  
 30918 defined in the <fcntl.h> header, as the value of *oflag*.

- 30919 O\_RDONLY Open for read access only.
- 30920 O\_WRONLY Open for write access only.
- 30921 O\_RDWR Open for read or write access.

30922 **RETURN VALUE**

30923 Upon successful completion, the *posix\_typed\_mem\_open()* function shall return a non-negative  
 30924 integer representing the lowest numbered unused file descriptor. Otherwise, it shall return -1  
 30925 and set *errno* to indicate the error.

30926 **ERRORS**

30927 The *posix\_typed\_mem\_open()* function shall fail if:

- 30928 [EACCES] The typed memory object exists and the permissions specified by *oflag* are  
 30929 denied.
- 30930 [EINTR] The *posix\_typed\_mem\_open()* operation was interrupted by a signal.
- 30931 [EINVAL] The flags specified in *tflag* are invalid (more than one of  
 30932 POSIX\_TYPED\_MEM\_ALLOCATE,  
 30933 POSIX\_TYPED\_MEM\_ALLOCATE\_CONTIG, or  
 30934 POSIX\_TYPED\_MEM\_MAP\_ALLOCATABLE is specified).
- 30935 [EMFILE] Too many file descriptors are currently in use by this process.
- 30936 [ENAMETOOLONG]  
 30937 The length of the *name* argument exceeds {PATH\_MAX} or a pathname  
 30938 component is longer than {NAME\_MAX}.
- 30939 [ENFILE] Too many file descriptors are currently open in the system.
- 30940 [ENOENT] The named typed memory object does not exist.
- 30941 [EPERM] The caller lacks the appropriate privilege to specify the flag  
 30942 POSIX\_TYPED\_MEM\_MAP\_ALLOCATABLE in argument *tflag*.

30943 **EXAMPLES**

30944 None.

30945 **APPLICATION USAGE**

30946 None.

30947 **RATIONALE**

30948 None.

30949 **FUTURE DIRECTIONS**

30950 None.

30951 **SEE ALSO**

30952 *close()*, *dup()*, *exec*, *fcntl()*, *fstat()*, *ftruncate()*, *mmap()*, *msync()*, *posix\_mem\_offset()*,  
30953 *posix\_typed\_mem\_get\_info()*, *umask()*, the Base Definitions volume of IEEE Std 1003.1-200x,  
30954 *<fcntl.h>*, *<sys/mman.h>*

30955 **CHANGE HISTORY**

30956 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

30957 **NAME**

30958 pow, powf, powl — power function

30959 **SYNOPSIS**

30960 #include &lt;math.h&gt;

30961 double pow(double x, double y);

30962 float powf(float x, float y);

30963 long double powl(long double x, long double y);

30964 **DESCRIPTION**

30965 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 30966 conflict between the requirements described here and the ISO C standard is unintentional. This  
 30967 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

30968 These functions shall compute the value of  $x$  raised to the power  $y$ ,  $x^y$ . If  $x$  is negative, the  
 30969 application shall ensure that  $y$  is an integer value.

30970 An application wishing to check for error situations should set *errno* to zero and call  
 30971 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 30972 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 30973 zero, an error has occurred.

30974 **RETURN VALUE**30975 Upon successful completion, these functions shall return the value of  $x$  raised to the power  $y$ .

30976 **MX** For finite values of  $x < 0$ , and finite non-integer values of  $y$ , a domain error shall occur and either  
 30977 a NaN (if representable), or an implementation-defined value shall be returned.

30978 If the correct value would cause overflow, a range error shall occur and *pow()*, *powf()*, and  
 30979 *powl()* shall return HUGE\_VAL, HUGE\_VALF, and HUGE\_VALL, respectively.

30980 If the correct value would cause underflow, and is not representable, a range error may occur,  
 30981 **MX** and either 0.0 (if supported), or an implementation-defined value shall be returned.

30982 **MX** If  $x$  or  $y$  is a NaN, a NaN shall be returned (unless specified elsewhere in this description).

30983 For any value of  $y$  (including NaN), if  $x$  is +1, 1.0 shall be returned.

30984 For any value of  $x$  (including NaN), if  $y$  is  $\pm 0$ , 1.0 shall be returned.

30985 For any odd integer value of  $y > 0$ , if  $x$  is  $\pm 0$ ,  $\pm 0$  shall be returned.

30986 For  $y > 0$  and not an odd integer, if  $x$  is  $\pm 0$ , +0 shall be returned.

30987 If  $x$  is  $-1$ , and  $y$  is  $\pm \text{Inf}$ , 1.0 shall be returned.

30988 For  $|x| < 1$ , if  $y$  is  $-\text{Inf}$ ,  $+\text{Inf}$  shall be returned.

30989 For  $|x| > 1$ , if  $y$  is  $-\text{Inf}$ , +0 shall be returned.

30990 For  $|x| < 1$ , if  $y$  is  $+\text{Inf}$ , +0 shall be returned.

30991 For  $|x| > 1$ , if  $y$  is  $+\text{Inf}$ ,  $+\text{Inf}$  shall be returned.

30992 For  $y$  an odd integer  $< 0$ , if  $x$  is  $-\text{Inf}$ ,  $-0$  shall be returned.

30993 For  $y < 0$  and not an odd integer, if  $x$  is  $-\text{Inf}$ , +0 shall be returned.

30994 For  $y$  an odd integer  $> 0$ , if  $x$  is  $-\text{Inf}$ ,  $-\text{Inf}$  shall be returned.

30995 For  $y > 0$  and not an odd integer, if  $x$  is  $-\text{Inf}$ ,  $+\text{Inf}$  shall be returned.

30996 For  $y < 0$ , if  $x$  is  $+\text{Inf}$ ,  $+0$  shall be returned.

30997 For  $y > 0$ , if  $x$  is  $+\text{Inf}$ ,  $+\text{Inf}$  shall be returned.

30998 For  $y$  an odd integer  $< 0$ , if  $x$  is  $\pm 0$ , a pole error shall occur and  $\pm\text{HUGE\_VAL}$ ,  $\pm\text{HUGE\_VALF}$ , and  
 30999  $\pm\text{HUGE\_VALL}$  shall be returned for  $\text{pow}()$ ,  $\text{powf}()$ , and  $\text{powl}()$ , respectively.

31000 For  $y < 0$  and not an odd integer, if  $x$  is  $\pm 0$ , a pole error shall occur and  $\text{HUGE\_VAL}$ ,  
 31001  $\text{HUGE\_VALF}$ , and  $\text{HUGE\_VALL}$  shall be returned for  $\text{pow}()$ ,  $\text{powf}()$ , and  $\text{powl}()$ , respectively.

31002 If the correct value would cause underflow, and is representable, a range error may occur and  
 31003 the correct value shall be returned.

**31004 ERRORS**

31005 These functions shall fail if:

31006 Domain Error The value of  $x$  is negative and  $y$  is a finite non-integer.

31007 If the integer expression ( $\text{math\_errhandling}$  &  $\text{MATH\_ERRNO}$ ) is non-zero, |  
 31008 then *errno* shall be set to [EDOM]. If the integer expression ( $\text{math\_errhandling}$  |  
 31009 &  $\text{MATH\_ERREXCEPT}$ ) is non-zero, then the invalid floating-point exception |  
 31010 shall be raised. |

31011 MX Pole Error The value of  $x$  is zero and  $y$  is negative.

31012 If the integer expression ( $\text{math\_errhandling}$  &  $\text{MATH\_ERRNO}$ ) is non-zero, |  
 31013 then *errno* shall be set to [ERANGE]. If the integer expression |  
 31014 ( $\text{math\_errhandling}$  &  $\text{MATH\_ERREXCEPT}$ ) is non-zero, then the divide-by- |  
 31015 zero floating-point exception shall be raised. |

31016 Range Error The result overflows.

31017 If the integer expression ( $\text{math\_errhandling}$  &  $\text{MATH\_ERRNO}$ ) is non-zero, |  
 31018 then *errno* shall be set to [ERANGE]. If the integer expression |  
 31019 ( $\text{math\_errhandling}$  &  $\text{MATH\_ERREXCEPT}$ ) is non-zero, then the overflow |  
 31020 floating-point exception shall be raised. |

31021 These functions may fail if:

31022 Range Error The result underflows.

31023 If the integer expression ( $\text{math\_errhandling}$  &  $\text{MATH\_ERRNO}$ ) is non-zero, |  
 31024 then *errno* shall be set to [ERANGE]. If the integer expression |  
 31025 ( $\text{math\_errhandling}$  &  $\text{MATH\_ERREXCEPT}$ ) is non-zero, then the underflow |  
 31026 floating-point exception shall be raised. |

**31027 EXAMPLES**

31028 None.

**31029 APPLICATION USAGE**

31030 On error, the expressions ( $\text{math\_errhandling}$  &  $\text{MATH\_ERRNO}$ ) and ( $\text{math\_errhandling}$  &  
 31031  $\text{MATH\_ERREXCEPT}$ ) are independent of each other, but at least one of them must be non-zero.

**31032 RATIONALE**

31033 None.

**31034 FUTURE DIRECTIONS**

31035 None.

31036 **SEE ALSO**

31037 *exp()*, *feclearexcept()*, *fetestexcept()*, *isnan()*, the Base Definitions volume of IEEE Std 1003.1-200x, |  
31038 Section 4.18, Treatment of Error Conditions for Mathematical Functions, <math.h> |

31039 **CHANGE HISTORY**

31040 First released in Issue 1. Derived from Issue 1 of the SVID.

31041 **Issue 5**

31042 The DESCRIPTION is updated to indicate how an application should check for an error. This  
31043 text was previously published in the APPLICATION USAGE section.

31044 **Issue 6**

31045 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

31046 The *powf()* and *powl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

31047 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
31048 revised to align with the ISO/IEC 9899:1999 standard.

31049 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
31050 marked.

31051 **NAME**

31052        pread — read from a file

31053 **SYNOPSIS**

31054 xSI        #include &lt;unistd.h&gt;

31055        ssize\_t pread(int *fd*, void \**buf*, size\_t *nbyte*, off\_t *offset*);

31056

31057 **DESCRIPTION**31058        Refer to *read()*.

31059 **NAME**

31060           printf — print formatted output

31061 **SYNOPSIS**

31062           #include <stdio.h>

31063           int printf(const char \*restrict *format*, ...);

31064 **DESCRIPTION**

31065           Refer to *fprintf()*.

31066 **NAME**

31067 pselect, select — synchronous I/O multiplexing

31068 **SYNOPSIS**

```

31069 #include <sys/select.h>

31070 int pselect(int nfds, fd_set *restrict readfds,
31071            fd_set *restrict writefds, fd_set *restrict errorfds,
31072            const struct timespec *restrict timeout,
31073            const sigset_t *restrict sigmask);
31074 int select(int nfds, fd_set *restrict readfds,
31075           fd_set *restrict writefds, fd_set *restrict errorfds,
31076           struct timeval *restrict timeout);
31077 void FD_CLR(int fd, fd_set *fdset);
31078 int FD_ISSET(int fd, fd_set *fdset);
31079 void FD_SET(int fd, fd_set *fdset);
31080 void FD_ZERO(fd_set *fdset);

```

31081 **DESCRIPTION**

31082 The *pselect()* function shall examine the file descriptor sets whose addresses are passed in the  
 31083 *readfds*, *writefds*, and *errorfds* parameters to see if some of their descriptors are ready for reading,  
 31084 are ready for writing, or have an exceptional condition pending, respectively.

31085 The *select()* function shall be equivalent to the *pselect()* function, except as follows:

- 31086 • For the *select()* function, the timeout period is given in seconds and microseconds in an  
 31087 argument of type **struct timeval**, whereas for the *pselect()* function the timeout period is  
 31088 given in seconds and nanoseconds in an argument of type **struct timespec**.
- 31089 • The *select()* function has no *sigmask* argument; it shall behave as *pselect()* does when *sigmask*  
 31090 is a null pointer.
- 31091 • Upon successful completion, the *select()* function may modify the object pointed to by the  
 31092 *timeout* argument.

31093 The *pselect()* and *select()* functions shall support regular files, terminal and pseudo-terminal  
 31094 XSR devices, **STREAMS**-based files, FIFOs, pipes, and sockets. The behavior of *pselect()* and *select()*  
 31095 on file descriptors that refer to other types of file is unspecified.

31096 The *nfds* argument specifies the range of descriptors to be tested. The first *nfds* descriptors shall  
 31097 be checked in each set; that is, the descriptors from zero through *nfds*–1 in the descriptor sets  
 31098 shall be examined.

31099 If the *readfds* argument is not a null pointer, it points to an object of type **fd\_set** that on input  
 31100 specifies the file descriptors to be checked for being ready to read, and on output indicates  
 31101 which file descriptors are ready to read.

31102 If the *writefds* argument is not a null pointer, it points to an object of type **fd\_set** that on input  
 31103 specifies the file descriptors to be checked for being ready to write, and on output indicates  
 31104 which file descriptors are ready to write.

31105 If the *errorfds* argument is not a null pointer, it points to an object of type **fd\_set** that on input  
 31106 specifies the file descriptors to be checked for error conditions pending, and on output indicates  
 31107 which file descriptors have error conditions pending.

31108 Upon successful completion, the *pselect()* or *select()* function shall modify the objects pointed to  
 31109 by the *readfds*, *writefds*, and *errorfds* arguments to indicate which file descriptors are ready for  
 31110 reading, ready for writing, or have an error condition pending, respectively, and shall return the  
 31111 total number of ready descriptors in all the output sets. For each file descriptor less than *nfds*, the

31112 corresponding bit shall be set on successful completion if it was set on input and the associated  
31113 condition is true for that file descriptor.

31114 If none of the selected descriptors are ready for the requested operation, the *pselect()* or *select()* |  
31115 function shall block until at least one of the requested operations becomes ready, until the |  
31116 *timeout* occurs, or until interrupted by a signal. The *timeout* parameter controls how long the |  
31117 *pselect()* or *select()* function shall take before timing out. If the *timeout* parameter is not a null |  
31118 pointer, it specifies a maximum interval to wait for the selection to complete. If the specified  
31119 time interval expires without any requested operation becoming ready, the function shall return.  
31120 If the *timeout* parameter is a null pointer, then the call to *pselect()* or *select()* shall block  
31121 indefinitely until at least one descriptor meets the specified criteria. To effect a poll, the *timeout*  
31122 parameter should not be a null pointer, and should point to a zero-valued **timespec** structure.

31123 The use of a timeout does not affect any pending timers set up by *alarm()*, *ualarm()*, or  
31124 *setitimer()*.

31125 Implementations may place limitations on the maximum timeout interval supported. All  
31126 implementations shall support a maximum timeout interval of at least 31 days. If the *timeout*  
31127 argument specifies a timeout interval greater than the implementation-defined maximum value,  
31128 the maximum value shall be used as the actual timeout value. Implementations may also place  
31129 limitations on the granularity of timeout intervals. If the requested timeout interval requires a  
31130 finer granularity than the implementation supports, the actual timeout interval shall be rounded  
31131 up to the next supported value.

31132 If *sigmask* is not a null pointer, then the *pselect()* function shall replace the signal mask of the  
31133 process by the set of signals pointed to by *sigmask* before examining the descriptors, and shall  
31134 restore the signal mask of the process before returning.

31135 A descriptor shall be considered ready for reading when a call to an input function with  
31136 O\_NONBLOCK clear would not block, whether or not the function would transfer data  
31137 successfully. (The function might return data, an end-of-file indication, or an error other than  
31138 one indicating that it is blocked, and in each of these cases the descriptor shall be considered  
31139 ready for reading.)

31140 A descriptor shall be considered ready for writing when a call to an output function with  
31141 O\_NONBLOCK clear would not block, whether or not the function would transfer data  
31142 successfully.

31143 If a socket has a pending error, it shall be considered to have an exceptional condition pending.  
31144 Otherwise, what constitutes an exceptional condition is file type-specific. For a file descriptor for  
31145 use with a socket, it is protocol-specific except as noted below. For other file types it is  
31146 implementation-defined. If the operation is meaningless for a particular file type, *pselect()* or  
31147 *select()* shall indicate that the descriptor is ready for read or write operations, and shall indicate  
31148 that the descriptor has no exceptional condition pending.

31149 If a descriptor refers to a socket, the implied input function is the *recvmsg()* function with  
31150 parameters requesting normal and ancillary data, such that the presence of either type shall  
31151 cause the socket to be marked as readable. The presence of out-of-band data shall be checked if |  
31152 the socket option SO\_OOBINLINE has been enabled, as out-of-band data is enqueued with |  
31153 normal data. If the socket is currently listening, then it shall be marked as readable if an |  
31154 incoming connection request has been received, and a call to the *accept()* function shall complete |  
31155 without blocking. |

31156 If a descriptor refers to a socket, the implied output function is the *sendmsg()* function supplying  
31157 an amount of normal data equal to the current value of the SO\_SNDLOWAT option for the  
31158 socket. If a non-blocking call to the *connect()* function has been made for a socket, and the  
31159 connection attempt has either succeeded or failed leaving a pending error, the socket shall be

31160 marked as writable.

31161 A socket shall be considered to have an exceptional condition pending if a receive operation  
 31162 with `O_NONBLOCK` clear for the open file description and with the `MSG_OOB` flag set would  
 31163 return out-of-band data without blocking. (It is protocol-specific whether the `MSG_OOB` flag  
 31164 would be used to read out-of-band data.) A socket shall also be considered to have an  
 31165 exceptional condition pending if an out-of-band data mark is present in the receive queue. Other  
 31166 circumstances under which a socket may be considered to have an exceptional condition  
 31167 pending are protocol-specific and implementation-defined.

31168 If the `readfds`, `writefds`, and `errorfds` arguments are all null pointers and the `timeout` argument is not  
 31169 a null pointer, the `pselect()` or `select()` function shall block for the time specified, or until |  
 31170 interrupted by a signal. If the `readfds`, `writefds`, and `errorfds` arguments are all null pointers and the |  
 31171 `timeout` argument is a null pointer, the `pselect()` or `select()` function shall block until interrupted |  
 31172 by a signal. |

31173 File descriptors associated with regular files shall always select true for ready to read, ready to  
 31174 write, and error conditions.

31175 On failure, the objects pointed to by the `readfds`, `writefds`, and `errorfds` arguments shall not be |  
 31176 modified. If the timeout interval expires without the specified condition being true for any of the |  
 31177 specified file descriptors, the objects pointed to by the `readfds`, `writefds`, and `errorfds` arguments |  
 31178 shall have all bits set to 0. |

31179 File descriptor masks of type `fd_set` can be initialized and tested with `FD_CLR()`, `FD_ISSET()`,  
 31180 `FD_SET()`, and `FD_ZERO()`. It is unspecified whether each of these is a macro or a function. If a  
 31181 macro definition is suppressed in order to access an actual function, or a program defines an  
 31182 external identifier with any of these names, the behavior is undefined.

31183 `FD_CLR(fd, fdsetp)` shall remove the file descriptor `fd` from the set pointed to by `fdsetp`. If `fd` is not |  
 31184 a member of this set, there shall be no effect on the set, nor will an error be returned.

31185 `FD_ISSET(fd, fdsetp)` shall evaluate to non-zero if the file descriptor `fd` is a member of the set |  
 31186 pointed to by `fdsetp`, and shall evaluate to zero otherwise.

31187 `FD_SET(fd, fdsetp)` shall add the file descriptor `fd` to the set pointed to by `fdsetp`. If the file |  
 31188 descriptor `fd` is already in this set, there shall be no effect on the set, nor will an error be returned.

31189 `FD_ZERO(fdsetp)` shall initialize the descriptor set pointed to by `fdsetp` to the null set. No error is |  
 31190 returned if the set is not empty at the time `FD_ZERO()` is invoked.

31191 The behavior of these macros is undefined if the `fd` argument is less than 0 or greater than or  
 31192 equal to `FD_SETSIZE`, or if `fd` is not a valid file descriptor, or if any of the arguments are  
 31193 expressions with side effects.

#### 31194 RETURN VALUE

31195 Upon successful completion, the `pselect()` and `select()` functions shall return the total number of  
 31196 bits set in the bit masks. Otherwise, `-1` shall be returned, and shall set `errno` to indicate the error.

31197 `FD_CLR()`, `FD_SET()`, and `FD_ZERO()` not return a value. `FD_ISSET()` shall return a non-zero  
 31198 value if the bit for the file descriptor `fd` is set in the file descriptor set pointed to by `fdset`, and 0  
 31199 otherwise.

#### 31200 ERRORS

31201 Under the following conditions, `pselect()` and `select()` shall fail and set `errno` to:

31202 [EBADF] One or more of the file descriptor sets specified a file descriptor that is not a  
 31203 valid open file descriptor.

- 31204 [EINTR] The function was interrupted before any of the selected events occurred and  
31205 before the timeout interval expired.
- 31206 XSI If SA\_RESTART has been set for the interrupting signal, it is implementation-  
31207 defined whether the function restarts or returns with [EINTR].
- 31208 [EINVAL] An invalid timeout interval was specified.
- 31209 [EINVAL] The *nfds* argument is less than 0 or greater than FD\_SETSIZE.
- 31210 XSR [EINVAL] One of the specified file descriptors refers to a STREAM or multiplexer that is  
31211 linked (directly or indirectly) downstream from a multiplexer.

31212 **EXAMPLES**

31213 None.

31214 **APPLICATION USAGE**

31215 None.

31216 **RATIONALE**

31217 In previous versions of the Single UNIX Specification, the *select()* function was defined in the  
31218 `<sys/time.h>` header. This is now changed to `<sys/select.h>`. The rationale for this change was  
31219 as follows: the introduction of the *pselect()* function included the `<sys/select.h>` header and the  
31220 `<sys/select.h>` header defines all the related definitions for the *pselect()* and *select()* functions.  
31221 Backwards-compatibility to existing XSI implementations is handled by allowing `<sys/time.h>`  
31222 to include `<sys/select.h>`.

31223 **FUTURE DIRECTIONS**

31224 None.

31225 **SEE ALSO**

31226 *accept()*, *alarm()*, *connect()*, *fcntl()*, *poll()*, *read()*, *recvmsg()*, *sendmsg()*, *setitimer()*, *ualarm()*,  
31227 *write()*, the Base Definitions volume of IEEE Std 1003.1-200x, `<sys/select.h>`, `<sys/time.h>`

31228 **CHANGE HISTORY**

31229 First released in Issue 4, Version 2.

31230 **Issue 5**

31231 Moved from X/OPEN UNIX extension to BASE.

31232 In the ERRORS section, the text has been changed to indicate that [EINVAL] is returned when  
31233 *nfds* is less than 0 or greater than FD\_SETSIZE. It previously stated less than 0, or greater than or  
31234 equal to FD\_SETSIZE.

31235 Text about *timeout* is moved from the APPLICATION USAGE section to the DESCRIPTION.31236 **Issue 6**31237 The Open Group Corrigendum U026/6 is applied, changing the occurrences of *readfs* and *writefs*  
31238 in the *select()* DESCRIPTION to be *readfds* and *writefds*.

31239 Text referring to sockets is added to the DESCRIPTION.

31240 The DESCRIPTION and ERRORS sections are updated so that references to STREAMS are  
31241 marked as part of the XSI STREAMS Option Group.

31242 The following new requirements on POSIX implementations derive from alignment with the  
31243 Single UNIX Specification:

- 31244 • These functions are now mandatory.

31245 The *pselect()* function is added for alignment with IEEE Std 1003.1g-2000 and additional detail  
31246 related to sockets semantics is added to the DESCRIPTION.

- 31247        The *select()* function now requires inclusion of `<sys/select.h>`.
- 31248        The **restrict** keyword is added to the *select()* prototype for alignment with the
- 31249        ISO/IEC 9899:1999 standard.

31250 **NAME**

31251 pthread\_atfork — register fork handlers

31252 **SYNOPSIS**

31253 THR #include &lt;pthread.h&gt;

```
31254 int pthread_atfork(void (*prepare)(void), void (*parent)(void),
31255 void (*child)(void));
31256
```

31257 **DESCRIPTION**

31258 The *pthread\_atfork()* function shall declare fork handlers to be called before and after *fork()*, in  
 31259 the context of the thread that called *fork()*. The *prepare* fork handler shall be called before *fork()*  
 31260 processing commences. The *parent* fork handle shall be called after *fork()* processing completes  
 31261 in the parent process. The *child* fork handler shall be called after *fork()* processing completes in  
 31262 the child process. If no handling is desired at one or more of these three points, the  
 31263 corresponding fork handler address(es) may be set to NULL.

31264 The order of calls to *pthread\_atfork()* is significant. The *parent* and *child* fork handlers shall be  
 31265 called in the order in which they were established by calls to *pthread\_atfork()*. The *prepare* fork  
 31266 handlers shall be called in the opposite order.

31267 **RETURN VALUE**

31268 Upon successful completion, *pthread\_atfork()* shall return a value of zero; otherwise, an error  
 31269 number shall be returned to indicate the error.

31270 **ERRORS**31271 The *pthread\_atfork()* function shall fail if:

31272 [ENOMEM] Insufficient table space exists to record the fork handler addresses.

31273 The *pthread\_atfork()* function shall not return an error code of [EINTR].31274 **EXAMPLES**

31275 None.

31276 **APPLICATION USAGE**

31277 None.

31278 **RATIONALE**

31279 There are at least two serious problems with the semantics of *fork()* in a multi-threaded  
 31280 program. One problem has to do with state (for example, memory) covered by mutexes.  
 31281 Consider the case where one thread has a mutex locked and the state covered by that mutex is  
 31282 inconsistent while another thread calls *fork()*. In the child, the mutex is in the locked state  
 31283 (locked by a nonexistent thread and thus can never be unlocked). Having the child simply  
 31284 reinitialize the mutex is unsatisfactory since this approach does not resolve the question about  
 31285 how to correct or otherwise deal with the inconsistent state in the child.

31286 It is suggested that programs that use *fork()* call an *exec* function very soon afterwards in the  
 31287 child process, thus resetting all states. In the meantime, only a short list of async-signal-safe  
 31288 library routines are promised to be available.

31289 Unfortunately, this solution does not address the needs of multi-threaded libraries. Application  
 31290 programs may not be aware that a multi-threaded library is in use, and they feel free to call any  
 31291 number of library routines between the *fork()* and *exec* calls, just as they always have. Indeed,  
 31292 they may be extant single-threaded programs and cannot, therefore, be expected to obey new  
 31293 restrictions imposed by the threads library.

31294 On the other hand, the multi-threaded library needs a way to protect its internal state during  
 31295 *fork()* in case it is re-entered later in the child process. The problem arises especially in multi-  
 31296 threaded I/O libraries, which are almost sure to be invoked between the *fork()* and *exec* calls to  
 31297 effect I/O redirection. The solution may require locking mutex variables during *fork()*, or it may  
 31298 entail simply resetting the state in the child after the *fork()* processing completes.

31299 The *pthread\_atfork()* function provides multi-threaded libraries with a means to protect  
 31300 themselves from innocent application programs that call *fork()*, and it provides multi-threaded  
 31301 application programs with a standard mechanism for protecting themselves from *fork()* calls in  
 31302 a library routine or the application itself.

31303 The expected usage is that the *prepare* handler acquires all mutex locks and the other two fork  
 31304 handlers release them.

31305 For example, an application can supply a *prepare* routine that acquires the necessary mutexes the  
 31306 library maintains and supply *child* and *parent* routines that release those mutexes, thus ensuring  
 31307 that the child gets a consistent snapshot of the state of the library (and that no mutexes are left  
 31308 stranded). Alternatively, some libraries might be able to supply just a *child* routine that  
 31309 reinitializes the mutexes in the library and all associated states to some known value (for  
 31310 example, what it was when the image was originally executed).

31311 When *fork()* is called, only the calling thread is duplicated in the child process. Synchronization  
 31312 variables remain in the same state in the child as they were in the parent at the time *fork()* was  
 31313 called. Thus, for example, mutex locks may be held by threads that no longer exist in the child  
 31314 process, and any associated states may be inconsistent. The parent process may avoid this by  
 31315 explicit code that acquires and releases locks critical to the child via *pthread\_atfork()*. In addition,  
 31316 any critical threads need to be recreated and reinitialized to the proper state in the child (also via  
 31317 *pthread\_atfork()*).

31318 A higher-level package may acquire locks on its own data structures before invoking lower-level  
 31319 packages. Under this scenario, the order specified for fork handler calls allows a simple rule of  
 31320 initialization for avoiding package deadlock: a package initializes all packages on which it  
 31321 depends before it calls the *pthread\_atfork()* function for itself.

#### 31322 **FUTURE DIRECTIONS**

31323 None.

#### 31324 **SEE ALSO**

31325 *atexit()*, *fork()*, the Base Definitions volume of IEEE Std 1003.1-200x, <sys/types.h>

#### 31326 **CHANGE HISTORY**

31327 First released in Issue 5. Derived from the POSIX Threads Extension.

31328 IEEE PASC Interpretation 1003.1c #4 is applied.

#### 31329 **Issue 6**

31330 The *pthread\_atfork()* function is marked as part of the Threads option.

31331 The <pthread.h> header is added to the SYNOPSIS.

## 31332 NAME

31333 pthread\_attr\_destroy, pthread\_attr\_init — destroy and initialize threads attributes object

## 31334 SYNOPSIS

```
31335 THR #include <pthread.h>
```

```
31336 int pthread_attr_destroy(pthread_attr_t *attr);
```

```
31337 int pthread_attr_init(pthread_attr_t *attr);
```

31338

## 31339 DESCRIPTION

31340 The *pthread\_attr\_destroy()* function shall destroy a thread attributes object. An implementation |  
31341 may cause *pthread\_attr\_destroy()* to set *attr* to an implementation-defined invalid value. A |  
31342 destroyed *attr* attributes object can be reinitialized using *pthread\_attr\_init()*; the results of |  
31343 otherwise referencing the object after it has been destroyed are undefined. |

31344 The *pthread\_attr\_init()* function shall initialize a thread attributes object *attr* with the default |  
31345 value for all of the individual attributes used by a given implementation.

31346 The resulting attributes object (possibly modified by setting individual attribute values), when |  
31347 used by *pthread\_create()* defines the attributes of the thread created. A single attributes object can |  
31348 be used in multiple simultaneous calls to *pthread\_create()*. Results are undefined if |  
31349 *pthread\_attr\_init()* is called specifying an already initialized *attr* attributes object. |

## 31350 RETURN VALUE

31351 Upon successful completion, *pthread\_attr\_destroy()* and *pthread\_attr\_init()* shall return a value of |  
31352 0; otherwise, an error number shall be returned to indicate the error.

## 31353 ERRORS

31354 The *pthread\_attr\_init()* function shall fail if:

31355 [ENOMEM] Insufficient memory exists to initialize the thread attributes object.

31356 These functions shall not return an error code of [EINTR].

## 31357 EXAMPLES

31358 None.

## 31359 APPLICATION USAGE

31360 None.

## 31361 RATIONALE

31362 Attributes objects are provided for threads, mutexes, and condition variables as a mechanism to |  
31363 support probable future standardization in these areas without requiring that the function itself |  
31364 be changed.

31365 Attributes objects provide clean isolation of the configurable aspects of threads. For example, |  
31366 “stack size” is an important attribute of a thread, but it cannot be expressed portably. When |  
31367 porting a threaded program, stack sizes often need to be adjusted. The use of attributes objects |  
31368 can help by allowing the changes to be isolated in a single place, rather than being spread across |  
31369 every instance of thread creation.

31370 Attributes objects can be used to set up “classes” of threads with similar attributes; for example, |  
31371 “threads with large stacks and high priority” or “threads with minimal stacks”. These classes |  
31372 can be defined in a single place and then referenced wherever threads need to be created. |  
31373 Changes to “class” decisions become straightforward, and detailed analysis of each |  
31374 *pthread\_create()* call is not required.

31375 The attributes objects are defined as opaque types as an aid to extensibility. If these objects had |  
31376 been specified as structures, adding new attributes would force recompilation of all multi-

31377 threaded programs when the attributes objects are extended; this might not be possible if  
31378 different program components were supplied by different vendors.

31379 Additionally, opaque attributes objects present opportunities for improving performance.  
31380 Argument validity can be checked once when attributes are set, rather than each time a thread is  
31381 created. Implementations often need to cache kernel objects that are expensive to create.  
31382 Opaque attributes objects provide an efficient mechanism to detect when cached objects become  
31383 invalid due to attribute changes.

31384 Since assignment is not necessarily defined on a given opaque type, implementation-defined  
31385 default values cannot be defined in a portable way. The solution to this problem is to allow  
31386 attributes objects to be initialized dynamically by attributes object initialization functions, so  
31387 that default values can be supplied automatically by the implementation.

31388 The following proposal was provided as a suggested alternative to the supplied attributes:

- 31389 1. Maintain the style of passing a parameter formed by the bitwise-inclusive OR of flags to  
31390 the initialization routines (*pthread\_create()*, *pthread\_mutex\_init()*, *pthread\_cond\_init()*). The  
31391 parameter containing the flags should be an opaque type for extensibility. If no flags are  
31392 set in the parameter, then the objects are created with default characteristics. An  
31393 implementation may specify implementation-defined flag values and associated behavior.
- 31394 2. If further specialization of mutexes and condition variables is necessary, implementations  
31395 may specify additional procedures that operate on the **pthread\_mutex\_t** and  
31396 **pthread\_cond\_t** objects (instead of on attributes objects).

31397 The difficulties with this solution are:

- 31398 1. A bitmask is not opaque if bits have to be set into bitvector attributes objects using  
31399 explicitly-coded bitwise-inclusive OR operations. If the set of options exceeds an **int**,  
31400 application programmers need to know the location of each bit. If bits are set or read by  
31401 encapsulation (that is, *get* and *set* functions), then the bitmask is merely an  
31402 implementation of attributes objects as currently defined and should not be exposed to the  
31403 programmer.
- 31404 2. Many attributes are not Boolean or very small integral values. For example, scheduling  
31405 policy may be placed in 3-bit or 4-bit, but priority requires 5-bit or more, thereby taking up  
31406 at least 8 bits out of a possible 16 bits on machines with 16-bit integers. Because of this, the  
31407 bitmask can only reasonably control whether particular attributes are set or not, and it  
31408 cannot serve as the repository of the value itself. The value needs to be specified as a  
31409 function parameter (which is non-extensible), or by setting a structure field (which is non-  
31410 opaque), or by *get* and *set* functions (making the bitmask a redundant addition to the  
31411 attributes objects).

31412 Stack size is defined as an optional attribute because the very notion of a stack is inherently  
31413 machine-dependent. Some implementations may not be able to change the size of the stack, for  
31414 example, and others may not need to because stack pages may be discontinuous and can be  
31415 allocated and released on demand.

31416 The attribute mechanism has been designed in large measure for extensibility. Future extensions  
31417 to the attribute mechanism or to any attributes object defined in this volume of  
31418 IEEE Std 1003.1-200x has to be done with care so as not to affect binary-compatibility.

31419 Attributes objects, even if allocated by means of dynamic allocation functions such as *malloc()*,  
31420 may have their size fixed at compile time. This means, for example, a *pthread\_create()* in an  
31421 implementation with extensions to the **pthread\_attr\_t** cannot look beyond the area that the  
31422 binary application assumes is valid. This suggests that implementations should maintain a size  
31423 field in the attributes object, as well as possibly version information, if extensions in different

31424 directions (possibly by different vendors) are to be accommodated.

31425 **FUTURE DIRECTIONS**

31426 None.

31427 **SEE ALSO**

31428 *pthread\_attr\_getstackaddr()*, *pthread\_attr\_getstacksize()*, *pthread\_attr\_getdetachstate()*,  
31429 *pthread\_create()*, the Base Definitions volume of IEEE Std 1003.1-200x, <pthread.h>

31430 **CHANGE HISTORY**

31431 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

31432 **Issue 6**

31433 The *pthread\_attr\_destroy()* and *pthread\_attr\_init()* functions marked as part of the Threads  
31434 option.

31435 IEEE PASC Interpretation 1003.1 #107 is applied, noting that the effect of initializing an already  
31436 initialized thread attributes object is undefined.

31437 **NAME**

31438 pthread\_attr\_getdetachstate, pthread\_attr\_setdetachstate — get and set detachstate attribute

31439 **SYNOPSIS**

31440 THR #include &lt;pthread.h&gt;

31441 int pthread\_attr\_getdetachstate(const pthread\_attr\_t \*attr,

31442 int \*detachstate);

31443 int pthread\_attr\_setdetachstate(pthread\_attr\_t \*attr, int detachstate);

31444

31445 **DESCRIPTION**

31446 The *detachstate* attribute controls whether the thread is created in a detached state. If the thread  
 31447 is created detached, then use of the ID of the newly created thread by the *pthread\_detach()* or  
 31448 *pthread\_join()* function is an error.

31449 The *pthread\_attr\_getdetachstate()* and *pthread\_attr\_setdetachstate()* functions, respectively, shall  
 31450 get and set the *detachstate* attribute in the *attr* object.

31451 For *pthread\_attr\_getdetachstate()*, *detachstate* shall be set to either  
 31452 PTHREAD\_CREATE\_DETACHED or PTHREAD\_CREATE\_JOINABLE.

31453 For *pthread\_attr\_setdetachstate()*, the application shall set *detachstate* to either  
 31454 PTHREAD\_CREATE\_DETACHED or PTHREAD\_CREATE\_JOINABLE.

31455 A value of PTHREAD\_CREATE\_DETACHED shall cause all threads created with *attr* to be in  
 31456 the detached state, whereas using a value of PTHREAD\_CREATE\_JOINABLE shall cause all  
 31457 threads created with *attr* to be in the joinable state. The default value of the *detachstate* attribute  
 31458 shall be PTHREAD\_CREATE\_JOINABLE.

31459 **RETURN VALUE**

31460 Upon successful completion, *pthread\_attr\_getdetachstate()* and *pthread\_attr\_setdetachstate()* shall  
 31461 return a value of 0; otherwise, an error number shall be returned to indicate the error.

31462 The *pthread\_attr\_getdetachstate()* function stores the value of the *detachstate* attribute in *detachstate*  
 31463 if successful.

31464 **ERRORS**

31465 The *pthread\_attr\_setdetachstate()* function shall fail if:

31466 [EINVAL] The value of *detachstate* was not valid

31467 These functions shall not return an error code of [EINTR].

31468 **EXAMPLES**

31469 None.

31470 **APPLICATION USAGE**

31471 None.

31472 **RATIONALE**

31473 None.

31474 **FUTURE DIRECTIONS**

31475 None.

31476 **SEE ALSO**

31477 *pthread\_attr\_destroy()*, *pthread\_attr\_getstackaddr()*, *pthread\_attr\_getstacksize()*, *pthread\_create()*, the  
 31478 Base Definitions volume of IEEE Std 1003.1-200x, <pthread.h>

31479 **CHANGE HISTORY**

31480 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

31481 **Issue 6**

31482 The *pthread\_attr\_setdetachstate()* and *pthread\_attr\_getdetachstate()* functions are marked as part of  
31483 the Threads option.

31484 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

31485 **NAME**

31486 pthread\_attr\_getguardsize, pthread\_attr\_setguardsize — get and set the thread guardsize  
 31487 attribute

31488 **SYNOPSIS**

```
31489 XSI #include <pthread.h>
31490
31490 int pthread_attr_getguardsize(const pthread_attr_t *restrict attr,
31491 size_t *restrict guardsize);
31492 int pthread_attr_setguardsize(pthread_attr_t *attr,
31493 size_t guardsize);
31494
```

31495 **DESCRIPTION**

31496 The *pthread\_attr\_getguardsize()* function shall get the *guardsize* attribute in the *attr* object. This  
 31497 attribute shall be returned in the *guardsize* parameter.

31498 The *pthread\_attr\_setguardsize()* function shall set the *guardsize* attribute in the *attr* object. The new  
 31499 value of this attribute shall be obtained from the *guardsize* parameter. If *guardsize* is zero, a guard  
 31500 area shall not be provided for threads created with *attr*. If *guardsize* is greater than zero, a guard  
 31501 area of at least size *guardsize* bytes shall be provided for each thread created with *attr*.

31502 The *guardsize* attribute controls the size of the guard area for the created thread's stack. The  
 31503 *guardsize* attribute provides protection against overflow of the stack pointer. If a thread's stack is  
 31504 created with guard protection, the implementation allocates extra memory at the overflow end  
 31505 of the stack as a buffer against stack overflow of the stack pointer. If an application overflows  
 31506 into this buffer an error shall result (possibly in a SIGSEGV signal being delivered to the thread).

31507 A conforming implementation may round up the value contained in *guardsize* to a multiple of  
 31508 the configurable system variable {PAGESIZE} (see <sys/mman.h>). If an implementation  
 31509 rounds up the value of *guardsize* to a multiple of {PAGESIZE}, a call to *pthread\_attr\_getguardsize()*  
 31510 specifying *attr* shall store in the *guardsize* parameter the guard size specified by the previous  
 31511 *pthread\_attr\_setguardsize()* function call.

31512 The default value of the *guardsize* attribute is {PAGESIZE} bytes. The actual value of {PAGESIZE}  
 31513 is implementation-defined.

31514 If the *stackaddr* or *stack* attribute has been set (that is, the caller is allocating and managing its  
 31515 own thread stacks), the *guardsize* attribute shall be ignored and no protection shall be provided  
 31516 by the implementation. It is the responsibility of the application to manage stack overflow along  
 31517 with stack allocation and management in this case.

31518 **RETURN VALUE**

31519 If successful, the *pthread\_attr\_getguardsize()* and *pthread\_attr\_setguardsize()* functions shall return  
 31520 zero; otherwise, an error number shall be returned to indicate the error.

31521 **ERRORS**

31522 The *pthread\_attr\_getguardsize()* and *pthread\_attr\_setguardsize()* functions shall fail if:

31523 [EINVAL] The attribute *attr* is invalid.

31524 [EINVAL] The parameter *guardsize* is invalid.

31525 These functions shall not return an error code of [EINTR].

## 31526 EXAMPLES

31527 None.

## 31528 APPLICATION USAGE

31529 None.

## 31530 RATIONALE

31531 The *guardsize* attribute is provided to the application for two reasons:

- 31532 1. Overflow protection can potentially result in wasted system resources. An application  
31533 that creates a large number of threads, and which knows its threads never overflow their  
31534 stack, can save system resources by turning off guard areas.
- 31535 2. When threads allocate large data structures on the stack, large guard areas may be needed  
31536 to detect stack overflow.

## 31537 FUTURE DIRECTIONS

31538 None.

## 31539 SEE ALSO

31540 The Base Definitions volume of IEEE Std 1003.1-200x, `<pthread.h>`, `<sys/mman.h>`

## 31541 CHANGE HISTORY

31542 First released in Issue 5.

### 31543 Issue 6

31544 In the ERRORS section, a third [EINVAL] error condition is removed as it is covered by the  
31545 second error condition.

31546 The **restrict** keyword is added to the *pthread\_attr\_getguardsize()* prototype for alignment with the  
31547 ISO/IEC 9899:1999 standard.

31548 **NAME**

31549 pthread\_attr\_getinheritsched, pthread\_attr\_setinheritsched — get and set inheritsched attribute  
 31550 (**REALTIME THREADS**)

31551 **SYNOPSIS**

31552 THR TPS #include <pthread.h>

```
31553 int pthread_attr_getinheritsched(const pthread_attr_t *restrict attr,
31554                               int *restrict inheritsched);
31555 int pthread_attr_setinheritsched(pthread_attr_t *attr,
31556                               int inheritsched);
31557
```

31558 **DESCRIPTION**

31559 The *pthread\_attr\_getinheritsched()*, and *pthread\_attr\_setinheritsched()* functions, respectively, shall  
 31560 get and set the *inheritsched* attribute in the *attr* argument.

31561 When the attributes objects are used by *pthread\_create()*, the *inheritsched* attribute determines  
 31562 how the other scheduling attributes of the created thread shall be set.

31563 **PTHREAD\_INHERIT\_SCHED**

31564 Specifies that the thread scheduling attributes shall be inherited from the creating thread,  
 31565 and the scheduling attributes in this *attr* argument shall be ignored.

31566 **PTHREAD\_EXPLICIT\_SCHED**

31567 Specifies that the thread scheduling attributes shall be set to the corresponding values from  
 31568 this attributes object.

31569 The symbols **PTHREAD\_INHERIT\_SCHED** and **PTHREAD\_EXPLICIT\_SCHED** are defined in  
 31570 the <**pthread.h**> header.

31571 The following “thread scheduling attributes” defined by IEEE Std 1003.1-200x are affected by  
 31572 the *inheritsched* attribute: scheduling policy (*schedpolicy*), scheduling parameters (*schedparam*),  
 31573 and scheduling contention scope (*contentionscope*).

31574 **RETURN VALUE**

31575 If successful, the *pthread\_attr\_getinheritsched()* and *pthread\_attr\_setinheritsched()* functions shall  
 31576 return zero; otherwise, an error number shall be returned to indicate the error.

31577 **ERRORS**

31578 The *pthread\_attr\_setinheritsched()* function may fail if:

31579 [EINVAL] The value of *inheritsched* is not valid.

31580 [ENOTSUP] An attempt was made to set the attribute to an unsupported value.

31581 These functions shall not return an error code of [EINTR].

31582 **EXAMPLES**

31583 None.

31584 **APPLICATION USAGE**

31585 After these attributes have been set, a thread can be created with the specified attributes using  
 31586 *pthread\_create()*. Using these routines does not affect the current running thread.

31587 **RATIONALE**

31588 None.

## 31589 FUTURE DIRECTIONS

31590 None.

## 31591 SEE ALSO

31592 *pthread\_attr\_destroy()*, *pthread\_attr\_getscope()*, *pthread\_attr\_getschedpolicy()*,  
31593 *pthread\_attr\_getschedparam()*, *pthread\_create()*, the Base Definitions volume of  
31594 IEEE Std 1003.1-200x, <pthread.h>, <sched.h>

## 31595 CHANGE HISTORY

31596 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

31597 Marked as part of the Realtime Threads Feature Group.

## 31598 Issue 6

31599 The *pthread\_attr\_getinheritsched()* and *pthread\_attr\_setinheritsched()* functions are marked as part  
31600 of the Threads and Thread Execution Scheduling options.

31601 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
31602 implementation does not support the Thread Execution Scheduling option.

31603 The **restrict** keyword is added to the *pthread\_attr\_getinheritsched()* prototype for alignment with  
31604 the ISO/IEC 9899:1999 standard.

31605 **NAME**

31606 pthread\_attr\_getschedparam, pthread\_attr\_setschedparam — get and set schedparam attribute

31607 **SYNOPSIS**

31608 THR #include <pthread.h>

```
31609 int pthread_attr_getschedparam(const pthread_attr_t *restrict attr,
31610 struct sched_param *restrict param);
```

```
31611 int pthread_attr_setschedparam(pthread_attr_t *restrict attr,
31612 const struct sched_param *restrict param);
```

31613

31614 **DESCRIPTION**

31615 The *pthread\_attr\_getschedparam()*, and *pthread\_attr\_setschedparam()* functions, respectively, shall  
 31616 get and set the scheduling parameter attributes in the *attr* argument. The contents of the *param* |  
 31617 structure are defined in the <sched.h> header. For the SCHED\_FIFO and SCHED\_RR policies, |  
 31618 the only required member of *param* is *sched\_priority*.

31619 TSP For the SCHED\_SPORADIC policy, the required members of the *param* structure are  
 31620 *sched\_priority*, *sched\_ss\_low\_priority*, *sched\_ss\_repl\_period*, *sched\_ss\_init\_budget*, and  
 31621 *sched\_ss\_max\_repl*. The specified *sched\_ss\_repl\_period* must be greater than or equal to the  
 31622 specified *sched\_ss\_init\_budget* for the function to succeed; if it is not, then the function shall fail.  
 31623 The value of *sched\_ss\_max\_repl* shall be within the inclusive range [1,{SS\_REPL\_MAX}] for the  
 31624 function to succeed; if not, the function shall fail.

31625 **RETURN VALUE**

31626 If successful, the *pthread\_attr\_getschedparam()* and *pthread\_attr\_setschedparam()* functions shall  
 31627 return zero; otherwise, an error number shall be returned to indicate the error.

31628 **ERRORS**

31629 The *pthread\_attr\_setschedparam()* function may fail if:

31630 [EINVAL] The value of *param* is not valid.

31631 [ENOTSUP] An attempt was made to set the attribute to an unsupported value.

31632 These functions shall not return an error code of [EINTR].

31633 **EXAMPLES**

31634 None.

31635 **APPLICATION USAGE**

31636 After these attributes have been set, a thread can be created with the specified attributes using  
 31637 *pthread\_create()*. Using these routines does not affect the current running thread.

31638 **RATIONALE**

31639 None.

31640 **FUTURE DIRECTIONS**

31641 None.

31642 **SEE ALSO**

31643 *pthread\_attr\_destroy()*, *pthread\_attr\_getscope()*, *pthread\_attr\_getinheritsched()*,  
 31644 *pthread\_attr\_getschedpolicy()*, *pthread\_create()*, the Base Definitions volume of  
 31645 IEEE Std 1003.1-200x, <pthread.h>, <sched.h>

## 31646 CHANGE HISTORY

31647 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

## 31648 Issue 6

31649 The *pthread\_attr\_getschedparam()* and *pthread\_attr\_setschedparam()* functions are marked as part  
31650 of the Threads option.

31651 The SCHED\_SPORADIC scheduling policy is added for alignment with IEEE Std 1003.1d-1999.

31652 The **restrict** keyword is added to the *pthread\_attr\_getschedparam()* and  
31653 *pthread\_attr\_setschedparam()* prototypes for alignment with the ISO/IEC 9899:1999 standard.

31654 **NAME**

31655 pthread\_attr\_getschedpolicy, pthread\_attr\_setschedpolicy — get and set schedpolicy attribute  
 31656 (**REALTIME THREADS**)

31657 **SYNOPSIS**

```
31658 THR TPS #include <pthread.h>
31659
31659 int pthread_attr_getschedpolicy(const pthread_attr_t *restrict attr,
31660 int *restrict policy);
31661 int pthread_attr_setschedpolicy(pthread_attr_t *attr, int policy);
31662
```

31663 **DESCRIPTION**

31664 The *pthread\_attr\_getschedpolicy()* and *pthread\_attr\_setschedpolicy()* functions, respectively, shall  
 31665 get and set the *schedpolicy* attribute in the *attr* argument.

31666 The supported values of *policy* shall include SCHED\_FIFO, SCHED\_RR, and SCHED\_OTHER, |  
 31667 which are defined in the <*sched.h*> header. When threads executing with the scheduling policy |  
 31668 TSP SCHED\_FIFO, SCHED\_RR, or SCHED\_SPORADIC are waiting on a mutex, they shall acquire |  
 31669 the mutex in priority order when the mutex is unlocked. |

31670 **RETURN VALUE**

31671 If successful, the *pthread\_attr\_getschedpolicy()* and *pthread\_attr\_setschedpolicy()* functions shall  
 31672 return zero; otherwise, an error number shall be returned to indicate the error.

31673 **ERRORS**

31674 The *pthread\_attr\_setschedpolicy()* function may fail if:

- 31675 [EINVAL] The value of *policy* is not valid.
- 31676 [ENOTSUP] An attempt was made to set the attribute to an unsupported value.
- 31677 These functions shall not return an error code of [EINTR].

31678 **EXAMPLES**

31679 None.

31680 **APPLICATION USAGE**

31681 After these attributes have been set, a thread can be created with the specified attributes using  
 31682 *pthread\_create()*. Using these routines does not affect the current running thread.

31683 **RATIONALE**

31684 None.

31685 **FUTURE DIRECTIONS**

31686 None.

31687 **SEE ALSO**

31688 *pthread\_attr\_destroy()*, *pthread\_attr\_getscope()*, *pthread\_attr\_getinheritsched()*,  
 31689 *pthread\_attr\_getschedparam()*, *pthread\_create()*, the Base Definitions volume of  
 31690 IEEE Std 1003.1-200x, <*pthread.h*>, <*sched.h*>

31691 **CHANGE HISTORY**

- 31692 First released in Issue 5. Included for alignment with the POSIX Threads Extension.
- 31693 Marked as part of the Realtime Threads Feature Group.

31694 **Issue 6**

31695 The *pthread\_attr\_getschedpolicy()* and *pthread\_attr\_setschedpolicy()* functions are marked as part of  
31696 the Threads and Thread Execution Scheduling options.

31697 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
31698 implementation does not support the Thread Execution Scheduling option.

31699 The SCHED\_SPORADIC scheduling policy is added for alignment with IEEE Std 1003.1d-1999.

31700 The **restrict** keyword is added to the *pthread\_attr\_getschedpolicy()* prototype for alignment with  
31701 the ISO/IEC 9899:1999 standard.

31702 **NAME**

31703 pthread\_attr\_getscope, pthread\_attr\_setscope — get and set contention scope attribute  
 31704 (**REALTIME THREADS**)

31705 **SYNOPSIS**

31706 THR TPS #include <pthread.h>

```
31707 int pthread_attr_getscope(const pthread_attr_t *restrict attr,
31708                          int *restrict contention_scope);
31709 int pthread_attr_setscope(pthread_attr_t *attr, int contention_scope);
31710
```

31711 **DESCRIPTION**

31712 The *pthread\_attr\_getscope()* and *pthread\_attr\_setscope()* functions, respectively, shall get and set  
 31713 the *contention\_scope* attribute in the *attr* object.

31714 The *contention\_scope* attribute may have the values PTHREAD\_SCOPE\_SYSTEM, signifying  
 31715 system scheduling contention scope, or PTHREAD\_SCOPE\_PROCESS, signifying process  
 31716 scheduling contention scope. The symbols PTHREAD\_SCOPE\_SYSTEM and  
 31717 PTHREAD\_SCOPE\_PROCESS are defined in the <pthread.h> header.

31718 **RETURN VALUE**

31719 If successful, the *pthread\_attr\_getscope()* and *pthread\_attr\_setscope()* functions shall return zero;  
 31720 otherwise, an error number shall be returned to indicate the error.

31721 **ERRORS**

31722 The *pthread\_attr\_setscope()* function may fail if:

31723 [EINVAL] The value of *contention\_scope* is not valid.

31724 [ENOTSUP] An attempt was made to set the attribute to an unsupported value.

31725 These functions shall not return an error code of [EINTR].

31726 **EXAMPLES**

31727 None.

31728 **APPLICATION USAGE**

31729 After these attributes have been set, a thread can be created with the specified attributes using  
 31730 *pthread\_create()*. Using these routines does not affect the current running thread.

31731 **RATIONALE**

31732 None.

31733 **FUTURE DIRECTIONS**

31734 None.

31735 **SEE ALSO**

31736 *pthread\_attr\_destroy()*, *pthread\_attr\_getinheritsched()*, *pthread\_attr\_getschedpolicy()*,  
 31737 *pthread\_attr\_getschedparam()*, *pthread\_create()*, the Base Definitions volume of  
 31738 IEEE Std 1003.1-200x, <pthread.h>, <sched.h>

31739 **CHANGE HISTORY**

31740 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

31741 Marked as part of the Realtime Threads Feature Group.

31742 **Issue 6**

31743 The *pthread\_attr\_getscope()* and *pthread\_attr\_setscope()* functions are marked as part of the  
31744 Threads and Thread Execution Scheduling options.

31745 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
31746 implementation does not support the Thread Execution Scheduling option.

31747 The **restrict** keyword is added to the *pthread\_attr\_getscope()* prototype for alignment with the  
31748 ISO/IEC 9899:1999 standard.

31749 **NAME**

31750 pthread\_attr\_getstack, pthread\_attr\_setstack — get and set stack attributes

31751 **SYNOPSIS**

31752 THR #include &lt;pthread.h&gt;

```

31753 TSA TSS int pthread_attr_getstack(const pthread_attr_t *restrict attr,
31754 void **restrict stackaddr, size_t *restrict stacksize);
31755 int pthread_attr_setstack(pthread_attr_t *attr, void *stackaddr,
31756 size_t stacksize);
31757

```

31758 **DESCRIPTION**

31759 The *pthread\_attr\_getstack()* and *pthread\_attr\_setstack()* functions, respectively, shall get and set  
 31760 the thread creation stack attributes *stackaddr* and *stacksize* in the *attr* object.

31761 The stack attributes specify the area of storage to be used for the created thread's stack. The base  
 31762 (lowest addressable byte) of the storage shall be *stackaddr*, and the size of the storage shall be  
 31763 *stacksize* bytes. The *stacksize* shall be at least {PTHREAD\_STACK\_MIN}. The *stackaddr* shall be  
 31764 aligned appropriately to be used as a stack; for example, *pthread\_attr\_setstack()* may fail with  
 31765 [EINVAL] if (*stackaddr* & 0x7) is not 0. All pages within the stack described by *stackaddr* and  
 31766 *stacksize* shall be both readable and writable by the thread.

31767 **RETURN VALUE**

31768 Upon successful completion, these functions shall return a value of 0; otherwise, an error  
 31769 number shall be returned to indicate the error.

31770 The *pthread\_attr\_getstack()* function shall store the stack attribute values in *stackaddr* and *stacksize*  
 31771 if successful.

31772 **ERRORS**

31773 The *pthread\_attr\_setstack()* function shall fail if:

31774 [EINVAL] The value of *stacksize* is less than {PTHREAD\_STACK\_MIN} or exceeds an  
 31775 implementation-defined limit.

31776 The *pthread\_attr\_setstack()* function may fail if:

31777 [EINVAL] The value of *stackaddr* does not have proper alignment to be used as a stack, or  
 31778 if (*stackaddr* + *stacksize*) lacks proper alignment.

31779 [EACCES] The stack page(s) described by *stackaddr* and *stacksize* are not both readable  
 31780 and writable by the thread.

31781 These functions shall not return an error code of [EINTR].

31782 **EXAMPLES**

31783 None.

31784 **APPLICATION USAGE**

31785 These functions are appropriate for use by applications in an environment where the stack for a  
 31786 thread must be placed in some particular region of memory.

31787 While it might seem that an application could detect stack overflow by providing a protected  
 31788 page outside the specified stack region, this cannot be done portably. Implementations are free  
 31789 to place the thread's initial stack pointer anywhere within the specified region to accommodate  
 31790 the machine's stack pointer behavior and allocation requirements. Furthermore, on some  
 31791 architectures, such as the IA-64, "overflow" might mean that two separate stack pointers  
 31792 allocated within the region will overlap somewhere in the middle of the region.

31793 **RATIONALE**

31794       None.

31795 **FUTURE DIRECTIONS**

31796       None.

31797 **SEE ALSO**

31798       *pthread\_attr\_init()*, *pthread\_attr\_setdetachstate()*, *pthread\_attr\_setstacksize()*, *pthread\_create()*, the  
31799       Base Definitions volume of IEEE Std 1003.1-200x, <limits.h>, <pthread.h>

31800 **CHANGE HISTORY**

31801       First released in Issue 6. Developed as an XSI extension and brought into the BASE by IEEE

31802       PASC Interpretation 1003.1 #101.

31803 **NAME**

31804 pthread\_attr\_getstackaddr, pthread\_attr\_setstackaddr — get and set stackaddr attribute

31805 **SYNOPSIS**

31806 THR TSA #include &lt;pthread.h&gt;

```
31807 OB int pthread_attr_getstackaddr(const pthread_attr_t *restrict attr,
31808 void **restrict stackaddr);
31809 int pthread_attr_setstackaddr(pthread_attr_t *attr, void *stackaddr);
31810
```

31811 **DESCRIPTION**

31812 The *pthread\_attr\_getstackaddr()* and *pthread\_attr\_setstackaddr()* functions, respectively, shall get  
 31813 and set the thread creation *stackaddr* attribute in the *attr* object.

31814 The *stackaddr* attribute specifies the location of storage to be used for the created thread's stack.  
 31815 The size of the storage shall be at least {PTHREAD\_STACK\_MIN}.

31816 **RETURN VALUE**

31817 Upon successful completion, *pthread\_attr\_getstackaddr()* and *pthread\_attr\_setstackaddr()* shall  
 31818 return a value of 0; otherwise, an error number shall be returned to indicate the error.

31819 The *pthread\_attr\_getstackaddr()* function stores the *stackaddr* attribute value in *stackaddr* if  
 31820 successful.

31821 **ERRORS**

31822 No errors are defined.

31823 These functions shall not return an error code of [EINTR].

31824 **EXAMPLES**

31825 None.

31826 **APPLICATION USAGE**

31827 The specification of the *stackaddr* attribute presents several ambiguities that make portable use of  
 31828 these interfaces impossible. The description of the single address parameter as a “stack” does  
 31829 not specify a particular relationship between the address and the “stack” implied by that  
 31830 address. For example, the address may be taken as the low memory address of a buffer intended  
 31831 for use as a stack, or it may be taken as the address to be used as the initial stack pointer register  
 31832 value for the new thread. These two are not the same except for a machine on which the stack  
 31833 grows “up” from low memory to high, and on which a “push” operation first stores the value in  
 31834 memory and then increments the stack pointer register. Further, on a machine where the stack  
 31835 grows “down” from high memory to low, interpretation of the address as the “low memory”  
 31836 address requires a determination of the intended size of the stack. IEEE Std 1003.1-200x has  
 31837 introduced the new interfaces *pthread\_attr\_setstack()* and *pthread\_attr\_getstack()* to resolve these  
 31838 ambiguities.

31839 **RATIONALE**

31840 None.

31841 **FUTURE DIRECTIONS**

31842 None.

31843 **SEE ALSO**

31844 *pthread\_attr\_destroy()*, *pthread\_attr\_getdetachstate()*, *pthread\_attr\_getstack()*,  
 31845 *pthread\_attr\_getstacksize()*, *pthread\_attr\_setstack()*, *pthread\_create()*, the Base Definitions volume  
 31846 of IEEE Std 1003.1-200x, <limits.h>, <pthread.h>

## 31847 CHANGE HISTORY

31848 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

## 31849 Issue 6

31850 The *pthread\_attr\_getstackaddr()* and *pthread\_attr\_setstackaddr()* functions are marked as part of  
31851 the Threads and Thread Stack Address Attribute options.

31852 The **restrict** keyword is added to the *pthread\_attr\_getstackaddr()* prototype for alignment with the  
31853 ISO/IEC 9899:1999 standard.

31854 These functions are marked obsolescent.

31855 **NAME**

31856 pthread\_attr\_getstacksize, pthread\_attr\_setstacksize — get and set stacksize attribute

31857 **SYNOPSIS**

31858 THR TSA #include <pthread.h>

```
31859 int pthread_attr_getstacksize(const pthread_attr_t *restrict attr,
31860 size_t *restrict stacksize);
31861 int pthread_attr_setstacksize(pthread_attr_t *attr, size_t stacksize);
31862
```

31863 **DESCRIPTION**

31864 The *pthread\_attr\_getstacksize()* and *pthread\_attr\_setstacksize()* functions, respectively, shall get  
31865 and set the thread creation *stacksize* attribute in the *attr* object.

31866 The *stacksize* attribute shall define the minimum stack size (in bytes) allocated for the created  
31867 threads stack.

31868 **RETURN VALUE**

31869 Upon successful completion, *pthread\_attr\_getstacksize()* and *pthread\_attr\_setstacksize()* shall  
31870 return a value of 0; otherwise, an error number shall be returned to indicate the error.

31871 The *pthread\_attr\_getstacksize()* function stores the *stacksize* attribute value in *stacksize* if  
31872 successful.

31873 **ERRORS**

31874 The *pthread\_attr\_setstacksize()* function shall fail if:

31875 [EINVAL] The value of *stacksize* is less than {PTHREAD\_STACK\_MIN} or exceeds a  
31876 system-imposed limit.

31877 These functions shall not return an error code of [EINTR].

31878 **EXAMPLES**

31879 None.

31880 **APPLICATION USAGE**

31881 None.

31882 **RATIONALE**

31883 None.

31884 **FUTURE DIRECTIONS**

31885 None.

31886 **SEE ALSO**

31887 *pthread\_attr\_destroy()*, *pthread\_attr\_getstackaddr()*, *pthread\_attr\_getdetachstate()*, *pthread\_create()*,  
31888 the Base Definitions volume of IEEE Std 1003.1-200x, <limits.h>, <pthread.h>

31889 **CHANGE HISTORY**

31890 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

31891 **Issue 6**

31892 The *pthread\_attr\_getstacksize()* and *pthread\_attr\_setstacksize()* functions are marked as part of the  
31893 Threads and Thread Stack Address Attribute options.

31894 The **restrict** keyword is added to the *pthread\_attr\_getstacksize()* prototype for alignment with the  
31895 ISO/IEC 9899:1999 standard.

31896 **NAME**

31897 pthread\_attr\_init — initialize threads attributes object

31898 **SYNOPSIS**

31899 THR #include <pthread.h>

31900 int pthread\_attr\_init(pthread\_attr\_t \*attr);

31901

31902 **DESCRIPTION**

31903 Refer to *pthread\_attr\_destroy()*.

31904 **NAME**

31905 pthread\_attr\_setdetachstate — set detachstate attribute

31906 **SYNOPSIS**

31907 THR #include &lt;pthread.h&gt;

31908 int pthread\_attr\_setdetachstate(pthread\_attr\_t \*attr, int detachstate);

31909

31910 **DESCRIPTION**31911 Refer to *pthread\_attr\_getdetachstate()*.

31912 **NAME**

31913 pthread\_attr\_setguardsize — set thread guardsize attribute

31914 **SYNOPSIS**

31915 XSI #include <pthread.h>

```
31916 int pthread_attr_setguardsize(pthread_attr_t *attr,  
31917 size_t guardsize);
```

31918

31919 **DESCRIPTION**

31920 Refer to *pthread\_attr\_getguardsize()*.

31921 **NAME**

31922 pthread\_attr\_setinheritsched — set inheritsched attribute (**REALTIME THREADS**)

31923 **SYNOPSIS**

31924 THR TPS #include <pthread.h>

```
31925 int pthread_attr_setinheritsched(pthread_attr_t *attr,  
31926 int inheritsched);
```

31927

31928 **DESCRIPTION**

31929 Refer to *pthread\_attr\_getinheritsched()*.

31930 **NAME**

31931 pthread\_attr\_setschedparam — set schedparam attribute

31932 **SYNOPSIS**

31933 THR #include <pthread.h>

```
31934 int pthread_attr_setschedparam(pthread_attr_t *restrict attr,  
31935                               const struct sched_param *restrict param);
```

31936

31937 **DESCRIPTION**

31938 Refer to *pthread\_attr\_getschedparam()*.

31939 **NAME**31940 pthread\_attr\_setschedpolicy — set schedpolicy attribute (**REALTIME THREADS**)31941 **SYNOPSIS**

31942 THR TPS #include &lt;pthread.h&gt;

31943 int pthread\_attr\_setschedpolicy(pthread\_attr\_t \*attr, int policy);

31944

31945 **DESCRIPTION**31946 Refer to *pthread\_attr\_getschedpolicy()*.

31947 **NAME**

31948 pthread\_attr\_setscope — set contentionscope attribute (**REALTIME THREADS**)

31949 **SYNOPSIS**

31950 THR TPS #include <pthread.h>

31951 int pthread\_attr\_setscope(pthread\_attr\_t \*attr, int contentionscope);

31952

31953 **DESCRIPTION**

31954 Refer to *pthread\_attr\_getscope()*.

31955 **NAME**

31956 pthread\_attr\_setstack — set stack attribute

31957 **SYNOPSIS**

31958 XSI #include &lt;pthread.h&gt;

31959 int pthread\_attr\_setstack(pthread\_attr\_t \*attr, void \*stackaddr,  
31960 size\_t stacksize);

31961

31962 **DESCRIPTION**31963 Refer to *pthread\_attr\_getstack()*.

31964 **NAME**

31965 pthread\_attr\_setstackaddr — set stackaddr attribute

31966 **SYNOPSIS**

31967 THR TSA #include <pthread.h>

31968 OB int pthread\_attr\_setstackaddr(pthread\_attr\_t \*attr, void \*stackaddr);

31969

31970 **DESCRIPTION**

31971 Refer to *pthread\_attr\_getstackaddr()*.

31972 **NAME**

31973 pthread\_attr\_setstacksize — set stacksize attribute

31974 **SYNOPSIS**

31975 THR TSA #include &lt;pthread.h&gt;

31976 int pthread\_attr\_setstacksize(pthread\_attr\_t \*attr, size\_t stacksize);

31977

31978 **DESCRIPTION**31979 Refer to *pthread\_attr\_getstacksize()*.

## 31980 NAME

31981 pthread\_barrier\_destroy, pthread\_barrier\_init — destroy and initialize a barrier object  
31982 (ADVANCED REALTIME THREADS)

## 31983 SYNOPSIS

```
31984 THR BAR #include <pthread.h>
```

```
31985 int pthread_barrier_destroy(pthread_barrier_t *barrier);  
31986 int pthread_barrier_init(pthread_barrier_t *restrict barrier,  
31987     const pthread_barrierattr_t *restrict attr, unsigned count);  
31988
```

## 31989 DESCRIPTION

31990 The *pthread\_barrier\_destroy()* function shall destroy the barrier referenced by *barrier* and release  
31991 any resources used by the barrier. The effect of subsequent use of the barrier is undefined until  
31992 the barrier is reinitialized by another call to *pthread\_barrier\_init()*. An implementation may use  
31993 this function to set *barrier* to an invalid value. The results are undefined if  
31994 *pthread\_barrier\_destroy()* is called when any thread is blocked on the barrier, or if this function is  
31995 called with an uninitialized barrier.

31996 The *pthread\_barrier\_init()* function shall allocate any resources required to use the barrier  
31997 referenced by *barrier* and shall initialize the barrier with attributes referenced by *attr*. If *attr* is  
31998 NULL, the default barrier attributes shall be used; the effect is the same as passing the address of  
31999 a default barrier attributes object. The results are undefined if *pthread\_barrier\_init()* is called  
32000 when any thread is blocked on the barrier (that is, has not returned from the  
32001 *pthread\_barrier\_wait()* call). The results are undefined if a barrier is used without first being  
32002 initialized. The results are undefined if *pthread\_barrier\_init()* is called specifying an already  
32003 initialized barrier.

32004 The *count* argument specifies the number of threads that must call *pthread\_barrier\_wait()* before  
32005 any of them successfully return from the call. The value specified by *count* must be greater than  
32006 zero.

32007 If the *pthread\_barrier\_init()* function fails, the barrier shall not be initialized and the contents of  
32008 *barrier* are undefined.

32009 Only the object referenced by *barrier* may be used for performing synchronization. The result of  
32010 referring to copies of that object in calls to *pthread\_barrier\_destroy()* or *pthread\_barrier\_wait()* is  
32011 undefined.

## 32012 RETURN VALUE

32013 Upon successful completion, these functions shall return zero; otherwise, an error number shall  
32014 be returned to indicate the error.

## 32015 ERRORS

32016 The *pthread\_barrier\_destroy()* function may fail if:

32017 [EBUSY] The implementation has detected an attempt to destroy a barrier while it is in  
32018 use (for example, while being used in a *pthread\_barrier\_wait()* call) by another  
32019 thread.

32020 [EINVAL] The value specified by *barrier* is invalid.

32021 The *pthread\_barrier\_init()* function shall fail if:

32022 [EAGAIN] The system lacks the necessary resources to initialize another barrier.

32023 [EINVAL] The value specified by *count* is equal to zero.

- 32024 [ENOMEM] Insufficient memory exists to initialize the barrier.
- 32025 The *pthread\_barrier\_init()* function may fail if:
- 32026 [EBUSY] The implementation has detected an attempt to reinitialize a barrier while it is  
32027 in use (for example, while being used in a *pthread\_barrier\_wait()* call) by  
32028 another thread.
- 32029 [EINVAL] The value specified by *attr* is invalid.
- 32030 These functions shall not return an error code of [EINTR].
- 32031 **EXAMPLES**
- 32032 None.
- 32033 **APPLICATION USAGE**
- 32034 The *pthread\_barrier\_destroy()* and *pthread\_barrier\_init()* functions are part of the Barriers option  
32035 and need not be provided on all implementations.
- 32036 **RATIONALE**
- 32037 None.
- 32038 **FUTURE DIRECTIONS**
- 32039 None.
- 32040 **SEE ALSO**
- 32041 *pthread\_barrier\_wait()*, the Base Definitions volume of IEEE Std 1003.1-200x, <pthread.h>
- 32042 **CHANGE HISTORY**
- 32043 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

32044 **NAME**

32045 pthread\_barrier\_init — initialize a barrier object (**ADVANCED REALTIME THREADS**)

32046 **SYNOPSIS**

32047 THR BAR #include <pthread.h>

```
32048 int pthread_barrier_init(pthread_barrier_t *restrict barrier,  
32049                          const pthread_barrierattr_t *restrict attr, unsigned count);  
32050
```

32051 **DESCRIPTION**

32052 Refer to *pthread\_barrier\_destroy()*.

32053 **NAME**

32054 pthread\_barrier\_wait — synchronize at a barrier (ADVANCED REALTIME THREADS)

32055 **SYNOPSIS**

32056 THR BAR #include &lt;pthread.h&gt;

32057 int pthread\_barrier\_wait(pthread\_barrier\_t \*barrier);

32058

32059 **DESCRIPTION**

32060 The *pthread\_barrier\_wait()* function shall synchronize participating threads at the barrier  
 32061 referenced by *barrier*. The calling thread shall block until the required number of threads have  
 32062 called *pthread\_barrier\_wait()* specifying the barrier.

32063 When the required number of threads have called *pthread\_barrier\_wait()* specifying the barrier,  
 32064 the constant PTHREAD\_BARRIER\_SERIAL\_THREAD shall be returned to one unspecified  
 32065 thread and zero shall be returned to each of the remaining threads. At this point, the barrier shall  
 32066 be reset to the state it had as a result of the most recent *pthread\_barrier\_init()* function that  
 32067 referenced it.

32068 The constant PTHREAD\_BARRIER\_SERIAL\_THREAD is defined in <pthread.h> and its value  
 32069 shall be distinct from any other value returned by *pthread\_barrier\_wait()*.

32070 The results are undefined if this function is called with an uninitialized barrier.

32071 If a signal is delivered to a thread blocked on a barrier, upon return from the signal handler the  
 32072 thread shall resume waiting at the barrier if the barrier wait has not completed (that is, if the  
 32073 required number of threads have not arrived at the barrier during the execution of the signal  
 32074 handler); otherwise, the thread shall continue as normal from the completed barrier wait. Until  
 32075 the thread in the signal handler returns from it, it is unspecified whether other threads may  
 32076 proceed past the barrier once they have all reached it.

32077 A thread that has blocked on a barrier shall not prevent any unblocked thread that is eligible to  
 32078 use the same processing resources from eventually making forward progress in its execution.  
 32079 Eligibility for processing resources shall be determined by the scheduling policy.

32080 **RETURN VALUE**

32081 Upon successful completion, the *pthread\_barrier\_wait()* function shall return  
 32082 PTHREAD\_BARRIER\_SERIAL\_THREAD for a single (arbitrary) thread synchronized at the  
 32083 barrier and zero for each of the other threads. Otherwise, an error number shall be returned to  
 32084 indicate the error.

32085 **ERRORS**

32086 The *pthread\_barrier\_wait()* function may fail if:

32087 [EINVAL] The value specified by *barrier* does not refer to an initialized barrier object.

32088 This function shall not return an error code of [EINTR].

32089 **EXAMPLES**

32090 None.

32091 **APPLICATION USAGE**

32092 Applications using this function may be subject to priority inversion, as discussed in the Base  
 32093 Definitions volume of IEEE Std 1003.1-200x, Section 3.285, Priority Inversion.

32094 The *pthread\_barrier\_wait()* function is part of the Barriers option and need not be provided on all  
 32095 implementations.

32096 **RATIONALE**

32097       None.

32098 **FUTURE DIRECTIONS**

32099       None.

32100 **SEE ALSO**

32101       *pthread\_barrier\_destroy()*, *pthread\_barrier\_init()*, the Base Definitions volume of  
32102       IEEE Std 1003.1-200x, <**pthread.h**>

32103 **CHANGE HISTORY**

32104       First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

32105       In the SYNOPSIS, the inclusion of <**sys/types.h**> is no longer required.

32106 **NAME**

32107 pthread\_barrierattr\_destroy, pthread\_barrierattr\_init — destroy and initialize barrier attributes  
 32108 object (**ADVANCED REALTIME THREADS**)

32109 **SYNOPSIS**

```
32110 THR BAR #include <pthread.h>
```

```
32111 int pthread_barrierattr_destroy(pthread_barrierattr_t *attr);
```

```
32112 int pthread_barrierattr_init(pthread_barrierattr_t *attr);
```

32113

32114 **DESCRIPTION**

32115 The *pthread\_barrierattr\_destroy()* function shall destroy a barrier attributes object. A destroyed  
 32116 *attr* attributes object can be reinitialized using *pthread\_barrierattr\_init()*; the results of otherwise  
 32117 referencing the object after it has been destroyed are undefined. An implementation may cause  
 32118 *pthread\_barrierattr\_destroy()* to set the object referenced by *attr* to an invalid value.

32119 The *pthread\_barrierattr\_init()* function shall initialize a barrier attributes object *attr* with the  
 32120 default value for all of the attributes defined by the implementation.

32121 Results are undefined if *pthread\_barrierattr\_init()* is called specifying an already initialized *attr*  
 32122 attributes object.

32123 After a barrier attributes object has been used to initialize one or more barriers, any function  
 32124 affecting the attributes object (including destruction) shall not affect any previously initialized  
 32125 barrier.

32126 **RETURN VALUE**

32127 If successful, the *pthread\_barrierattr\_destroy()* and *pthread\_barrierattr\_init()* functions shall return  
 32128 zero; otherwise, an error number shall be returned to indicate the error.

32129 **ERRORS**

32130 The *pthread\_barrierattr\_destroy()* function may fail if:

32131 [EINVAL] The value specified by *attr* is invalid.

32132 The *pthread\_barrierattr\_init()* function shall fail if:

32133 [ENOMEM] Insufficient memory exists to initialize the barrier attributes object.

32134 These functions shall not return an error code of [EINTR].

32135 **EXAMPLES**

32136 None.

32137 **APPLICATION USAGE**

32138 The *pthread\_barrierattr\_destroy()* and *pthread\_barrierattr\_init()* functions are part of the Barriers  
 32139 option and need not be provided on all implementations.

32140 **RATIONALE**

32141 None.

32142 **FUTURE DIRECTIONS**

32143 None.

32144 **SEE ALSO**

32145 *pthread\_barrierattr\_getshared()*, *pthread\_barrierattr\_setshared()*, the Base Definitions volume of  
 32146 IEEE Std 1003.1-200x, <pthread.h>.

32147 **CHANGE HISTORY**

32148           First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

32149           In the SYNOPSIS, the inclusion of `<sys/types.h>` is no longer required.

32150 **NAME**

32151 pthread\_barrierattr\_getpshared, pthread\_barrierattr\_setpshared — get and set process-shared  
 32152 attribute of barrier attributes object (**ADVANCED REALTIME THREADS**)

32153 **SYNOPSIS**

32154 THR #include <pthread.h>

```
32155 BAR TSH int pthread_barrierattr_getpshared(
32156           const pthread_barrierattr_t *restrict attr,
32157           int *restrict pshared);
32158 int pthread_barrierattr_setpshared(pthread_barrierattr_t *attr,
32159           int pshared);
32160
```

32161 **DESCRIPTION**

32162 The *pthread\_barrierattr\_getpshared()* function shall obtain the value of the process-shared |  
 32163 attribute from the attributes object referenced by *attr*. The *pthread\_barrierattr\_setpshared()* |  
 32164 function shall set the process-shared attribute in an initialized attributes object referenced by |  
 32165 *attr*. |

32166 The process-shared attribute is set to `PTHREAD_PROCESS_SHARED` to permit a barrier to be |  
 32167 operated upon by any thread that has access to the memory where the barrier is allocated. If the |  
 32168 process-shared attribute is `PTHREAD_PROCESS_PRIVATE`, the barrier shall only be operated |  
 32169 upon by threads created within the same process as the thread that initialized the barrier; if |  
 32170 threads of different processes attempt to operate on such a barrier, the behavior is undefined. |  
 32171 The default value of the attribute shall be `PTHREAD_PROCESS_PRIVATE`. Both constants |  
 32172 `PTHREAD_PROCESS_SHARED` and `PTHREAD_PROCESS_PRIVATE` are defined in |  
 32173 `<pthread.h>`.

32174 Additional attributes, their default values, and the names of the associated functions to get and |  
 32175 set those attribute values are implementation-defined.

32176 **RETURN VALUE**

32177 If successful, the *pthread\_barrierattr\_getpshared()* function shall return zero and store the value of  
 32178 the process-shared attribute of *attr* into the object referenced by the *pshared* parameter.  
 32179 Otherwise, an error number shall be returned to indicate the error.

32180 If successful, the *pthread\_barrierattr\_setpshared()* function shall return zero; otherwise, an error  
 32181 number shall be returned to indicate the error.

32182 **ERRORS**

32183 These functions may fail if:

32184 [EINVAL] The value specified by *attr* is invalid.

32185 The *pthread\_barrierattr\_setpshared()* function may fail if:

32186 [EINVAL] The new value specified for the process-shared attribute is not one of the legal  
 32187 values `PTHREAD_PROCESS_SHARED` or `PTHREAD_PROCESS_PRIVATE`.

32188 These functions shall not return an error code of [EINTR].

32189 **EXAMPLES**

32190           None.

32191 **APPLICATION USAGE**

32192           The *pthread\_barrierattr\_getpshared()* and *pthread\_barrierattr\_setpshared()* functions are part of the  
32193           Barriers option and need not be provided on all implementations.

32194 **RATIONALE**

32195           None.

32196 **FUTURE DIRECTIONS**

32197           None.

32198 **SEE ALSO**

32199           *pthread\_barrier\_init()*, *pthread\_barrierattr\_destroy()*, *pthread\_barrierattr\_init()*, the Base Definitions  
32200           volume of IEEE Std 1003.1-200x, <**pthread.h**>

32201 **CHANGE HISTORY**

32202           First released in Issue 6. Derived from IEEE Std 1003.1j-2000

32203 **NAME**

32204 pthread\_barrierattr\_init — initialize barrier attributes object (**ADVANCED REALTIME**  
32205 **THREADS**)

32206 **SYNOPSIS**

32207 THR BAR #include <pthread.h>

32208 int pthread\_barrierattr\_init(pthread\_barrierattr\_t \*attr);

32209

32210 **DESCRIPTION**

32211 Refer to *pthread\_barrierattr\_destroy()*.

32212 **NAME**

32213 pthread\_barrierattr\_setpshared — set process-shared attribute of barrier attributes object  
32214 (**ADVANCED REALTIME THREADS**)

32215 **SYNOPSIS**

```
32216 THR #include <pthread.h>
```

```
32217 BAR TSH int pthread_barrierattr_setpshared(pthread_barrierattr_t *attr,  
32218 int pshared);
```

32219

32220 **DESCRIPTION**

32221 Refer to *pthread\_barrierattr\_getpshared()*.

32222 **NAME**

32223 pthread\_cancel — cancel execution of a thread

32224 **SYNOPSIS**

32225 THR #include &lt;pthread.h&gt;

32226 int pthread\_cancel(pthread\_t thread);

32227

32228 **DESCRIPTION**

32229 The *pthread\_cancel()* function shall request that *thread* be canceled. The target thread's  
 32230 cancelability state and type determines when the cancelation takes effect. When the cancelation  
 32231 is acted on, the cancelation cleanup handlers for *thread* shall be called. When the last cancelation  
 32232 cleanup handler returns, the thread-specific data destructor functions shall be called for *thread*.  
 32233 When the last destructor function returns, *thread* shall be terminated.

32234 The cancelation processing in the target thread shall run asynchronously with respect to the  
 32235 calling thread returning from *pthread\_cancel()*.

32236 **RETURN VALUE**

32237 If successful, the *pthread\_cancel()* function shall return zero; otherwise, an error number shall be  
 32238 returned to indicate the error.

32239 **ERRORS**32240 The *pthread\_cancel()* function may fail if:

32241 [ESRCH] No thread could be found corresponding to that specified by the given thread  
 32242 ID.

32243 The *pthread\_cancel()* function shall not return an error code of [EINTR].32244 **EXAMPLES**

32245 None.

32246 **APPLICATION USAGE**

32247 None.

32248 **RATIONALE**

32249 Two alternative functions were considered to sending the cancelation notification to a thread.  
 32250 One would be to define a new SIGCANCEL signal that had the cancelation semantics when  
 32251 delivered; the other was to define the new *pthread\_cancel()* function, which would trigger the  
 32252 cancelation semantics.

32253 The advantage of a new signal was that so much of the delivery criteria were identical to that  
 32254 used when trying to deliver a signal that making cancelation notification a signal was seen as  
 32255 consistent. Indeed, many implementations implement cancelation using a special signal. On the  
 32256 other hand, there would be no signal functions that could be used with this signal except  
 32257 *pthread\_kill()*, and the behavior of the delivered cancelation signal would be unlike any  
 32258 previously existing defined signal.

32259 The benefits of a special function include the recognition that this signal would be defined  
 32260 because of the similar delivery criteria and that this is the only common behavior between a  
 32261 cancelation request and a signal. In addition, the cancelation delivery mechanism does not have  
 32262 to be implemented as a signal. There are also strong, if not stronger, parallels with language  
 32263 exception mechanisms than with signals that are potentially obscured if the delivery mechanism  
 32264 is visibly closer to signals.

32265 In the end, it was considered that as there were so many exceptions to the use of the new signal  
 32266 with existing signals functions that it would be misleading. A special function has resolved this

32267 problem. This function was carefully defined so that an implementation wishing to provide the  
32268 cancelation functions on top of signals could do so. The special function also means that  
32269 implementations are not obliged to implement cancelation with signals.

## 32270 FUTURE DIRECTIONS

32271 None.

## 32272 SEE ALSO

32273 *pthread\_exit()*, *pthread\_cond\_wait()*, *pthread\_cond\_timedwait()*, *pthread\_join()*,  
32274 *pthread\_setcancelstate()*, the Base Definitions volume of IEEE Std 1003.1-200x, <pthread.h>

## 32275 CHANGE HISTORY

32276 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

## 32277 Issue 6

32278 The *pthread\_cancel()* function is marked as part of the Threads option.

32279 **NAME**

32280 pthread\_cleanup\_pop, pthread\_cleanup\_push — establish cancelation handlers

32281 **SYNOPSIS**

```
32282 THR #include <pthread.h>
32283 void pthread_cleanup_pop(int execute);
32284 void pthread_cleanup_push(void (*routine)(void*), void *arg);
32285
```

32286 **DESCRIPTION**

32287 The *pthread\_cleanup\_pop()* function shall remove the routine at the top of the calling thread's  
 32288 cancelation cleanup stack and optionally invoke it (if *execute* is non-zero).

32289 The *pthread\_cleanup\_push()* function shall push the specified cancelation cleanup handler *routine*  
 32290 onto the calling thread's cancelation cleanup stack. The cancelation cleanup handler shall be  
 32291 popped from the cancelation cleanup stack and invoked with the argument *arg* when:

- 32292 • The thread exits (that is, calls *pthread\_exit()*).
- 32293 • The thread acts upon a cancelation request.
- 32294 • The thread calls *pthread\_cleanup\_pop()* with a non-zero *execute* argument.

32295 These functions may be implemented as macros. The application shall ensure that they appear  
 32296 as statements, and in pairs within the same lexical scope (that is, the *pthread\_cleanup\_push()*  
 32297 macro may be thought to expand to a token list whose first token is '{' with  
 32298 *pthread\_cleanup\_pop()* expanding to a token list whose last token is the corresponding '}').

32299 The effect of calling *longjmp()* or *siglongjmp()* is undefined if there have been any calls to  
 32300 *pthread\_cleanup\_push()* or *pthread\_cleanup\_pop()* made without the matching call since the jump  
 32301 buffer was filled. The effect of calling *longjmp()* or *siglongjmp()* from inside a cancelation  
 32302 cleanup handler is also undefined unless the jump buffer was also filled in the cancelation  
 32303 cleanup handler.

32304 **RETURN VALUE**32305 The *pthread\_cleanup\_push()* and *pthread\_cleanup\_pop()* functions shall not return a value.32306 **ERRORS**

32307 No errors are defined.

32308 These functions shall not return an error code of [EINTR].

32309 **EXAMPLES**

32310 The following is an example using thread primitives to implement a cancelable, writers-priority  
 32311 read-write lock:

```
32312 typedef struct {
32313     pthread_mutex_t lock;
32314     pthread_cond_t rcond,
32315     wcond;
32316     int lock_count; /* < 0 .. Held by writer. */
32317                   /* > 0 .. Held by lock_count readers. */
32318                   /* = 0 .. Held by nobody. */
32319     int waiting_writers; /* Count of waiting writers. */
32320 } rwlock;
32321 void
32322 waiting_reader_cleanup(void *arg)
32323 {
```

```
32324         rwlock *l;
32325         l = (rwlock *) arg;
32326         pthread_mutex_unlock(&l->lock);
32327     }
32328 void
32329 lock_for_read(rwlock *l)
32330 {
32331     pthread_mutex_lock(&l->lock);
32332     pthread_cleanup_push(waiting_reader_cleanup, l);
32333     while ((l->lock_count < 0) && (l->waiting_writers != 0))
32334         pthread_cond_wait(&l->rcond, &l->lock);
32335     l->lock_count++;
32336     /*
32337      * Note the pthread_cleanup_pop executes
32338      * waiting_reader_cleanup.
32339      */
32340     pthread_cleanup_pop(1);
32341 }
32342 void
32343 release_read_lock(rwlock *l)
32344 {
32345     pthread_mutex_lock(&l->lock);
32346     if (--l->lock_count == 0)
32347         pthread_cond_signal(&l->wcond);
32348     pthread_mutex_unlock(l);
32349 }
32350 void
32351 waiting_writer_cleanup(void *arg)
32352 {
32353     rwlock *l;
32354     l = (rwlock *) arg;
32355     if ((--l->waiting_writers == 0) && (l->lock_count >= 0)) {
32356         /*
32357          * This only happens if we have been canceled.
32358          */
32359         pthread_cond_broadcast(&l->wcond);
32360     }
32361     pthread_mutex_unlock(&l->lock);
32362 }
32363 void
32364 lock_for_write(rwlock *l)
32365 {
32366     pthread_mutex_lock(&l->lock);
32367     l->waiting_writers++;
32368     pthread_cleanup_push(waiting_writer_cleanup, l);
32369     while (l->lock_count != 0)
32370         pthread_cond_wait(&l->wcond, &l->lock);
32371     l->lock_count = -1;
32372     /*
```

```
32373         * Note the pthread_cleanup_pop executes
32374         * waiting_writer_cleanup.
32375         */
32376         pthread_cleanup_pop(1);
32377     }
32378
32378 void
32379 release_write_lock(rwlock *l)
32380 {
32381     pthread_mutex_lock(&l->lock);
32382     l->lock_count = 0;
32383     if (l->waiting_writers == 0)
32384         pthread_cond_broadcast(&l->rcond)
32385     else
32386         pthread_cond_signal(&l->wcond);
32387     pthread_mutex_unlock(&l->lock);
32388 }
32389
32389 /*
32390  * This function is called to initialize the read/write lock.
32391  */
32392 void
32393 initialize_rwlock(rwlock *l)
32394 {
32395     pthread_mutex_init(&l->lock, pthread_mutexattr_default);
32396     pthread_cond_init(&l->wcond, pthread_condattr_default);
32397     pthread_cond_init(&l->rcond, pthread_condattr_default);
32398     l->lock_count = 0;
32399     l->waiting_writers = 0;
32400 }
32401
32401 reader_thread()
32402 {
32403     lock_for_read(&lock);
32404     pthread_cleanup_push(release_read_lock, &lock);
32405     /*
32406      * Thread has read lock.
32407      */
32408     pthread_cleanup_pop(1);
32409 }
32410
32410 writer_thread()
32411 {
32412     lock_for_write(&lock);
32413     pthread_cleanup_push(release_write_lock, &lock);
32414     /*
32415      * Thread has write lock.
32416      */
32417     pthread_cleanup_pop(1);
32418 }
```

32419 **APPLICATION USAGE**

32420 The two routines that push and pop cancelation cleanup handlers, *pthread\_cleanup\_push()* and  
32421 *pthread\_cleanup\_pop()*, can be thought of as left and right parentheses. They always need to be  
32422 matched.

32423 **RATIONALE**

32424 The restriction that the two routines that push and pop cancelation cleanup handlers,  
32425 *pthread\_cleanup\_push()* and *pthread\_cleanup\_pop()*, have to appear in the same lexical scope  
32426 allows for efficient macro or compiler implementations and efficient storage management. A  
32427 sample implementation of these routines as macros might look like this:

```
32428 #define pthread_cleanup_push(rtn,arg) { \  
32429     struct _pthread_handler_rec __cleanup_handler, **__head; \  
32430     __cleanup_handler.rtn = rtn; \  
32431     __cleanup_handler.arg = arg; \  
32432     (void) pthread_getspecific(_pthread_handler_key, &__head); \  
32433     __cleanup_handler.next = *__head; \  
32434     *__head = &__cleanup_handler;  
  
32435 #define pthread_cleanup_pop(ex) \  
32436     *__head = __cleanup_handler.next; \  
32437     if (ex) (*__cleanup_handler.rtn)(__cleanup_handler.arg); \  
32438 }
```

32439 A more ambitious implementation of these routines might do even better by allowing the  
32440 compiler to note that the cancelation cleanup handler is a constant and can be expanded inline.

32441 This volume of IEEE Std 1003.1-200x currently leaves unspecified the effect of calling *longjmp()*  
32442 from a signal handler executing in a POSIX System Interfaces function. If an implementation  
32443 wants to allow this and give the programmer reasonable behavior, the *longjmp()* function has to  
32444 call all cancelation cleanup handlers that have been pushed but not popped since the time  
32445 *setjmp()* was called.

32446 Consider a multi-threaded function called by a thread that uses signals. If a signal were  
32447 delivered to a signal handler during the operation of *qsort()* and that handler were to call  
32448 *longjmp()* (which, in turn, did *not* call the cancelation cleanup handlers) the helper threads  
32449 created by the *qsort()* function would not be canceled. Instead, they would continue to execute  
32450 and write into the argument array even though the array might have been popped off of the  
32451 stack.

32452 Note that the specified cleanup handling mechanism is especially tied to the C language and,  
32453 while the requirement for a uniform mechanism for expressing cleanup is language-  
32454 independent, the mechanism used in other languages may be quite different. In addition, this  
32455 mechanism is really only necessary due to the lack of a real exception mechanism in the C  
32456 language, which would be the ideal solution.

32457 There is no notion of a cancelation cleanup-safe function. If an application has no cancelation  
32458 points in its signal handlers, blocks any signal whose handler may have cancelation points while  
32459 calling async-unsafe functions, or disables cancelation while calling async-unsafe functions, all  
32460 functions may be safely called from cancelation cleanup routines.

32461 **FUTURE DIRECTIONS**

32462 None.

32463 **SEE ALSO**

32464 *pthread\_cancel()*, *pthread\_setcancelstate()*, the Base Definitions volume of IEEE Std 1003.1-200x,  
32465 <pthread.h>

32466 **CHANGE HISTORY**

32467 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

32468 **Issue 6**

32469 The *pthread\_cleanup\_pop()* and *pthread\_cleanup\_push()* functions are marked as part of the  
32470 Threads option.

32471 The APPLICATION USAGE section is added.

32472 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

32473 **NAME**

32474 pthread\_cond\_broadcast, pthread\_cond\_signal — broadcast or signal a condition

32475 **SYNOPSIS**

32476 THR #include &lt;pthread.h&gt;

32477 int pthread\_cond\_broadcast(pthread\_cond\_t \*cond);

32478 int pthread\_cond\_signal(pthread\_cond\_t \*cond);

32479

32480 **DESCRIPTION**

32481 These functions shall unblock threads blocked on a condition variable. |

32482 The *pthread\_cond\_broadcast()* function shall unblock all threads currently blocked on the  
32483 specified condition variable *cond*.32484 The *pthread\_cond\_signal()* function shall unblock at least one of the threads that are blocked on  
32485 the specified condition variable *cond* (if any threads are blocked on *cond*).32486 If more than one thread is blocked on a condition variable, the scheduling policy shall determine |  
32487 the order in which threads are unblocked. When each thread unblocked as a result of a |  
32488 *pthread\_cond\_broadcast()* or *pthread\_cond\_signal()* returns from its call to *pthread\_cond\_wait()* or |  
32489 *pthread\_cond\_timedwait()*, the thread shall own the mutex with which it called |  
32490 *pthread\_cond\_wait()* or *pthread\_cond\_timedwait()*. The thread(s) that are unblocked shall contend |  
32491 for the mutex according to the scheduling policy (if applicable), and as if each had called |  
32492 *pthread\_mutex\_lock()*.32493 The *pthread\_cond\_broadcast()* or *pthread\_cond\_signal()* functions may be called by a thread  
32494 whether or not it currently owns the mutex that threads calling *pthread\_cond\_wait()* or  
32495 *pthread\_cond\_timedwait()* have associated with the condition variable during their waits;  
32496 however, if predictable scheduling behavior is required, then that mutex shall be locked by the  
32497 thread calling *pthread\_cond\_broadcast()* or *pthread\_cond\_signal()*.32498 The *pthread\_cond\_broadcast()* and *pthread\_cond\_signal()* functions shall have no effect if there are |  
32499 no threads currently blocked on *cond*. |32500 **RETURN VALUE**32501 If successful, the *pthread\_cond\_broadcast()* and *pthread\_cond\_signal()* functions shall return zero;  
32502 otherwise, an error number shall be returned to indicate the error.32503 **ERRORS**32504 The *pthread\_cond\_broadcast()* and *pthread\_cond\_signal()* function may fail if:32505 [EINVAL] The value *cond* does not refer to an initialized condition variable.

32506 These functions shall not return an error code of [EINTR].

32507 **EXAMPLES**

32508 None.

32509 **APPLICATION USAGE**32510 The *pthread\_cond\_broadcast()* function is used whenever the shared-variable state has been  
32511 changed in a way that more than one thread can proceed with its task. Consider a single  
32512 producer/multiple consumer problem, where the producer can insert multiple items on a list  
32513 that is accessed one item at a time by the consumers. By calling the *pthread\_cond\_broadcast()*  
32514 function, the producer would notify all consumers that might be waiting, and thereby the  
32515 application would receive more throughput on a multi-processor. In addition,  
32516 *pthread\_cond\_broadcast()* makes it easier to implement a read-write lock. The  
32517 *pthread\_cond\_broadcast()* function is needed in order to wake up all waiting readers when a

32518 writer releases its lock. Finally, the two-phase commit algorithm can use this broadcast function  
32519 to notify all clients of an impending transaction commit.

32520 It is not safe to use the *pthread\_cond\_signal()* function in a signal handler that is invoked  
32521 asynchronously. Even if it were safe, there would still be a race between the test of the Boolean  
32522 *pthread\_cond\_wait()* that could not be efficiently eliminated.

32523 Mutexes and condition variables are thus not suitable for releasing a waiting thread by signaling  
32524 from code running in a signal handler.

### 32525 RATIONALE

#### 32526 Multiple Awakenings by Condition Signal

32527 On a multi-processor, it may be impossible for an implementation of *pthread\_cond\_signal()* to  
32528 avoid the unblocking of more than one thread blocked on a condition variable. For example,  
32529 consider the following partial implementation of *pthread\_cond\_wait()* and *pthread\_cond\_signal()*,  
32530 executed by two threads in the order given. One thread is trying to wait on the condition  
32531 variable, another is concurrently executing *pthread\_cond\_signal()*, while a third thread is already  
32532 waiting.

```
32533 pthread_cond_wait(mutex, cond):
32534     value = cond->value; /* 1 */
32535     pthread_mutex_unlock(mutex); /* 2 */
32536     pthread_mutex_lock(cond->mutex); /* 10 */
32537     if (value == cond->value) { /* 11 */
32538         me->next_cond = cond->waiter;
32539         cond->waiter = me;
32540         pthread_mutex_unlock(cond->mutex);
32541         unable_to_run(me);
32542     } else
32543         pthread_mutex_unlock(cond->mutex); /* 12 */
32544     pthread_mutex_lock(mutex); /* 13 */

32545 pthread_cond_signal(cond):
32546     pthread_mutex_lock(cond->mutex); /* 3 */
32547     cond->value++; /* 4 */
32548     if (cond->waiter) { /* 5 */
32549         sleeper = cond->waiter; /* 6 */
32550         cond->waiter = sleeper->next_cond; /* 7 */
32551         able_to_run(sleeper); /* 8 */
32552     }
32553     pthread_mutex_unlock(cond->mutex); /* 9 */
```

32554 The effect is that more than one thread can return from its call to *pthread\_cond\_wait()* or  
32555 *pthread\_cond\_timedwait()* as a result of one call to *pthread\_cond\_signal()*. This effect is called  
32556 “spurious wakeup”. Note that the situation is self-correcting in that the number of threads that  
32557 are so awakened is finite; for example, the next thread to call *pthread\_cond\_wait()* after the  
32558 sequence of events above blocks.

32559 While this problem could be resolved, the loss of efficiency for a fringe condition that occurs  
32560 only rarely is unacceptable, especially given that one has to check the predicate associated with a  
32561 condition variable anyway. Correcting this problem would unnecessarily reduce the degree of  
32562 concurrency in this basic building block for all higher-level synchronization operations.

32563 An added benefit of allowing spurious wakeups is that applications are forced to code a  
32564 predicate-testing-loop around the condition wait. This also makes the application tolerate

32565           superfluous condition broadcasts or signals on the same condition variable that may be coded in  
32566           some other part of the application. The resulting applications are thus more robust. Therefore,  
32567           IEEE Std 1003.1-200x explicitly documents that spurious wakeups may occur.

## 32568 **FUTURE DIRECTIONS**

32569           None.

## 32570 **SEE ALSO**

32571           *pthread\_cond\_destroy()*, *pthread\_cond\_timedwait()*, *pthread\_cond\_wait()*, the Base Definitions  
32572           volume of IEEE Std 1003.1-200x, <**pthread.h**>

## 32573 **CHANGE HISTORY**

32574           First released in Issue 5. Included for alignment with the POSIX Threads Extension.

## 32575 **Issue 6**

32576           The *pthread\_cond\_broadcast()* and *pthread\_cond\_signal()* functions are marked as part of the  
32577           Threads option.

32578           The APPLICATION USAGE section is added.

32579 **NAME**

32580 pthread\_cond\_destroy, pthread\_cond\_init — destroy and initialize condition variables

32581 **SYNOPSIS**

32582 THR #include &lt;pthread.h&gt;

```

32583 int pthread_cond_destroy(pthread_cond_t *cond);
32584 int pthread_cond_init(pthread_cond_t *restrict cond,
32585     const pthread_condattr_t *restrict attr);
32586 pthread_cond_t cond = PTHREAD_COND_INITIALIZER;
32587

```

32588 **DESCRIPTION**

32589 The *pthread\_cond\_destroy()* function shall destroy the given condition variable specified by *cond*;  
 32590 the object becomes, in effect, uninitialized. An implementation may cause *pthread\_cond\_destroy()*  
 32591 to set the object referenced by *cond* to an invalid value. A destroyed condition variable object can  
 32592 be reinitialized using *pthread\_cond\_init()*; the results of otherwise referencing the object after it  
 32593 has been destroyed are undefined.

32594 It shall be safe to destroy an initialized condition variable upon which no threads are currently  
 32595 blocked. Attempting to destroy a condition variable upon which other threads are currently  
 32596 blocked results in undefined behavior.

32597 The *pthread\_cond\_init()* function shall initialize the condition variable referenced by *cond* with  
 32598 attributes referenced by *attr*. If *attr* is NULL, the default condition variable attributes shall be  
 32599 used; the effect is the same as passing the address of a default condition variable attributes  
 32600 object. Upon successful initialization, the state of the condition variable shall become initialized.

32601 Only *cond* itself may be used for performing synchronization. The result of referring to copies of  
 32602 *cond* in calls to *pthread\_cond\_wait()*, *pthread\_cond\_timedwait()*, *pthread\_cond\_signal()*,  
 32603 *pthread\_cond\_broadcast()*, and *pthread\_cond\_destroy()* is undefined.

32604 Attempting to initialize an already initialized condition variable results in undefined behavior.

32605 In cases where default condition variable attributes are appropriate, the macro  
 32606 PTHREAD\_COND\_INITIALIZER can be used to initialize condition variables that are statically  
 32607 allocated. The effect shall be equivalent to dynamic initialization by a call to *pthread\_cond\_init()*  
 32608 with parameter *attr* specified as NULL, except that no error checks are performed.

32609 **RETURN VALUE**

32610 If successful, the *pthread\_cond\_destroy()* and *pthread\_cond\_init()* functions shall return zero;  
 32611 otherwise, an error number shall be returned to indicate the error.

32612 The [EBUSY] and [EINVAL] error checks, if implemented, shall act as if they were performed  
 32613 immediately at the beginning of processing for the function and caused an error return prior to  
 32614 modifying the state of the condition variable specified by *cond*.

32615 **ERRORS**

32616 The *pthread\_cond\_destroy()* function may fail if:

32617 [EBUSY] The implementation has detected an attempt to destroy the object referenced  
 32618 by *cond* while it is referenced (for example, while being used in a  
 32619 *pthread\_cond\_wait()* or *pthread\_cond\_timedwait()*) by another thread.

32620 [EINVAL] The value specified by *cond* is invalid.

32621 The *pthread\_cond\_init()* function shall fail if:

32622 [EAGAIN] The system lacked the necessary resources (other than memory) to initialize  
 32623 another condition variable.

32624 [ENOMEM] Insufficient memory exists to initialize the condition variable.

32625 The `pthread_cond_init()` function may fail if:

32626 [EBUSY] The implementation has detected an attempt to reinitialize the object |  
32627 referenced by `cond`, a previously initialized, but not yet destroyed, condition  
32628 variable.

32629 [EINVAL] The value specified by `attr` is invalid.

32630 These functions shall not return an error code of [EINTR].

### 32631 EXAMPLES

32632 A condition variable can be destroyed immediately after all the threads that are blocked on it are  
32633 awakened. For example, consider the following code:

```
32634 struct list {
32635     pthread_mutex_t lm;
32636     ...
32637 }
32638 struct elt {
32639     key k;
32640     int busy;
32641     pthread_cond_t notbusy;
32642     ...
32643 }
32644 /* Find a list element and reserve it. */
32645 struct elt *
32646 list_find(struct list *lp, key k)
32647 {
32648     struct elt *ep;
32649     pthread_mutex_lock(&lp->lm);
32650     while ((ep = find_elt(l, k) != NULL) && ep->busy)
32651         pthread_cond_wait(&ep->notbusy, &lp->lm);
32652     if (ep != NULL)
32653         ep->busy = 1;
32654     pthread_mutex_unlock(&lp->lm);
32655     return(ep);
32656 }
32657 delete_elt(struct list *lp, struct elt *ep)
32658 {
32659     pthread_mutex_lock(&lp->lm);
32660     assert(ep->busy);
32661     ... remove ep from list ...
32662     ep->busy = 0; /* Paranoid. */
32663     (A) pthread_cond_broadcast(&ep->notbusy);
32664     pthread_mutex_unlock(&lp->lm);
32665     (B) pthread_cond_destroy(&rp->notbusy);
32666     free(ep);
32667 }
```

32668 In this example, the condition variable and its list element may be freed (line B) immediately  
32669 after all threads waiting for it are awakened (line A), since the mutex and the code ensure that no  
32670 other thread can touch the element to be deleted.

32671 **APPLICATION USAGE**

32672 None.

32673 **RATIONALE**32674 See *pthread\_mutex\_init()*; a similar rationale applies to condition variables.32675 **FUTURE DIRECTIONS**

32676 None.

32677 **SEE ALSO**32678 *pthread\_cond\_broadcast()*, *pthread\_cond\_signal()*, *pthread\_cond\_timedwait()*, *pthread\_cond\_wait()*,  
32679 the Base Definitions volume of IEEE Std 1003.1-200x, <pthread.h>32680 **CHANGE HISTORY**

32681 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

32682 **Issue 6**32683 The *pthread\_cond\_destroy()* and *pthread\_cond\_init()* functions are marked as part of the Threads  
32684 option.

32685 IEEE PASC Interpretation 1003.1c #34 is applied, updating the DESCRIPTION.

32686 The **restrict** keyword is added to the *pthread\_cond\_init()* prototype for alignment with the  
32687 ISO/IEC 9899:1999 standard.

32688 **NAME**

32689 pthread\_cond\_init — initialize condition variables

32690 **SYNOPSIS**

32691 THR #include <pthread.h>

32692 int pthread\_cond\_init(pthread\_cond\_t \*restrict cond,

32693 const pthread\_condattr\_t \*restrict attr);

32694 pthread\_cond\_t cond = PTHREAD\_COND\_INITIALIZER;

32695

32696 **DESCRIPTION**

32697 Refer to *pthread\_cond\_destroy()*.

32698 **NAME**

32699 pthread\_cond\_signal — signal a condition

32700 **SYNOPSIS**

32701 THR #include &lt;pthread.h&gt;

32702 int pthread\_cond\_signal(pthread\_cond\_t \*cond);

32703

32704 **DESCRIPTION**32705 Refer to *pthread\_cond\_broadcast()*.

## 32706 NAME

32707 pthread\_cond\_timedwait, pthread\_cond\_wait — wait on a condition

## 32708 SYNOPSIS

32709 THR #include &lt;pthread.h&gt;

```

32710 int pthread_cond_timedwait(pthread_cond_t *restrict cond,
32711 pthread_mutex_t *restrict mutex,
32712 const struct timespec *restrict abstime);
32713 int pthread_cond_wait(pthread_cond_t *restrict cond,
32714 pthread_mutex_t *restrict mutex);
32715

```

## 32716 DESCRIPTION

32717 The *pthread\_cond\_timedwait()* and *pthread\_cond\_wait()* functions shall block on a condition |  
32718 variable. They shall be called with *mutex* locked by the calling thread or undefined behavior |  
32719 results.

32720 These functions atomically release *mutex* and cause the calling thread to block on the condition  
32721 variable *cond*; atomically here means “atomically with respect to access by another thread to the  
32722 mutex and then the condition variable”. That is, if another thread is able to acquire the mutex  
32723 after the about-to-block thread has released it, then a subsequent call to *pthread\_cond\_broadcast()*  
32724 or *pthread\_cond\_signal()* in that thread shall behave as if it were issued after the about-to-block  
32725 thread has blocked.

32726 Upon successful return, the mutex shall have been locked and shall be owned by the calling |  
32727 thread. |

32728 When using condition variables there is always a Boolean predicate involving shared variables  
32729 associated with each condition wait that is true if the thread should proceed. Spurious wakeups  
32730 from the *pthread\_cond\_timedwait()* or *pthread\_cond\_wait()* functions may occur. Since the return  
32731 from *pthread\_cond\_timedwait()* or *pthread\_cond\_wait()* does not imply anything about the value  
32732 of this predicate, the predicate should be re-evaluated upon such return.

32733 The effect of using more than one mutex for concurrent *pthread\_cond\_timedwait()* or  
32734 *pthread\_cond\_wait()* operations on the same condition variable is undefined; that is, a condition  
32735 variable becomes bound to a unique mutex when a thread waits on the condition variable, and  
32736 this (dynamic) binding shall end when the wait returns.

32737 A condition wait (whether timed or not) is a cancelation point. When the cancelability enable  
32738 state of a thread is set to PTHREAD\_CANCEL\_DEFERRED, a side effect of acting upon a  
32739 cancelation request while in a condition wait is that the mutex is (in effect) re-acquired before  
32740 calling the first cancelation cleanup handler. The effect is as if the thread were unblocked,  
32741 allowed to execute up to the point of returning from the call to *pthread\_cond\_timedwait()* or  
32742 *pthread\_cond\_wait()*, but at that point notices the cancelation request and instead of returning to  
32743 the caller of *pthread\_cond\_timedwait()* or *pthread\_cond\_wait()*, starts the thread cancelation  
32744 activities, which includes calling cancelation cleanup handlers.

32745 A thread that has been unblocked because it has been canceled while blocked in a call to  
32746 *pthread\_cond\_timedwait()* or *pthread\_cond\_wait()* shall not consume any condition signal that  
32747 may be directed concurrently at the condition variable if there are other threads blocked on the  
32748 condition variable.

32749 The *pthread\_cond\_timedwait()* function shall be equivalent to *pthread\_cond\_wait()*, except that an |  
32750 error is returned if the absolute time specified by *abstime* passes (that is, system time equals or  
32751 exceeds *abstime*) before the condition *cond* is signaled or broadcasted, or if the absolute time

32752 CS

32753 specified by *abstime* has already been passed at the time of the call. If the Clock Selection option  
 32754 is supported, the condition variable shall have a clock attribute which specifies the clock that  
 32755 shall be used to measure the time specified by the *abstime* argument. When such timeouts occur,  
 32756 *pthread\_cond\_timedwait()* shall nonetheless release and re-acquire the mutex referenced by *mutex*.  
 32757 The *pthread\_cond\_timedwait()* function is also a cancellation point.

32758 If a signal is delivered to a thread waiting for a condition variable, upon return from the signal  
 32759 handler the thread resumes waiting for the condition variable as if it was not interrupted, or it  
 32760 shall return zero due to spurious wakeup.

#### 32761 RETURN VALUE

32762 Except in the case of [ETIMEDOUT], all these error checks shall act as if they were performed  
 32763 immediately at the beginning of processing for the function and shall cause an error return, in  
 32764 effect, prior to modifying the state of the mutex specified by *mutex* or the condition variable  
 32765 specified by *cond*.

32766 Upon successful completion, a value of zero shall be returned; otherwise, an error number shall  
 32767 be returned to indicate the error.

#### 32768 ERRORS

32769 The *pthread\_cond\_timedwait()* function shall fail if:

32770 [ETIMEDOUT] The time specified by *abstime* to *pthread\_cond\_timedwait()* has passed.

32771 The *pthread\_cond\_timedwait()* and *pthread\_cond\_wait()* functions may fail if:

32772 [EINVAL] The value specified by *cond*, *mutex*, or *abstime* is invalid.

32773 [EINVAL] Different mutexes were supplied for concurrent *pthread\_cond\_timedwait()* or  
 32774 *pthread\_cond\_wait()* operations on the same condition variable.

32775 [EPERM] The mutex was not owned by the current thread at the time of the call.

32776 These functions shall not return an error code of [EINTR].

#### 32777 EXAMPLES

32778 None.

#### 32779 APPLICATION USAGE

32780 None.

#### 32781 RATIONALE

##### 32782 Condition Wait Semantics

32783 It is important to note that when *pthread\_cond\_wait()* and *pthread\_cond\_timedwait()* return  
 32784 without error, the associated predicate may still be false. Similarly, when  
 32785 *pthread\_cond\_timedwait()* returns with the timeout error, the associated predicate may be true  
 32786 due to an unavoidable race between the expiration of the timeout and the predicate state change.

32787 Some implementations, particularly on a multi-processor, may sometimes cause multiple  
 32788 threads to wake up when the condition variable is signaled simultaneously on different  
 32789 processors.

32790 In general, whenever a condition wait returns, the thread has to re-evaluate the predicate  
 32791 associated with the condition wait to determine whether it can safely proceed, should wait  
 32792 again, or should declare a timeout. A return from the wait does not imply that the associated  
 32793 predicate is either true or false.

32794 It is thus recommended that a condition wait be enclosed in the equivalent of a “while loop”  
 32795 that checks the predicate.

32796 **Timed Wait Semantics**

32797 An absolute time measure was chosen for specifying the timeout parameter for two reasons.  
32798 First, a relative time measure can be easily implemented on top of a function that specifies  
32799 absolute time, but there is a race condition associated with specifying an absolute timeout on top  
32800 of a function that specifies relative timeouts. For example, assume that `clock_gettime()` returns  
32801 the current time and `cond_relative_timed_wait()` uses relative timeouts:

```
32802 clock_gettime(CLOCK_REALTIME, &now)  
32803 reltime = sleep_til_this_absolute_time -now;  
32804 cond_relative_timed_wait(c, m, &reltime);
```

32805 If the thread is preempted between the first statement and the last statement, the thread blocks  
32806 for too long. Blocking, however, is irrelevant if an absolute timeout is used. An absolute timeout  
32807 also need not be recomputed if it is used multiple times in a loop, such as that enclosing a  
32808 condition wait.

32809 For cases when the system clock is advanced discontinuously by an operator, it is expected that  
32810 implementations process any timed wait expiring at an intervening time as if that time had  
32811 actually occurred.

32812 **Cancelation and Condition Wait**

32813 A condition wait, whether timed or not, is a cancelation point. That is, the functions  
32814 `pthread_cond_wait()` or `pthread_cond_timedwait()` are points where a pending (or concurrent)  
32815 cancelation request is noticed. The reason for this is that an indefinite wait is possible at these  
32816 points—whatever event is being waited for, even if the program is totally correct, might never  
32817 occur; for example, some input data being awaited might never be sent. By making condition  
32818 wait a cancelation point, the thread can be canceled and perform its cancelation cleanup handler  
32819 even though it may be stuck in some indefinite wait.

32820 A side effect of acting on a cancelation request while a thread is blocked on a condition variable  
32821 is to re-acquire the mutex before calling any of the cancelation cleanup handlers. This is done in  
32822 order to ensure that the cancelation cleanup handler is executed in the same state as the critical  
32823 code that lies both before and after the call to the condition wait function. This rule is also  
32824 required when interfacing to POSIX threads from languages, such as Ada or C++, which may  
32825 choose to map cancelation onto a language exception; this rule ensures that each exception  
32826 handler guarding a critical section can always safely depend upon the fact that the associated  
32827 mutex has already been locked regardless of exactly where within the critical section the  
32828 exception was raised. Without this rule, there would not be a uniform rule that exception  
32829 handlers could follow regarding the lock, and so coding would become very cumbersome.

32830 Therefore, since *some* statement has to be made regarding the state of the lock when a  
32831 cancelation is delivered during a wait, a definition has been chosen that makes application  
32832 coding most convenient and error free.

32833 When acting on a cancelation request while a thread is blocked on a condition variable, the  
32834 implementation is required to ensure that the thread does not consume any condition signals  
32835 directed at that condition variable if there are any other threads waiting on that condition  
32836 variable. This rule is specified in order to avoid deadlock conditions that could occur if these two  
32837 independent requests (one acting on a thread and the other acting on the condition variable)  
32838 were not processed independently.

32839 **Performance of Mutexes and Condition Variables**

32840 Mutexes are expected to be locked only for a few instructions. This practice is almost  
 32841 automatically enforced by the desire of programmers to avoid long serial regions of execution  
 32842 (which would reduce total effective parallelism).

32843 When using mutexes and condition variables, one tries to ensure that the usual case is to lock the  
 32844 mutex, access shared data, and unlock the mutex. Waiting on a condition variable should be a  
 32845 relatively rare situation. For example, when implementing a read-write lock, code that acquires a  
 32846 read-lock typically needs only to increment the count of readers (under mutual-exclusion) and  
 32847 return. The calling thread would actually wait on the condition variable only when there is  
 32848 already an active writer. So the efficiency of a synchronization operation is bounded by the cost  
 32849 of mutex lock/unlock and not by condition wait. Note that in the usual case there is no context  
 32850 switch.

32851 This is not to say that the efficiency of condition waiting is unimportant. Since there needs to be  
 32852 at least one context switch per Ada rendezvous, the efficiency of waiting on a condition variable  
 32853 is important. The cost of waiting on a condition variable should be little more than the minimal  
 32854 cost for a context switch plus the time to unlock and lock the mutex.

32855 **Features of Mutexes and Condition Variables**

32856 It had been suggested that the mutex acquisition and release be decoupled from condition wait.  
 32857 This was rejected because it is the combined nature of the operation that, in fact, facilitates  
 32858 realtime implementations. Those implementations can atomically move a high-priority thread  
 32859 between the condition variable and the mutex in a manner that is transparent to the caller. This  
 32860 can prevent extra context switches and provide more deterministic acquisition of a mutex when  
 32861 the waiting thread is signaled. Thus, fairness and priority issues can be dealt with directly by the  
 32862 scheduling discipline. Furthermore, the current condition wait operation matches existing  
 32863 practice.

32864 **Scheduling Behavior of Mutexes and Condition Variables**

32865 Synchronization primitives that attempt to interfere with scheduling policy by specifying an  
 32866 ordering rule are considered undesirable. Threads waiting on mutexes and condition variables  
 32867 are selected to proceed in an order dependent upon the scheduling policy rather than in some  
 32868 fixed order (for example, FIFO or priority). Thus, the scheduling policy determines which  
 32869 thread(s) are awakened and allowed to proceed.

32870 **Timed Condition Wait**

32871 The *pthread\_cond\_timedwait()* function allows an application to give up waiting for a particular  
 32872 condition after a given amount of time. An example of its use follows:

```
32873 (void) pthread_mutex_lock(&t.mn);
32874     t.waiters++;
32875     clock_gettime(CLOCK_REALTIME, &ts);
32876     ts.tv_sec += 5;
32877     rc = 0;
32878     while (! mypredicate(&t) && rc == 0)
32879         rc = pthread_cond_timedwait(&t.cond, &t.mn, &ts);
32880     t.waiters--;
32881     if (rc == 0) setmystate(&t);
32882 (void) pthread_mutex_unlock(&t.mn);
```

32883 By making the timeout parameter absolute, it does not need to be recomputed each time the  
32884 program checks its blocking predicate. If the timeout was relative, it would have to be  
32885 recomputed before each call. This would be especially difficult since such code would need to  
32886 take into account the possibility of extra wakeups that result from extra broadcasts or signals on  
32887 the condition variable that occur before either the predicate is true or the timeout is due.

## 32888 FUTURE DIRECTIONS

32889 None.

## 32890 SEE ALSO

32891 *pthread\_cond\_signal()*, *pthread\_cond\_broadcast()*, the Base Definitions volume of  
32892 IEEE Std 1003.1-200x, <pthread.h>

## 32893 CHANGE HISTORY

32894 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

## 32895 Issue 6

32896 The *pthread\_cond\_timedwait()* and *pthread\_cond\_wait()* functions are marked as part of the  
32897 Threads option.

32898 The Open Group Corrigendum U021/9 is applied, correcting the prototype for the  
32899 *pthread\_cond\_wait()* function.

32900 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by adding semantics for  
32901 the Clock Selection option.

32902 The ERRORS section has an additional case for [EPERM] in response to IEEE PASC  
32903 Interpretation 1003.1c #28.

32904 The **restrict** keyword is added to the *pthread\_cond\_timedwait()* and *pthread\_cond\_wait()*  
32905 prototypes for alignment with the ISO/IEC 9899:1999 standard.

32906 **NAME**

32907 pthread\_cond\_wait — wait on a condition

32908 **SYNOPSIS**

32909 THR #include &lt;pthread.h&gt;

32910 int pthread\_cond\_wait(pthread\_cond\_t \*restrict cond,  
32911 pthread\_mutex\_t \*restrict mutex);

32912

32913 **DESCRIPTION**32914 Refer to *pthread\_cond\_timedwait()*.

32915 **NAME**

32916 pthread\_condattr\_destroy, pthread\_condattr\_init — destroy and initialize condition variable  
 32917 attributes object

32918 **SYNOPSIS**

```
32919 THR #include <pthread.h>
```

```
32920 int pthread_condattr_destroy(pthread_condattr_t *attr);
```

```
32921 int pthread_condattr_init(pthread_condattr_t *attr);
```

32922

32923 **DESCRIPTION**

32924 The *pthread\_condattr\_destroy()* function shall destroy a condition variable attributes object; the  
 32925 object becomes, in effect, uninitialized. An implementation may cause *pthread\_condattr\_destroy()*  
 32926 to set the object referenced by *attr* to an invalid value. A destroyed *attr* attributes object can be  
 32927 reinitialized using *pthread\_condattr\_init()*; the results of otherwise referencing the object after it  
 32928 has been destroyed are undefined.

32929 The *pthread\_condattr\_init()* function shall initialize a condition variable attributes object *attr* with  
 32930 the default value for all of the attributes defined by the implementation.

32931 Results are undefined if *pthread\_condattr\_init()* is called specifying an already initialized *attr*  
 32932 attributes object.

32933 After a condition variable attributes object has been used to initialize one or more condition  
 32934 variables, any function affecting the attributes object (including destruction) shall not affect any  
 32935 previously initialized condition variables.

32936 This volume of IEEE Std 1003.1-200x requires two attributes, the *clock* attribute and the *process-*  
 32937 *shared* attribute.

32938 Additional attributes, their default values, and the names of the associated functions to get and  
 32939 set those attribute values are implementation-defined.

32940 **RETURN VALUE**

32941 If successful, the *pthread\_condattr\_destroy()* and *pthread\_condattr\_init()* functions shall return  
 32942 zero; otherwise, an error number shall be returned to indicate the error.

32943 **ERRORS**

32944 The *pthread\_condattr\_destroy()* function may fail if:

32945 [EINVAL] The value specified by *attr* is invalid.

32946 The *pthread\_condattr\_init()* function shall fail if:

32947 [ENOMEM] Insufficient memory exists to initialize the condition variable attributes object.

32948 These functions shall not return an error code of [EINTR].

32949 **EXAMPLES**

32950 None.

32951 **APPLICATION USAGE**

32952 None.

32953 **RATIONALE**

32954 See *pthread\_attr\_init()* and *pthread\_mutex\_init()*.

32955 A *process-shared* attribute has been defined for condition variables for the same reason it has been  
 32956 defined for mutexes.

32957 **FUTURE DIRECTIONS**

32958       None.

32959 **SEE ALSO**32960       *pthread\_cond\_destroy()*, *pthread\_condattr\_getpshared()*, *pthread\_create()*, *pthread\_mutex\_destroy()*,  
32961       the Base Definitions volume of IEEE Std 1003.1-200x, <**pthread.h**>32962 **CHANGE HISTORY**

32963       First released in Issue 5. Included for alignment with the POSIX Threads Extension.

32964 **Issue 6**32965       The *pthread\_condattr\_destroy()* and *pthread\_condattr\_init()* functions are marked as part of the  
32966       Threads option.

32967 **NAME**

32968 pthread\_condattr\_getclock, pthread\_condattr\_setclock — get and set the clock selection  
 32969 condition variable attribute (**ADVANCED REALTIME**)

32970 **SYNOPSIS**

32971 THR CS #include <pthread.h>

```
32972 int pthread_condattr_getclock(const pthread_condattr_t *restrict attr,
32973 clockid_t *restrict clock_id);
32974 int pthread_condattr_setclock(pthread_condattr_t *attr,
32975 clockid_t clock_id);
32976
```

32977 **DESCRIPTION**

32978 The *pthread\_condattr\_getclock()* function shall obtain the value of the *clock* attribute from the  
 32979 attributes object referenced by *attr*. The *pthread\_condattr\_setclock()* function shall set the *clock*  
 32980 attribute in an initialized attributes object referenced by *attr*. If *pthread\_condattr\_setclock()* is  
 32981 called with a *clock\_id* argument that refers to a CPU-time clock, the call shall fail.

32982 The *clock* attribute is the clock ID of the clock that shall be used to measure the timeout service of  
 32983 *pthread\_cond\_timedwait()*. The default value of the *clock* attribute shall refer to the system clock.

32984 **RETURN VALUE**

32985 If successful, the *pthread\_condattr\_getclock()* function shall return zero and store the value of the  
 32986 clock attribute of *attr* into the object referenced by the *clock\_id* argument. Otherwise, an error  
 32987 number shall be returned to indicate the error.

32988 If successful, the *pthread\_condattr\_setclock()* function shall return zero; otherwise, an error  
 32989 number shall be returned to indicate the error.

32990 **ERRORS**

32991 These functions may fail if:

32992 [EINVAL] The value specified by *attr* is invalid.

32993 The *pthread\_condattr\_setclock()* function may fail if:

32994 [EINVAL] The value specified by *clock\_id* does not refer to a known clock, or is a CPU-  
 32995 time clock.

32996 These functions shall not return an error code of [EINTR].

32997 **EXAMPLES**

32998 None.

32999 **APPLICATION USAGE**

33000 None.

33001 **RATIONALE**

33002 None.

33003 **FUTURE DIRECTIONS**

33004 None.

33005 **SEE ALSO**

33006 *pthread\_cond\_init()*, *pthread\_cond\_timedwait()*, *pthread\_condattr\_destroy()*,  
 33007 *pthread\_condattr\_getshared()* (on page 1536), *pthread\_condattr\_init()*,  
 33008 *pthread\_condattr\_setshared()* (on page 1540), *pthread\_create()*, *pthread\_mutex\_init()*, the Base  
 33009 Definitions volume of IEEE Std 1003.1-200x, <pthread.h>

33010 **CHANGE HISTORY**

33011 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

33012 **NAME**

33013 pthread\_condattr\_getpshared, pthread\_condattr\_setpshared — get and set the process-shared  
 33014 condition variable attributes

33015 **SYNOPSIS**

```
33016 THR TSH #include <pthread.h>
33017
33017 int pthread_condattr_getpshared(const pthread_condattr_t *restrict attr,
33018 int *restrict pshared);
33019 int pthread_condattr_setpshared(pthread_condattr_t *attr,
33020 int pshared);
33021
```

33022 **DESCRIPTION**

33023 The *pthread\_condattr\_getpshared()* function shall obtain the value of the *process-shared* attribute  
 33024 from the attributes object referenced by *attr*. The *pthread\_condattr\_setpshared()* function shall set  
 33025 the *process-shared* attribute in an initialized attributes object referenced by *attr*.

33026 The *process-shared* attribute is set to `PTHREAD_PROCESS_SHARED` to permit a condition  
 33027 variable to be operated upon by any thread that has access to the memory where the condition  
 33028 variable is allocated, even if the condition variable is allocated in memory that is shared by  
 33029 multiple processes. If the *process-shared* attribute is `PTHREAD_PROCESS_PRIVATE`, the  
 33030 condition variable shall only be operated upon by threads created within the same process as the  
 33031 thread that initialized the condition variable; if threads of differing processes attempt to operate  
 33032 on such a condition variable, the behavior is undefined. The default value of the attribute is  
 33033 `PTHREAD_PROCESS_PRIVATE`.

33034 **RETURN VALUE**

33035 If successful, the *pthread\_condattr\_setpshared()* function shall return zero; otherwise, an error  
 33036 number shall be returned to indicate the error.

33037 If successful, the *pthread\_condattr\_getpshared()* function shall return zero and store the value of  
 33038 the *process-shared* attribute of *attr* into the object referenced by the *pshared* parameter. Otherwise,  
 33039 an error number shall be returned to indicate the error.

33040 **ERRORS**

33041 The *pthread\_condattr\_getpshared()* and *pthread\_condattr\_setpshared()* functions may fail if:

33042 [EINVAL] The value specified by *attr* is invalid.

33043 The *pthread\_condattr\_setpshared()* function may fail if:

33044 [EINVAL] The new value specified for the attribute is outside the range of legal values  
 33045 for that attribute.

33046 These functions shall not return an error code of [EINTR].

33047 **EXAMPLES**

33048 None.

33049 **APPLICATION USAGE**

33050 None.

33051 **RATIONALE**

33052 None.

33053 **FUTURE DIRECTIONS**

33054 None.

33055 **SEE ALSO**

33056 *pthread\_create()*, *pthread\_cond\_destroy()*, *pthread\_condattr\_destroy()*, *pthread\_mutex\_destroy()*, the  
33057 Base Definitions volume of IEEE Std 1003.1-200x, <pthread.h>

33058 **CHANGE HISTORY**

33059 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

33060 **Issue 6**

33061 The *pthread\_condattr\_getpshared()* and *pthread\_condattr\_setpshared()* functions are marked as part  
33062 of the Threads and Thread Process-Shared Synchronization options.

33063 The **restrict** keyword is added to the *pthread\_condattr\_getpshared()* prototype for alignment with  
33064 the ISO/IEC 9899:1999 standard.

33065 **NAME**

33066 pthread\_condattr\_init — initialize condition variable attributes object

33067 **SYNOPSIS**

33068 THR #include <pthread.h>

33069 int pthread\_condattr\_init(pthread\_condattr\_t \*attr);

33070

33071 **DESCRIPTION**

33072 Refer to *pthread\_condattr\_destroy()*.

33073 **NAME**

33074 pthread\_condattr\_setclock — set the clock selection condition variable attribute

33075 **SYNOPSIS**

33076 THR CS #include <pthread.h>

```
33077 int pthread_condattr_setclock(pthread_condattr_t *attr,  
33078 clockid_t clock_id);
```

33079

33080 **DESCRIPTION**

33081 Refer to *pthread\_condattr\_getclock()*.

33082 **NAME**

33083 pthread\_condattr\_setpshared — set the process-shared condition variable attributes

33084 **SYNOPSIS**

33085 THR TSH #include <pthread.h>

```
33086 int pthread_condattr_setpshared(pthread_condattr_t *attr,  
33087 int pshared);
```

33088

33089 **DESCRIPTION**

33090 Refer to *pthread\_condattr\_getpshared()*.

33091 **NAME**

33092 pthread\_create — thread creation

33093 **SYNOPSIS**

33094 THR #include &lt;pthread.h&gt;

```
33095 int pthread_create(pthread_t *restrict thread,
33096                  const pthread_attr_t *restrict attr,
33097                  void *(*start_routine)(void*), void *restrict arg);
33098
```

33099 **DESCRIPTION**

33100 The *pthread\_create()* function shall create a new thread, with attributes specified by *attr*, within a |  
 33101 process. If *attr* is NULL, the default attributes shall be used. If the attributes specified by *attr* are |  
 33102 modified later, the thread's attributes shall not be affected. Upon successful completion, |  
 33103 *pthread\_create()* shall store the ID of the created thread in the location referenced by *thread*.

33104 The thread is created executing *start\_routine* with *arg* as its sole argument. If the *start\_routine* |  
 33105 returns, the effect shall be as if there was an implicit call to *pthread\_exit()* using the return value |  
 33106 of *start\_routine* as the exit status. Note that the thread in which *main()* was originally invoked |  
 33107 differs from this. When it returns from *main()*, the effect shall be as if there was an implicit call |  
 33108 to *exit()* using the return value of *main()* as the exit status.

33109 The signal state of the new thread shall be initialized as follows:

- 33110 • The signal mask shall be inherited from the creating thread.
- 33111 • The set of signals pending for the new thread shall be empty.

33112 The floating-point environment shall be inherited from the creating thread. |

33113 If *pthread\_create()* fails, no new thread is created and the contents of the location referenced by |  
 33114 *thread* are undefined.

33115 TCT If `_POSIX_THREAD_CPUTIME` is defined, the new thread shall have a CPU-time clock |  
 33116 accessible, and the initial value of this clock shall be set to zero.

33117 **RETURN VALUE**

33118 If successful, the *pthread\_create()* function shall return zero; otherwise, an error number shall be |  
 33119 returned to indicate the error.

33120 **ERRORS**

33121 The *pthread\_create()* function shall fail if:

33122 [EAGAIN] The system lacked the necessary resources to create another thread, or the |  
 33123 system-imposed limit on the total number of threads in a process |  
 33124 PTHREAD\_THREADS\_MAX would be exceeded.

33125 [EINVAL] The value specified by *attr* is invalid.

33126 [EPERM] The caller does not have appropriate permission to set the required |  
 33127 scheduling parameters or scheduling policy.

33128 The *pthread\_create()* function shall not return an error code of [EINTR].

33129 **EXAMPLES**

33130 None.

33131 **APPLICATION USAGE**

33132 None.

33133 **RATIONALE**

33134 A suggested alternative to *pthread\_create()* would be to define two separate operations: create  
 33135 and start. Some applications would find such behavior more natural. Ada, in particular,  
 33136 separates the “creation” of a task from its “activation”.

33137 Splitting the operation was rejected by the standard developers for many reasons:

- 33138 • The number of calls required to start a thread would increase from one to two and thus place  
 33139 an additional burden on applications that do not require the additional synchronization. The  
 33140 second call, however, could be avoided by the additional complication of a start-up state  
 33141 attribute.
- 33142 • An extra state would be introduced: “created but not started”. This would require the  
 33143 standard to specify the behavior of the thread operations when the target has not yet started  
 33144 executing.
- 33145 • For those applications that require such behavior, it is possible to simulate the two separate  
 33146 steps with the facilities that are currently provided. The *start\_routine()* can synchronize by  
 33147 waiting on a condition variable that is signaled by the start operation.

33148 An Ada implementor can choose to create the thread at either of two points in the Ada program:  
 33149 when the task object is created, or when the task is activated (generally at a “begin”). If the first  
 33150 approach is adopted, the *start\_routine()* needs to wait on a condition variable to receive the  
 33151 order to begin “activation”. The second approach requires no such condition variable or extra  
 33152 synchronization. In either approach, a separate Ada task control block would need to be created  
 33153 when the task object is created to hold rendezvous queues, and so on.

33154 An extension of the preceding model would be to allow the state of the thread to be modified  
 33155 between the create and start. This would allow the thread attributes object to be eliminated. This  
 33156 has been rejected because:

- 33157 • All state in the thread attributes object has to be able to be set for the thread. This would  
 33158 require the definition of functions to modify thread attributes. There would be no reduction  
 33159 in the number of function calls required to set up the thread. In fact, for an application that  
 33160 creates all threads using identical attributes, the number of function calls required to set up  
 33161 the threads would be dramatically increased. Use of a thread attributes object permits the  
 33162 application to make one set of attribute setting function calls. Otherwise, the set of attribute  
 33163 setting function calls needs to be made for each thread creation.
- 33164 • Depending on the implementation architecture, functions to set thread state would require  
 33165 kernel calls, or for other implementation reasons would not be able to be implemented as  
 33166 macros, thereby increasing the cost of thread creation.
- 33167 • The ability for applications to segregate threads by class would be lost.

33168 Another suggested alternative uses a model similar to that for process creation, such as “thread  
 33169 fork”. The fork semantics would provide more flexibility and the “create” function can be  
 33170 implemented simply by doing a thread fork followed immediately by a call to the desired “start  
 33171 routine” for the thread. This alternative has these problems:

- 33172 • For many implementations, the entire stack of the calling thread would need to be  
 33173 duplicated, since in many architectures there is no way to determine the size of the calling  
 33174 frame.

33175           • Efficiency is reduced since at least some part of the stack has to be copied, even though in  
33176           most cases the thread never needs the copied context, since it merely calls the desired start  
33177           routine.

33178 **FUTURE DIRECTIONS**

33179           None.

33180 **SEE ALSO**

33181           *fork()*, *pthread\_exit()*, *pthread\_join()*, the Base Definitions volume of IEEE Std 1003.1-200x,  
33182           <pthread.h>

33183 **CHANGE HISTORY**

33184           First released in Issue 5. Included for alignment with the POSIX Threads Extension.

33185 **Issue 6**

33186           The *pthread\_create()* function is marked as part of the Threads option.

33187           The following new requirements on POSIX implementations derive from alignment with the  
33188           Single UNIX Specification:

33189           • The [EPERM] mandatory error condition is added.

33190           The thread CPU-time clock semantics are added for alignment with IEEE Std 1003.1d-1999.

33191           The **restrict** keyword is added to the *pthread\_create()* prototype for alignment with the  
33192           ISO/IEC 9899:1999 standard. |

33193           The DESCRIPTION is updated to make it explicit that the floating-point environment is |  
33194           inherited from the creating thread. |

33195 **NAME**

33196 pthread\_detach — detach a thread

33197 **SYNOPSIS**

33198 THR #include &lt;pthread.h&gt;

33199 int pthread\_detach(pthread\_t thread);

33200

33201 **DESCRIPTION**

33202 The *pthread\_detach()* function shall indicate to the implementation that storage for the thread  
33203 *thread* can be reclaimed when that thread terminates. If *thread* has not terminated,  
33204 *pthread\_detach()* shall not cause it to terminate. The effect of multiple *pthread\_detach()* calls on  
33205 the same target thread is unspecified.

33206 **RETURN VALUE**

33207 If the call succeeds, *pthread\_detach()* shall return 0; otherwise, an error number shall be returned  
33208 to indicate the error.

33209 **ERRORS**33210 The *pthread\_detach()* function shall fail if:

33211 [EINVAL] The implementation has detected that the value specified by *thread* does not  
33212 refer to a joinable thread.

33213 [ESRCH] No thread could be found corresponding to that specified by the given thread  
33214 ID.

33215 The *pthread\_detach()* function shall not return an error code of [EINTR].33216 **EXAMPLES**

33217 None.

33218 **APPLICATION USAGE**

33219 None.

33220 **RATIONALE**

33221 The *pthread\_join()* or *pthread\_detach()* functions should eventually be called for every thread that  
33222 is created so that storage associated with the thread may be reclaimed.

33223 It has been suggested that a “detach” function is not necessary; the *detachstate* thread creation  
33224 attribute is sufficient, since a thread need never be dynamically detached. However, need arises  
33225 in at least two cases:

33226 1. In a cancellation handler for a *pthread\_join()* it is nearly essential to have a *pthread\_detach()*  
33227 function in order to detach the thread on which *pthread\_join()* was waiting. Without it, it  
33228 would be necessary to have the handler do another *pthread\_join()* to attempt to detach the  
33229 thread, which would both delay the cancellation processing for an unbounded period and  
33230 introduce a new call to *pthread\_join()*, which might itself need a cancellation handler. A  
33231 dynamic detach is nearly essential in this case.

33232 2. In order to detach the “initial thread” (as may be desirable in processes that set up server  
33233 threads).

33234 **FUTURE DIRECTIONS**

33235 None.

33236 **SEE ALSO**

33237 *pthread\_join()*, the Base Definitions volume of IEEE Std 1003.1-200x, <pthread.h>

33238 **CHANGE HISTORY**

33239 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

33240 **Issue 6**

33241 The *pthread\_detach()* function is marked as part of the Threads option.

33242 **NAME**

33243 pthread\_equal — compare thread IDs

33244 **SYNOPSIS**

33245 THR #include &lt;pthread.h&gt;

33246 int pthread\_equal(pthread\_t t1, pthread\_t t2);

33247

33248 **DESCRIPTION**33249 This function shall compare the thread IDs *t1* and *t2*.33250 **RETURN VALUE**33251 The *pthread\_equal()* function shall return a non-zero value if *t1* and *t2* are equal; otherwise, zero  
33252 shall be returned.33253 If either *t1* or *t2* are not valid thread IDs, the behavior is undefined.33254 **ERRORS**

33255 No errors are defined.

33256 The *pthread\_equal()* function shall not return an error code of [EINTR].33257 **EXAMPLES**

33258 None.

33259 **APPLICATION USAGE**

33260 None.

33261 **RATIONALE**33262 Implementations may choose to define a thread ID as a structure. This allows additional  
33263 flexibility and robustness over using an **int**. For example, a thread ID could include a sequence  
33264 number that allows detection of “dangling IDs” (copies of a thread ID that has been detached).  
33265 Since the C language does not support comparison on structure types, the *pthread\_equal()*  
33266 function is provided to compare thread IDs.33267 **FUTURE DIRECTIONS**

33268 None.

33269 **SEE ALSO**33270 *pthread\_create()*, *pthread\_self()*, the Base Definitions volume of IEEE Std 1003.1-200x, <pthread.h>33271 **CHANGE HISTORY**

33272 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

33273 **Issue 6**33274 The *pthread\_equal()* function is marked as part of the Threads option.

33275 **NAME**

33276 pthread\_exit — thread termination

33277 **SYNOPSIS**

33278 THR #include &lt;pthread.h&gt;

33279 void pthread\_exit(void \*value\_ptr);

33280

33281 **DESCRIPTION**

33282 The *pthread\_exit()* function shall terminate the calling thread and make the value *value\_ptr*  
33283 available to any successful join with the terminating thread. Any cancellation cleanup handlers  
33284 that have been pushed and not yet popped shall be popped in the reverse order that they were  
33285 pushed and then executed. After all cancellation cleanup handlers have been executed, if the  
33286 thread has any thread-specific data, appropriate destructor functions shall be called in an  
33287 unspecified order. Thread termination does not release any application visible process resources,  
33288 including, but not limited to, mutexes and file descriptors, nor does it perform any process-level  
33289 cleanup actions, including, but not limited to, calling any *atexit()* routines that may exist.

33290 An implicit call to *pthread\_exit()* is made when a thread other than the thread in which *main()*  
33291 was first invoked returns from the start routine that was used to create it. The function's return  
33292 value shall serve as the thread's exit status.

33293 The behavior of *pthread\_exit()* is undefined if called from a cancellation cleanup handler or  
33294 destructor function that was invoked as a result of either an implicit or explicit call to  
33295 *pthread\_exit()*.

33296 After a thread has terminated, the result of access to local (auto) variables of the thread is  
33297 undefined. Thus, references to local variables of the exiting thread should not be used for the  
33298 *pthread\_exit()* *value\_ptr* parameter value.

33299 The process shall exit with an exit status of 0 after the last thread has been terminated. The  
33300 behavior shall be as if the implementation called *exit()* with a zero argument at thread  
33301 termination time.

33302 **RETURN VALUE**33303 The *pthread\_exit()* function cannot return to its caller.33304 **ERRORS**

33305 No errors are defined.

33306 **EXAMPLES**

33307 None.

33308 **APPLICATION USAGE**

33309 None.

33310 **RATIONALE**

33311 The normal mechanism by which a thread terminates is to return from the routine that was  
33312 specified in the *pthread\_create()* call that started it. The *pthread\_exit()* function provides the  
33313 capability for a thread to terminate without requiring a return from the start routine of that  
33314 thread, thereby providing a function analogous to *exit()*.

33315 Regardless of the method of thread termination, any cancellation cleanup handlers that have  
33316 been pushed and not yet popped are executed, and the destructors for any existing thread-  
33317 specific data are executed. This volume of IEEE Std 1003.1-200x requires that cancellation  
33318 cleanup handlers be popped and called in order. After all cancellation cleanup handlers have  
33319 been executed, thread-specific data destructors are called, in an unspecified order, for each item  
33320 of thread-specific data that exists in the thread. This ordering is necessary because cancellation

- 33321 cleanup handlers may rely on thread-specific data.
- 33322 As the meaning of the status is determined by the application (except when the thread has been
- 33323 canceled, in which case it is PTHREAD\_CANCELED), the implementation has no idea what an
- 33324 illegal status value is, which is why no address error checking is done.
- 33325 **FUTURE DIRECTIONS**
- 33326 None.
- 33327 **SEE ALSO**
- 33328 *exit()*, *pthread\_create()*, *pthread\_join()*, the Base Definitions volume of IEEE Std 1003.1-200x,
- 33329 **<pthread.h>**
- 33330 **CHANGE HISTORY**
- 33331 First released in Issue 5. Included for alignment with the POSIX Threads Extension.
- 33332 **Issue 6**
- 33333 The *pthread\_exit()* function is marked as part of the Threads option.

33334 **NAME**

33335 pthread\_getconcurrency, pthread\_setconcurrency — get and set level of concurrency

33336 **SYNOPSIS**

```
33337 xSI #include <pthread.h>
33338 int pthread_getconcurrency(void);
33339 int pthread_setconcurrency(int new_level);
33340
```

33341 **DESCRIPTION**

33342 Unbound threads in a process may or may not be required to be simultaneously active. By  
 33343 default, the threads implementation ensures that a sufficient number of threads are active so that  
 33344 the process can continue to make progress. While this conserves system resources, it may not  
 33345 produce the most effective level of concurrency.

33346 The *pthread\_setconcurrency()* function allows an application to inform the threads  
 33347 implementation of its desired concurrency level, *new\_level*. The actual level of concurrency  
 33348 provided by the implementation as a result of this function call is unspecified.

33349 If *new\_level* is zero, it causes the implementation to maintain the concurrency level at its  
 33350 discretion as if *pthread\_setconcurrency()* had never been called.

33351 The *pthread\_getconcurrency()* function shall return the value set by a previous call to the  
 33352 *pthread\_setconcurrency()* function. If the *pthread\_setconcurrency()* function was not previously  
 33353 called, this function shall return zero to indicate that the implementation is maintaining the  
 33354 concurrency level.

33355 A call to *pthread\_setconcurrency()* shall inform the implementation of its desired concurrency |  
 33356 level. The implementation shall use this as a hint, not a requirement. |

33357 If an implementation does not support multiplexing of user threads on top of several kernel-  
 33358 scheduled entities, the *pthread\_setconcurrency()* and *pthread\_getconcurrency()* functions are |  
 33359 provided for source code compatibility but they shall have no effect when called. To maintain |  
 33360 the function semantics, the *new\_level* parameter is saved when *pthread\_setconcurrency()* is called  
 33361 so that a subsequent call to *pthread\_getconcurrency()* shall return the same value.

33362 **RETURN VALUE**

33363 If successful, the *pthread\_setconcurrency()* function shall return zero; otherwise, an error number  
 33364 shall be returned to indicate the error.

33365 The *pthread\_getconcurrency()* function shall always return the concurrency level set by a previous  
 33366 call to *pthread\_setconcurrency()*. If the *pthread\_setconcurrency()* function has never been called,  
 33367 *pthread\_getconcurrency()* shall return zero.

33368 **ERRORS**

33369 The *pthread\_setconcurrency()* function shall fail if:

33370 [EINVAL] The value specified by *new\_level* is negative.

33371 [EAGAIN] The value specific by *new\_level* would cause a system resource to be exceeded.

33372 These functions shall not return an error code of [EINTR].

33373 **EXAMPLES**

33374 None.

33375 **APPLICATION USAGE**

33376 Use of these functions changes the state of the underlying concurrency upon which the  
33377 application depends. Library developers are advised to not use the *pthread\_getconcurrency()* and  
33378 *pthread\_setconcurrency()* functions since their use may conflict with an applications use of these  
33379 functions.

33380 **RATIONALE**

33381 None.

33382 **FUTURE DIRECTIONS**

33383 None.

33384 **SEE ALSO**

33385 The Base Definitions volume of IEEE Std 1003.1-200x, <pthread.h>

33386 **CHANGE HISTORY**

33387 First released in Issue 5.

33388 **NAME**

33389 pthread\_getcpuclockid — access a thread CPU-time clock (**ADVANCED REALTIME**  
33390 **THREADS**)

33391 **SYNOPSIS**

```
33392 THR TCT #include <pthread.h>
```

```
33393 #include <time.h>
```

```
33394 int pthread_getcpuclockid(pthread_t thread_id, clockid_t *clock_id);
```

33395

33396 **DESCRIPTION**

33397 The *pthread\_getcpuclockid()* function shall return in *clock\_id* the clock ID of the CPU-time clock of  
33398 the thread specified by *thread\_id*, if the thread specified by *thread\_id* exists.

33399 **RETURN VALUE**

33400 Upon successful completion, *pthread\_getcpuclockid()* shall return zero; otherwise, an error  
33401 number shall be returned to indicate the error.

33402 **ERRORS**

33403 The *pthread\_getcpuclockid()* function may fail if:

33404 [ESRCH] The value specified by *thread\_id* does not refer to an existing thread.

33405 **EXAMPLES**

33406 None.

33407 **APPLICATION USAGE**

33408 The *pthread\_getcpuclockid()* function is part of the Thread CPU-Time Clocks option and need not  
33409 be provided on all implementations.

33410 **RATIONALE**

33411 None.

33412 **FUTURE DIRECTIONS**

33413 None.

33414 **SEE ALSO**

33415 *clock\_getcpuclockid()*, *clock\_getres()*, *timer\_create()*, the Base Definitions volume of  
33416 IEEE Std 1003.1-200x, <pthread.h>, <time.h>

33417 **CHANGE HISTORY**

33418 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

33419 In the SYNOPSIS, the inclusion of <sys/types.h> is no longer required.

## 33420 NAME

33421 pthread\_getschedparam, pthread\_setschedparam — dynamic thread scheduling parameters  
 33422 access (**REALTIME THREADS**)

## 33423 SYNOPSIS

33424 THR TPS #include <pthread.h>

```
33425 int pthread_getschedparam(pthread_t thread, int *restrict policy,
33426 struct sched_param *restrict param);
```

```
33427 int pthread_setschedparam(pthread_t thread, int policy,
33428 const struct sched_param *param);
```

33429

## 33430 DESCRIPTION

33431 The *pthread\_getschedparam()* and *pthread\_setschedparam()* functions shall, respectively, get and set |  
 33432 the scheduling policy and parameters of individual threads within a multi-threaded process to |  
 33433 be retrieved and set. For SCHED\_FIFO and SCHED\_RR, the only required member of the |  
 33434 **sched\_param** structure is the priority *sched\_priority*. For SCHED\_OTHER, the affected |  
 33435 scheduling parameters are implementation-defined.

33436 The *pthread\_getschedparam()* function shall retrieve the scheduling policy and scheduling |  
 33437 parameters for the thread whose thread ID is given by *thread* and shall store those values in |  
 33438 *policy* and *param*, respectively. The priority value returned from *pthread\_getschedparam()* shall be |  
 33439 the value specified by the most recent *pthread\_setschedparam()*, *pthread\_setschedprio()*, or |  
 33440 *pthread\_create()* call affecting the target thread. It shall not reflect any temporary adjustments to |  
 33441 its priority as a result of any priority inheritance or ceiling functions. The *pthread\_setschedparam()* |  
 33442 function shall set the scheduling policy and associated scheduling parameters for the thread |  
 33443 whose thread ID is given by *thread* to the policy and associated parameters provided in *policy* |  
 33444 and *param*, respectively.

33445 The *policy* parameter may have the value SCHED\_OTHER, SCHED\_FIFO, or SCHED\_RR. The |  
 33446 scheduling parameters for the SCHED\_OTHER policy are implementation-defined. The |  
 33447 SCHED\_FIFO and SCHED\_RR policies shall have a single scheduling parameter, *priority*.

33448 TSP If **\_POSIX\_THREAD\_SPORADIC\_SERVER** is defined, then the *policy* argument may have the |  
 33449 value SCHED\_SPORADIC, with the exception for the *pthread\_setschedparam()* function that if the |  
 33450 scheduling policy was not SCHED\_SPORADIC at the time of the call, it is implementation- |  
 33451 defined whether the function is supported; in other words, the implementation need not allow |  
 33452 the application to dynamically change the scheduling policy to SCHED\_SPORADIC. The |  
 33453 sporadic server scheduling policy has the associated parameters *sched\_ss\_low\_priority*, |  
 33454 *sched\_ss\_repl\_period*, *sched\_ss\_init\_budget*, *sched\_priority*, and *sched\_ss\_max\_repl*. The specified |  
 33455 *sched\_ss\_repl\_period* shall be greater than or equal to the specified *sched\_ss\_init\_budget* for the |  
 33456 function to succeed; if it is not, then the function shall fail. The value of *sched\_ss\_max\_repl* shall |  
 33457 be within the inclusive range [1,{SS\_REPL\_MAX}] for the function to succeed; if not, the function |  
 33458 shall fail.

33459 If the *pthread\_setschedparam()* function fails, the scheduling parameters shall not be changed for |  
 33460 the target thread. |

## 33461 RETURN VALUE

33462 If successful, the *pthread\_getschedparam()* and *pthread\_setschedparam()* functions shall return zero; |  
 33463 otherwise, an error number shall be returned to indicate the error.

33464 **ERRORS**

33465 The *pthread\_getschedparam()* function may fail if:

33466 [ESRCH] The value specified by *thread* does not refer to a existing thread.

33467 The *pthread\_setschedparam()* function may fail if:

33468 [EINVAL] The value specified by *policy* or one of the scheduling parameters associated  
33469 with the scheduling policy *policy* is invalid.

33470 [ENOTSUP] An attempt was made to set the policy or scheduling parameters to an  
33471 unsupported value.

33472 TSP [ENOTSUP] An attempt was made to dynamically change the scheduling policy to  
33473 SCHED\_SPORADIC, and the implementation does not support this change.

33474 [EPERM] The caller does not have the appropriate permission to set either the  
33475 scheduling parameters or the scheduling policy of the specified thread.

33476 [EPERM] The implementation does not allow the application to modify one of the  
33477 parameters to the value specified.

33478 [ESRCH] The value specified by *thread* does not refer to a existing thread.

33479 These functions shall not return an error code of [EINTR].

33480 **EXAMPLES**

33481 None.

33482 **APPLICATION USAGE**

33483 None.

33484 **RATIONALE**

33485 None.

33486 **FUTURE DIRECTIONS**

33487 None.

33488 **SEE ALSO**

33489 *pthread\_setschedprio()*, *sched\_getparam()*, *sched\_getscheduler()*, the Base Definitions volume of |  
33490 IEEE Std 1003.1-200x, <pthread.h>, <sched.h>

33491 **CHANGE HISTORY**

33492 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

33493 **Issue 6**

33494 The *pthread\_getschedparam()* and *pthread\_setschedparam()* functions are marked as part of the  
33495 Threads and Thread Execution Scheduling options.

33496 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
33497 implementation does not support the Thread Execution Scheduling option.

33498 The Open Group Corrigendum U026/2 is applied, correcting the prototype for the  
33499 *pthread\_setschedparam()* function so that its second argument is of type **int**.

33500 The SCHED\_SPORADIC scheduling policy is added for alignment with IEEE Std 1003.1d-1999.

33501 The **restrict** keyword is added to the *pthread\_getschedparam()* prototype for alignment with the  
33502 ISO/IEC 9899:1999 standard.

33503 The Open Group Corrigendum U047/1 is applied. |

33504  
33505

IEEE PASC Interpretation 1003.1 #96 is applied, noting that priority values can also be set by a call to the *pthread\_setschedprio()* function.

33506 **NAME**

33507 pthread\_getspecific, pthread\_setspecific — thread-specific data management

33508 **SYNOPSIS**

33509 THR #include &lt;pthread.h&gt;

33510 void \*pthread\_getspecific(pthread\_key\_t key);

33511 int pthread\_setspecific(pthread\_key\_t key, const void \*value);

33512

33513 **DESCRIPTION**33514 The *pthread\_getspecific()* function shall return the value currently bound to the specified *key* on  
33515 behalf of the calling thread.33516 The *pthread\_setspecific()* function shall associate a thread-specific *value* with a *key* obtained via a  
33517 previous call to *pthread\_key\_create()*. Different threads may bind different values to the same  
33518 key. These values are typically pointers to blocks of dynamically allocated memory that have  
33519 been reserved for use by the calling thread.33520 The effect of calling *pthread\_getspecific()* or *pthread\_setspecific()* with a *key* value not obtained  
33521 from *pthread\_key\_create()* or after *key* has been deleted with *pthread\_key\_delete()* is undefined.33522 Both *pthread\_getspecific()* and *pthread\_setspecific()* may be called from a thread-specific data  
33523 destructor function. A call to *pthread\_getspecific()* for the thread-specific data key being  
33524 destroyed shall return the value NULL, unless the value is changed (after the destructor starts)  
33525 by a call to *pthread\_setspecific()*. Calling *pthread\_setspecific()* from a thread-specific data  
33526 destructor routine may result either in lost storage (after at least  
33527 PTHREAD\_DESTRUCTOR\_ITERATIONS attempts at destruction) or in an infinite loop.

33528 Both functions may be implemented as macros.

33529 **RETURN VALUE**33530 The *pthread\_getspecific()* function shall return the thread-specific data value associated with the  
33531 given *key*. If no thread-specific data value is associated with *key*, then the value NULL shall be  
33532 returned.33533 If successful, the *pthread\_setspecific()* function shall return zero; otherwise, an error number shall  
33534 be returned to indicate the error.33535 **ERRORS**33536 No errors are returned from *pthread\_getspecific()*.33537 The *pthread\_setspecific()* function shall fail if:

33538 [ENOMEM] Insufficient memory exists to associate the value with the key.

33539 The *pthread\_setspecific()* function may fail if:

33540 [EINVAL] The key value is invalid.

33541 These functions shall not return an error code of [EINTR].

33542 **EXAMPLES**

33543           None.

33544 **APPLICATION USAGE**

33545           None.

33546 **RATIONALE**

33547           Performance and ease-of-use of *pthread\_getspecific()* is critical for functions that rely on  
33548           maintaining state in thread-specific data. Since no errors are required to be detected by it, and  
33549           since the only error that could be detected is the use of an invalid key, the function to  
33550           *pthread\_getspecific()* has been designed to favor speed and simplicity over error reporting.

33551 **FUTURE DIRECTIONS**

33552           None.

33553 **SEE ALSO**

33554           *pthread\_key\_create()*, the Base Definitions volume of IEEE Std 1003.1-200x, <**pthread.h**>

33555 **CHANGE HISTORY**

33556           First released in Issue 5. Included for alignment with the POSIX Threads Extension.

33557 **Issue 6**

33558           The *pthread\_getspecific()* and *pthread\_setspecific()* functions are marked as part of the Threads  
33559           option.

33560           IEEE PASC Interpretation 1003.1c #3 (Part 6) is applied, updating the DESCRIPTION.

33561 **NAME**

33562 pthread\_join — wait for thread termination

33563 **SYNOPSIS**

33564 THR #include &lt;pthread.h&gt;

33565 int pthread\_join(pthread\_t thread, void \*\*value\_ptr);

33566

33567 **DESCRIPTION**

33568 The *pthread\_join()* function shall suspend execution of the calling thread until the target *thread*  
 33569 terminates, unless the target *thread* has already terminated. On return from a successful  
 33570 *pthread\_join()* call with a non-NULL *value\_ptr* argument, the value passed to *pthread\_exit()* by  
 33571 the terminating thread shall be made available in the location referenced by *value\_ptr*. When a  
 33572 *pthread\_join()* returns successfully, the target thread has been terminated. The results of multiple  
 33573 simultaneous calls to *pthread\_join()* specifying the same target thread are undefined. If the  
 33574 thread calling *pthread\_join()* is canceled, then the target thread shall not be detached.

33575 It is unspecified whether a thread that has exited but remains unjoined counts against  
 33576 \_PTHREAD\_THREADS\_MAX.

33577 **RETURN VALUE**

33578 If successful, the *pthread\_join()* function shall return zero; otherwise, an error number shall be  
 33579 returned to indicate the error.

33580 **ERRORS**33581 The *pthread\_join()* function shall fail if:

33582 [EINVAL] The implementation has detected that the value specified by *thread* does not  
 33583 refer to a joinable thread.

33584 [ESRCH] No thread could be found corresponding to that specified by the given thread  
 33585 ID.

33586 The *pthread\_join()* function may fail if:

33587 [EDEADLK] A deadlock was detected or the value of *thread* specifies the calling thread.

33588 The *pthread\_join()* function shall not return an error code of [EINTR].33589 **EXAMPLES**

33590 An example of thread creation and deletion follows:

```

33591 typedef struct {
33592     int *ar;
33593     long n;
33594 } subarray;
33595
33596 void *
33597 incer(void *arg)
33598 {
33599     long i;
33599     for (i = 0; i < ((subarray *)arg)->n; i++)
33600         ((subarray *)arg)->ar[i]++;
33601 }
33602
33603 main()
33604 {
33604     int         ar[1000000];

```

```

33605         pthread_t  th1, th2;
33606         subarray  sb1, sb2;

33607         sb1.ar = &ar[0];
33608         sb1.n  = 500000;
33609         (void) pthread_create(&th1, NULL, incer, &sb1);

33610         sb2.ar = &ar[500000];
33611         sb2.n  = 500000;
33612         (void) pthread_create(&th2, NULL, incer, &sb2);

33613         (void) pthread_join(th1, NULL);
33614         (void) pthread_join(th2, NULL);
33615     }

```

#### 33616 APPLICATION USAGE

33617 None.

#### 33618 RATIONALE

33619 The *pthread\_join()* function is a convenience that has proven useful in multi-threaded  
 33620 applications. It is true that a programmer could simulate this function if it were not provided by  
 33621 passing extra state as part of the argument to the *start\_routine()*. The terminating thread would  
 33622 set a flag to indicate termination and broadcast a condition that is part of that state; a joining  
 33623 thread would wait on that condition variable. While such a technique would allow a thread to  
 33624 wait on more complex conditions (for example, waiting for multiple threads to terminate),  
 33625 waiting on individual thread termination is considered widely useful. Also, including the  
 33626 *pthread\_join()* function in no way precludes a programmer from coding such complex waits.  
 33627 Thus, while not a primitive, including *pthread\_join()* in this volume of IEEE Std 1003.1-200x was  
 33628 considered valuable.

33629 The *pthread\_join()* function provides a simple mechanism allowing an application to wait for a  
 33630 thread to terminate. After the thread terminates, the application may then choose to clean up  
 33631 resources that were used by the thread. For instance, after *pthread\_join()* returns, any  
 33632 application-provided stack storage could be reclaimed.

33633 The *pthread\_join()* or *pthread\_detach()* function should eventually be called for every thread that  
 33634 is created with the *detachstate* attribute set to *PTHREAD\_CREATE\_JOINABLE* so that storage  
 33635 associated with the thread may be reclaimed.

33636 The interaction between *pthread\_join()* and cancelation is well-defined for the following reasons:

- 33637 • The *pthread\_join()* function, like all other non-async-cancel-safe functions, can only be called  
 33638 with deferred cancelability type.
- 33639 • Cancelation cannot occur in the disabled cancelability state.

33640 Thus, only the default cancelability state need be considered. As specified, either the  
 33641 *pthread\_join()* call is canceled, or it succeeds, but not both. The difference is obvious to the  
 33642 application, since either a cancelation handler is run or *pthread\_join()* returns. There are no race  
 33643 conditions since *pthread\_join()* was called in the deferred cancelability state.

#### 33644 FUTURE DIRECTIONS

33645 None.

#### 33646 SEE ALSO

33647 *pthread\_create()*, *wait()*, the Base Definitions volume of IEEE Std 1003.1-200x, <**pthread.h**>

33648 **CHANGE HISTORY**

33649 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

33650 **Issue 6**

33651 The *pthread\_join()* function is marked as part of the Threads option.

33652 **NAME**

33653 pthread\_key\_create — thread-specific data key creation

33654 **SYNOPSIS**

33655 THR #include &lt;pthread.h&gt;

33656 int pthread\_key\_create(pthread\_key\_t \*key, void (\*destructor)(void\*));

33657

33658 **DESCRIPTION**

33659 The *pthread\_key\_create()* function shall create a thread-specific data key visible to all threads in  
33660 the process. Key values provided by *pthread\_key\_create()* are opaque objects used to locate  
33661 thread-specific data. Although the same key value may be used by different threads, the values  
33662 bound to the key by *pthread\_setspecific()* are maintained on a per-thread basis and persist for the  
33663 life of the calling thread.

33664 Upon key creation, the value NULL shall be associated with the new key in all active threads.  
33665 Upon thread creation, the value NULL shall be associated with all defined keys in the new  
33666 thread.

33667 An optional destructor function may be associated with each key value. At thread exit, if a key  
33668 value has a non-NULL destructor pointer, and the thread has a non-NULL value associated with  
33669 that key, the value of the key is set to NULL, and then the function pointed to is called with the  
33670 previously associated value as its sole argument. The order of destructor calls is unspecified if  
33671 more than one destructor exists for a thread when it exits.

33672 If, after all the destructors have been called for all non-NULL values with associated destructors,  
33673 there are still some non-NULL values with associated destructors, then the process is repeated.  
33674 If, after at least {PTHREAD\_DESTRUCTOR\_ITERATIONS} iterations of destructor calls for  
33675 outstanding non-NULL values, there are still some non-NULL values with associated  
33676 destructors, implementations may stop calling destructors, or they may continue calling  
33677 destructors until no non-NULL values with associated destructors exist, even though this might  
33678 result in an infinite loop.

33679 **RETURN VALUE**

33680 If successful, the *pthread\_key\_create()* function shall store the newly created key value at *\*key* and  
33681 shall return zero. Otherwise, an error number shall be returned to indicate the error.

33682 **ERRORS**33683 The *pthread\_key\_create()* function shall fail if:

33684 [EAGAIN] The system lacked the necessary resources to create another thread-specific  
33685 data key, or the system-imposed limit on the total number of keys per process  
33686 PTHREAD\_KEYS\_MAX has been exceeded.

33687 [ENOMEM] Insufficient memory exists to create the key.

33688 The *pthread\_key\_create()* function shall not return an error code of [EINTR].

33689 **EXAMPLES**

33690 The following example demonstrates a function that initializes a thread-specific data key when  
 33691 it is first called, and associates a thread-specific object with each calling thread, initializing this  
 33692 object when necessary.

```

33693     static pthread_key_t key;
33694     static pthread_once_t key_once = PTHREAD_ONCE_INIT;

33695     static void
33696     make_key()
33697     {
33698         (void) pthread_key_create(&key, NULL);
33699     }

33700     func()
33701     {
33702         void *ptr;

33703         (void) pthread_once(&key_once, make_key);
33704         if ((ptr = pthread_getspecific(key)) == NULL) {
33705             ptr = malloc(OBJECT_SIZE);
33706             ...
33707             (void) pthread_setspecific(key, ptr);
33708         }
33709         ...
33710     }
  
```

33711 Note that the key has to be initialized before *pthread\_getspecific()* or *pthread\_setspecific()* can be  
 33712 used. The *pthread\_key\_create()* call could either be explicitly made in a module initialization  
 33713 routine, or it can be done implicitly by the first call to a module as in this example. Any attempt  
 33714 to use the key before it is initialized is a programming error, making the code below incorrect.

```

33715     static pthread_key_t key;

33716     func()
33717     {
33718         void *ptr;

33719         /* KEY NOT INITIALIZED!!! THIS WON'T WORK!!! */
33720         if ((ptr = pthread_getspecific(key)) == NULL &&
33721             pthread_setspecific(key, NULL) != 0) {
33722             pthread_key_create(&key, NULL);
33723             ...
33724         }
33725     }
  
```

33726 **APPLICATION USAGE**

33727 None.

## 33728 RATIONALE

33729 **Destructor Functions**

33730 Normally, the value bound to a key on behalf of a particular thread is a pointer to storage  
33731 allocated dynamically on behalf of the calling thread. The destructor functions specified with  
33732 *pthread\_key\_create()* are intended to be used to free this storage when the thread exits. Thread  
33733 cancelation cleanup handlers cannot be used for this purpose because thread-specific data may  
33734 persist outside the lexical scope in which the cancelation cleanup handlers operate.

33735 If the value associated with a key needs to be updated during the lifetime of the thread, it may  
33736 be necessary to release the storage associated with the old value before the new value is bound.  
33737 Although the *pthread\_setspecific()* function could do this automatically, this feature is not needed  
33738 often enough to justify the added complexity. Instead, the programmer is responsible for freeing  
33739 the stale storage:

```
33740 pthread_getspecific(key, &old);  
33741 new = allocate();  
33742 destructor(old);  
33743 pthread_setspecific(key, new);
```

33744 **Note:** The above example could leak storage if run with asynchronous cancelation enabled. No such  
33745 problems occur in the default cancelation state if no cancelation points occur between the get  
33746 and set.

33747 There is no notion of a destructor-safe function. If an application does not call *pthread\_exit()*  
33748 from a signal handler, or if it blocks any signal whose handler may call *pthread\_exit()* while  
33749 calling async-unsafe functions, all functions may be safely called from destructors.

33750 **Non-Idempotent Data Key Creation**

33751 There were requests to make *pthread\_key\_create()* idempotent with respect to a given *key* address  
33752 parameter. This would allow applications to call *pthread\_key\_create()* multiple times for a given  
33753 *key* address and be guaranteed that only one key would be created. Doing so would require the  
33754 key value to be previously initialized (possibly at compile time) to a known null value and  
33755 would require that implicit mutual-exclusion be performed based on the address and contents of  
33756 the *key* parameter in order to guarantee that exactly one key would be created.

33757 Unfortunately, the implicit mutual-exclusion would not be limited to only *pthread\_key\_create()*.  
33758 On many implementations, implicit mutual-exclusion would also have to be performed by  
33759 *pthread\_getspecific()* and *pthread\_setspecific()* in order to guard against using incompletely stored  
33760 or not-yet-visible key values. This could significantly increase the cost of important operations,  
33761 particularly *pthread\_getspecific()*.

33762 Thus, this proposal was rejected. The *pthread\_key\_create()* function performs no implicit  
33763 synchronization. It is the responsibility of the programmer to ensure that it is called exactly once  
33764 per key before use of the key. Several straightforward mechanisms can already be used to  
33765 accomplish this, including calling explicit module initialization functions, using mutexes, and  
33766 using *pthread\_once()*. This places no significant burden on the programmer, introduces no  
33767 possibly confusing *ad hoc* implicit synchronization mechanism, and potentially allows  
33768 commonly used thread-specific data operations to be more efficient.

33769 **FUTURE DIRECTIONS**

33770 None.

33771 **SEE ALSO**

33772 *pthread\_getspecific()*, *pthread\_key\_delete()*, the Base Definitions volume of IEEE Std 1003.1-200x,  
33773 <pthread.h>

33774 **CHANGE HISTORY**

33775 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

33776 **Issue 6**

33777 The *pthread\_key\_create()* function is marked as part of the Threads option.

33778 IEEE PASC Interpretation 1003.1c #8 is applied, updating the DESCRIPTION.

33779 **NAME**

33780 pthread\_key\_delete — thread-specific data key deletion

33781 **SYNOPSIS**

33782 THR #include &lt;pthread.h&gt;

33783 int pthread\_key\_delete(pthread\_key\_t key);

33784

33785 **DESCRIPTION**

33786 The *pthread\_key\_delete()* function shall delete a thread-specific data key previously returned by  
33787 *pthread\_key\_create()*. The thread-specific data values associated with *key* need not be NULL at  
33788 the time *pthread\_key\_delete()* is called. It is the responsibility of the application to free any  
33789 application storage or perform any cleanup actions for data structures related to the deleted key  
33790 or associated thread-specific data in any threads; this cleanup can be done either before or after  
33791 *pthread\_key\_delete()* is called. Any attempt to use *key* following the call to *pthread\_key\_delete()*  
33792 results in undefined behavior.

33793 The *pthread\_key\_delete()* function shall be callable from within destructor functions. No  
33794 destructor functions shall be invoked by *pthread\_key\_delete()*. Any destructor function that may  
33795 have been associated with *key* shall no longer be called upon thread exit.

33796 **RETURN VALUE**

33797 If successful, the *pthread\_key\_delete()* function shall return zero; otherwise, an error number shall  
33798 be returned to indicate the error.

33799 **ERRORS**33800 The *pthread\_key\_delete()* function may fail if:33801 [EINVAL] The *key* value is invalid.33802 The *pthread\_key\_delete()* function shall not return an error code of [EINTR].33803 **EXAMPLES**

33804 None.

33805 **APPLICATION USAGE**

33806 None.

33807 **RATIONALE**

33808 A thread-specific data key deletion function has been included in order to allow the resources  
33809 associated with an unused thread-specific data key to be freed. Unused thread-specific data keys  
33810 can arise, among other scenarios, when a dynamically loaded module that allocated a key is  
33811 unloaded.

33812 Conforming applications are responsible for performing any cleanup actions needed for data  
33813 structures associated with the key to be deleted, including data referenced by thread-specific  
33814 data values. No such cleanup is done by *pthread\_key\_delete()*. In particular, destructor functions  
33815 are not called. There are several reasons for this division of responsibility:

- 33816 1. The associated destructor functions used to free thread-specific data at thread exit time are  
33817 only guaranteed to work correctly when called in the thread that allocated the thread-  
33818 specific data. (Destructors themselves may utilize thread-specific data.) Thus, they cannot  
33819 be used to free thread-specific data in other threads at key deletion time. Attempting to  
33820 have them called by other threads at key deletion time would require other threads to be  
33821 asynchronously interrupted. But since interrupted threads could be in an arbitrary state,  
33822 including holding locks necessary for the destructor to run, this approach would fail. In  
33823 general, there is no safe mechanism whereby an implementation could free thread-specific  
33824 data at key deletion time.

33825           2. Even if there were a means of safely freeing thread-specific data associated with keys to be  
33826 deleted, doing so would require that implementations be able to enumerate the threads  
33827 with non-NULL data and potentially keep them from creating more thread-specific data  
33828 while the key deletion is occurring. This special case could cause extra synchronization in  
33829 the normal case, which would otherwise be unnecessary.

33830           For an application to know that it is safe to delete a key, it has to know that all the threads that  
33831 might potentially ever use the key do not attempt to use it again. For example, it could know this  
33832 if all the client threads have called a cleanup procedure declaring that they are through with the  
33833 module that is being shut down, perhaps by zero'ing a reference count.

33834 **FUTURE DIRECTIONS**

33835           None.

33836 **SEE ALSO**

33837           *pthread\_key\_create()*, the Base Definitions volume of IEEE Std 1003.1-200x, <pthread.h>

33838 **CHANGE HISTORY**

33839           First released in Issue 5. Included for alignment with the POSIX Threads Extension.

33840 **Issue 6**

33841           The *pthread\_key\_delete()* function is marked as part of the Threads option.

33842 **NAME**

33843 pthread\_kill — send a signal to a thread

33844 **SYNOPSIS**

33845 THR #include &lt;signal.h&gt;

33846 int pthread\_kill(pthread\_t thread, int sig);

33847

33848 **DESCRIPTION**33849 The *pthread\_kill()* function shall request that a signal be delivered to the specified thread. |33850 As in *kill()*, if *sig* is zero, error checking shall be performed but no signal shall actually be sent. |33851 **RETURN VALUE**33852 Upon successful completion, the function shall return a value of zero. Otherwise, the function  
33853 shall return an error number. If the *pthread\_kill()* function fails, no signal shall be sent.33854 **ERRORS**33855 The *pthread\_kill()* function shall fail if:33856 [ESRCH] No thread could be found corresponding to that specified by the given thread  
33857 ID.33858 [EINVAL] The value of the *sig* argument is an invalid or unsupported signal number.33859 The *pthread\_kill()* function shall not return an error code of [EINTR].33860 **EXAMPLES**

33861 None.

33862 **APPLICATION USAGE**33863 The *pthread\_kill()* function provides a mechanism for asynchronously directing a signal at a  
33864 thread in the calling process. This could be used, for example, by one thread to affect broadcast  
33865 delivery of a signal to a set of threads.33866 Note that *pthread\_kill()* only causes the signal to be handled in the context of the given thread;  
33867 the signal action (termination or stopping) affects the process as a whole.33868 **RATIONALE**

33869 None.

33870 **FUTURE DIRECTIONS**

33871 None.

33872 **SEE ALSO**33873 *kill()*, *pthread\_self()*, *raise()*, the Base Definitions volume of IEEE Std 1003.1-200x, <signal.h>33874 **CHANGE HISTORY**

33875 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

33876 **Issue 6**33877 The *pthread\_kill()* function is marked as part of the Threads option.

33878 The APPLICATION USAGE section is added.

33879 **NAME**

33880 pthread\_mutex\_destroy, pthread\_mutex\_init — destroy and initialize a mutex

33881 **SYNOPSIS**

33882 THR #include &lt;pthread.h&gt;

33883 int pthread\_mutex\_destroy(pthread\_mutex\_t \*mutex);

33884 int pthread\_mutex\_init(pthread\_mutex\_t \*restrict mutex,

33885 const pthread\_mutexattr\_t \*restrict attr);

33886 pthread\_mutex\_t mutex = PTHREAD\_MUTEX\_INITIALIZER;

33887

33888 **DESCRIPTION**

33889 The *pthread\_mutex\_destroy()* function shall destroy the mutex object referenced by *mutex*; the  
 33890 mutex object becomes, in effect, uninitialized. An implementation may cause  
 33891 *pthread\_mutex\_destroy()* to set the object referenced by *mutex* to an invalid value. A destroyed  
 33892 mutex object can be reinitialized using *pthread\_mutex\_init()*; the results of otherwise referencing  
 33893 the object after it has been destroyed are undefined.

33894 It shall be safe to destroy an initialized mutex that is unlocked. Attempting to destroy a locked  
 33895 mutex results in undefined behavior.

33896 The *pthread\_mutex\_init()* function shall initialize the mutex referenced by *mutex* with attributes  
 33897 specified by *attr*. If *attr* is NULL, the default mutex attributes are used; the effect shall be the  
 33898 same as passing the address of a default mutex attributes object. Upon successful initialization,  
 33899 the state of the mutex becomes initialized and unlocked.

33900 Only *mutex* itself may be used for performing synchronization. The result of referring to copies  
 33901 of *mutex* in calls to *pthread\_mutex\_lock()*, *pthread\_mutex\_trylock()*, *pthread\_mutex\_unlock()*, and  
 33902 *pthread\_mutex\_destroy()* is undefined.

33903 Attempting to initialize an already initialized mutex results in undefined behavior.

33904 In cases where default mutex attributes are appropriate, the macro  
 33905 PTHREAD\_MUTEX\_INITIALIZER can be used to initialize mutexes that are statically allocated.  
 33906 The effect shall be equivalent to dynamic initialization by a call to *pthread\_mutex\_init()* with  
 33907 parameter *attr* specified as NULL, except that no error checks are performed.

33908 **RETURN VALUE**

33909 If successful, the *pthread\_mutex\_destroy()* and *pthread\_mutex\_init()* functions shall return zero;  
 33910 otherwise, an error number shall be returned to indicate the error.

33911 The [EBUSY] and [EINVAL] error checks, if implemented, act as if they were performed  
 33912 immediately at the beginning of processing for the function and shall cause an error return prior  
 33913 to modifying the state of the mutex specified by *mutex*.

33914 **ERRORS**

33915 The *pthread\_mutex\_destroy()* function may fail if:

33916 [EBUSY] The implementation has detected an attempt to destroy the object referenced  
 33917 by *mutex* while it is locked or referenced (for example, while being used in a  
 33918 *pthread\_cond\_timedwait()* or *pthread\_cond\_wait()*) by another thread.

33919 [EINVAL] The value specified by *mutex* is invalid.

33920 The *pthread\_mutex\_init()* function shall fail if:

33921 [EAGAIN] The system lacked the necessary resources (other than memory) to initialize  
 33922 another mutex.

- 33923 [ENOMEM] Insufficient memory exists to initialize the mutex.
- 33924 [EPERM] The caller does not have the privilege to perform the operation.
- 33925 The *pthread\_mutex\_init()* function may fail if:
- 33926 [EBUSY] The implementation has detected an attempt to reinitialize the object |  
33927 referenced by *mutex*, a previously initialized, but not yet destroyed, mutex.
- 33928 [EINVAL] The value specified by *attr* is invalid.
- 33929 These functions shall not return an error code of [EINTR].

**33930 EXAMPLES**

33931 None.

**33932 APPLICATION USAGE**

33933 None.

**33934 RATIONALE****33935 Alternate Implementations Possible**

33936 This volume of IEEE Std 1003.1-200x supports several alternative implementations of mutexes.  
33937 An implementation may store the lock directly in the object of type **pthread\_mutex\_t**.  
33938 Alternatively, an implementation may store the lock in the heap and merely store a pointer,  
33939 handle, or unique ID in the mutex object. Either implementation has advantages or may be  
33940 required on certain hardware configurations. So that portable code can be written that is  
33941 invariant to this choice, this volume of IEEE Std 1003.1-200x does not define assignment or  
33942 equality for this type, and it uses the term “initialize” to reinforce the (more restrictive) notion  
33943 that the lock may actually reside in the mutex object itself.

33944 Note that this precludes an over-specification of the type of the mutex or condition variable and  
33945 motivates the opacity of the type.

33946 An implementation is permitted, but not required, to have *pthread\_mutex\_destroy()* store an  
33947 illegal value into the mutex. This may help detect erroneous programs that try to lock (or  
33948 otherwise reference) a mutex that has already been destroyed.

**33949 Tradeoff Between Error Checks and Performance Supported**

33950 Many of the error checks were made optional in order to let implementations trade off  
33951 performance *versus* degree of error checking according to the needs of their specific applications  
33952 and execution environment. As a general rule, errors or conditions caused by the system (such as  
33953 insufficient memory) always need to be reported, but errors due to an erroneously coded  
33954 application (such as failing to provide adequate synchronization to prevent a mutex from being  
33955 deleted while in use) are made optional.

33956 A wide range of implementations is thus made possible. For example, an implementation  
33957 intended for application debugging may implement all of the error checks, but an  
33958 implementation running a single, provably correct application under very tight performance  
33959 constraints in an embedded computer might implement minimal checks. An implementation  
33960 might even be provided in two versions, similar to the options that compilers provide: a full-  
33961 checking, but slower version; and a limited-checking, but faster version. To forbid this  
33962 optionality would be a disservice to users.

33963 By carefully limiting the use of “undefined behavior” only to things that an erroneous (badly  
33964 coded) application might do, and by defining that resource-not-available errors are mandatory,  
33965 this volume of IEEE Std 1003.1-200x ensures that a fully-conforming application is portable

33966 across the full range of implementations, while not forcing all implementations to add overhead  
33967 to check for numerous things that a correct program never does.

### 33968 **Why No Limits Defined**

33969 Defining symbols for the maximum number of mutexes and condition variables was considered  
33970 but rejected because the number of these objects may change dynamically. Furthermore, many  
33971 implementations place these objects into application memory; thus, there is no explicit  
33972 maximum.

### 33973 **Static Initializers for Mutexes and Condition Variables**

33974 Providing for static initialization of statically allocated synchronization objects allows modules  
33975 with private static synchronization variables to avoid runtime initialization tests and overhead.  
33976 Furthermore, it simplifies the coding of self-initializing modules. Such modules are common in  
33977 C libraries, where for various reasons the design calls for self-initialization instead of requiring  
33978 an explicit module initialization function to be called. An example use of static initialization  
33979 follows.

33980 Without static initialization, a self-initializing routine *foo()* might look as follows:

```
33981 static pthread_once_t foo_once = PTHREAD_ONCE_INIT;
33982 static pthread_mutex_t foo_mutex;

33983 void foo_init()
33984 {
33985     pthread_mutex_init(&foo_mutex, NULL);
33986 }

33987 void foo()
33988 {
33989     pthread_once(&foo_once, foo_init);
33990     pthread_mutex_lock(&foo_mutex);
33991     /* Do work. */
33992     pthread_mutex_unlock(&foo_mutex);
33993 }
```

33994 With static initialization, the same routine could be coded as follows:

```
33995 static pthread_mutex_t foo_mutex = PTHREAD_MUTEX_INITIALIZER;

33996 void foo()
33997 {
33998     pthread_mutex_lock(&foo_mutex);
33999     /* Do work. */
34000     pthread_mutex_unlock(&foo_mutex);
34001 }
```

34002 Note that the static initialization both eliminates the need for the initialization test inside  
34003 *pthread\_once()* and the fetch of *&foo\_mutex* to learn the address to be passed to  
34004 *pthread\_mutex\_lock()* or *pthread\_mutex\_unlock()*.

34005 Thus, the C code written to initialize static objects is simpler on all systems and is also faster on a  
34006 large class of systems; those where the (entire) synchronization object can be stored in  
34007 application memory.

34008 Yet the locking performance question is likely to be raised for machines that require mutexes to  
34009 be allocated out of special memory. Such machines actually have to have mutexes and possibly

34010 condition variables contain pointers to the actual hardware locks. For static initialization to work  
34011 on such machines, *pthread\_mutex\_lock()* also has to test whether or not the pointer to the actual  
34012 lock has been allocated. If it has not, *pthread\_mutex\_lock()* has to initialize it before use. The  
34013 reservation of such resources can be made when the program is loaded, and hence return codes  
34014 have not been added to mutex locking and condition variable waiting to indicate failure to  
34015 complete initialization.

34016 This runtime test in *pthread\_mutex\_lock()* would at first seem to be extra work; an extra test is  
34017 required to see whether the pointer has been initialized. On most machines this would actually  
34018 be implemented as a fetch of the pointer, testing the pointer against zero, and then using the  
34019 pointer if it has already been initialized. While the test might seem to add extra work, the extra  
34020 effort of testing a register is usually negligible since no extra memory references are actually  
34021 done. As more and more machines provide caches, the real expenses are memory references, not  
34022 instructions executed.

34023 Alternatively, depending on the machine architecture, there are often ways to eliminate *all*  
34024 overhead in the most important case: on the lock operations that occur *after* the lock has been  
34025 initialized. This can be done by shifting more overhead to the less frequent operation:  
34026 initialization. Since out-of-line mutex allocation also means that an address has to be  
34027 dereferenced to find the actual lock, one technique that is widely applicable is to have static  
34028 initialization store a bogus value for that address; in particular, an address that causes a machine  
34029 fault to occur. When such a fault occurs upon the first attempt to lock such a mutex, validity  
34030 checks can be done, and then the correct address for the actual lock can be filled in. Subsequent  
34031 lock operations incur no extra overhead since they do not “fault”. This is merely one technique  
34032 that can be used to support static initialization, while not adversely affecting the performance of  
34033 lock acquisition. No doubt there are other techniques that are highly machine-dependent.

34034 The locking overhead for machines doing out-of-line mutex allocation is thus similar for  
34035 modules being implicitly initialized, where it is improved for those doing mutex allocation  
34036 entirely inline. The inline case is thus made much faster, and the out-of-line case is not  
34037 significantly worse.

34038 Besides the issue of locking performance for such machines, a concern is raised that it is possible  
34039 that threads would serialize contending for initialization locks when attempting to finish  
34040 initializing statically allocated mutexes. (Such finishing would typically involve taking an  
34041 internal lock, allocating a structure, storing a pointer to the structure in the mutex, and releasing  
34042 the internal lock.) First, many implementations would reduce such serialization by hashing on  
34043 the mutex address. Second, such serialization can only occur a bounded number of times. In  
34044 particular, it can happen at most as many times as there are statically allocated synchronization  
34045 objects. Dynamically allocated objects would still be initialized via *pthread\_mutex\_init()* or  
34046 *pthread\_cond\_init()*.

34047 Finally, if none of the above optimization techniques for out-of-line allocation yields sufficient  
34048 performance for an application on some implementation, the application can avoid static  
34049 initialization altogether by explicitly initializing all synchronization objects with the  
34050 corresponding *pthread\_\*\_init()* functions, which are supported by all implementations. An  
34051 implementation can also document the tradeoffs and advise which initialization technique is  
34052 more efficient for that particular implementation.

34053 **Destroying Mutexes**

34054 A mutex can be destroyed immediately after it is unlocked. For example, consider the following  
 34055 code:

```

34056 struct obj {
34057     pthread_mutex_t om;
34058     int refcnt;
34059     ...
34060 };

34061 obj_done(struct obj *op)
34062 {
34063     pthread_mutex_lock(&op->om);
34064     if (--op->refcnt == 0) {
34065         pthread_mutex_unlock(&op->om);
34066         (A) pthread_mutex_destroy(&op->om);
34067         (B) free(op);
34068     } else
34069         (C) pthread_mutex_unlock(&op->om);
34070 }

```

34071 In this case *obj* is reference counted and *obj\_done()* is called whenever a reference to the object is  
 34072 dropped. Implementations are required to allow an object to be destroyed and freed and  
 34073 potentially unmapped (for example, lines A and B) immediately after the object is unlocked (line  
 34074 C).

34075 **FUTURE DIRECTIONS**

34076 None.

34077 **SEE ALSO**

34078 *pthread\_mutex\_getprioceiling()*, *pthread\_mutex\_lock()*, *pthread\_mutex\_timedlock()*,  
 34079 *pthread\_mutexattr\_getpshared()*, the Base Definitions volume of IEEE Std 1003.1-200x,  
 34080 <pthread.h>

34081 **CHANGE HISTORY**

34082 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

34083 **Issue 6**

34084 The *pthread\_mutex\_destroy()* and *pthread\_mutex\_init()* functions are marked as part of the  
 34085 Threads option.

34086 The *pthread\_mutex\_timedlock()* function is added to the SEE ALSO section for alignment with  
 34087 IEEE Std 1003.1d-1999.

34088 IEEE PASC Interpretation 1003.1c #34 is applied, updating the DESCRIPTION.

34089 The **restrict** keyword is added to the *pthread\_mutex\_init()* prototype for alignment with the  
 34090 ISO/IEC 9899:1999 standard.

## 34091 NAME

34092 pthread\_mutex\_getprioceiling, pthread\_mutex\_setprioceiling — get and set the priority ceiling  
 34093 of a mutex (**REALTIME THREADS**)

## 34094 SYNOPSIS

```
34095 THR TPP #include <pthread.h>
34096
34096 int pthread_mutex_getprioceiling(const pthread_mutex_t *restrict mutex,
34097 int *restrict prioceiling);
34098 int pthread_mutex_setprioceiling(pthread_mutex_t *restrict mutex,
34099 int prioceiling, int *restrict old_ceiling);
34100
```

## 34101 DESCRIPTION

34102 The *pthread\_mutex\_getprioceiling()* function shall return the current priority ceiling of the mutex.

34103 The *pthread\_mutex\_setprioceiling()* function shall either lock the mutex if it is unlocked, or block |  
 34104 until it can successfully lock the mutex, then it shall change the mutex's priority ceiling and |  
 34105 release the mutex. When the change is successful, the previous value of the priority ceiling shall |  
 34106 be returned in *old\_ceiling*. The process of locking the mutex need not adhere to the priority |  
 34107 protect protocol.

34108 If the *pthread\_mutex\_setprioceiling()* function fails, the mutex priority ceiling shall not be  
 34109 changed.

## 34110 RETURN VALUE

34111 If successful, the *pthread\_mutex\_getprioceiling()* and *pthread\_mutex\_setprioceiling()* functions shall  
 34112 return zero; otherwise, an error number shall be returned to indicate the error.

## 34113 ERRORS

34114 The *pthread\_mutex\_getprioceiling()* and *pthread\_mutex\_setprioceiling()* functions may fail if:

- 34115 [EINVAL] The priority requested by *prioceiling* is out of range.
- 34116 [EINVAL] The value specified by *mutex* does not refer to a currently existing mutex.
- 34117 [EPERM] The caller does not have the privilege to perform the operation.
- 34118 These functions shall not return an error code of [EINTR].

## 34119 EXAMPLES

34120 None.

## 34121 APPLICATION USAGE

34122 None.

## 34123 RATIONALE

34124 None.

## 34125 FUTURE DIRECTIONS

34126 None.

## 34127 SEE ALSO

34128 *pthread\_mutex\_destroy()*, *pthread\_mutex\_lock()*, *pthread\_mutex\_timedlock()*, the Base Definitions  
 34129 volume of IEEE Std 1003.1-200x, <pthread.h>

## 34130 CHANGE HISTORY

- 34131 First released in Issue 5. Included for alignment with the POSIX Threads Extension.
- 34132 Marked as part of the Realtime Threads Feature Group.

34133 **Issue 6**

34134 The *pthread\_mutex\_getprioceiling()* and *pthread\_mutex\_setprioceiling()* functions are marked as  
34135 part of the Threads and Thread Priority Protection options.

34136 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
34137 implementation does not support the Thread Priority Protection option.

34138 The [ENOSYS] error denoting non-support of the priority ceiling protocol for mutexes has been  
34139 removed. This is since if the implementation provides the functions (regardless of whether  
34140 `_POSIX_PTHREAD_PRIO_PROTECT` is defined), they must function as in the DESCRIPTION  
34141 and therefore the priority ceiling protocol for mutexes is supported.

34142 The *pthread\_mutex\_timedlock()* function is added to the SEE ALSO section for alignment with  
34143 IEEE Std 1003.1d-1999.

34144 The **restrict** keyword is added to the *pthread\_mutex\_getprioceiling()* and  
34145 *pthread\_mutex\_setprioceiling()* prototypes for alignment with the ISO/IEC 9899:1999 standard.

34146 **NAME**

34147 pthread\_mutex\_init — initialize a mutex

34148 **SYNOPSIS**

34149 THR #include <pthread.h>

34150 int pthread\_mutex\_init(pthread\_mutex\_t \*restrict mutex,

34151 const pthread\_mutexattr\_t \*restrict attr);

34152 pthread\_mutex\_t mutex = PTHREAD\_MUTEX\_INITIALIZER;

34153

34154 **DESCRIPTION**

34155 Refer to *pthread\_mutex\_destroy()*.

34156 **NAME**

34157 pthread\_mutex\_lock, pthread\_mutex\_trylock, pthread\_mutex\_unlock — lock and unlock a  
34158 mutex

34159 **SYNOPSIS**

34160 THR #include <pthread.h>

34161 int pthread\_mutex\_lock(pthread\_mutex\_t \*mutex);

34162 int pthread\_mutex\_trylock(pthread\_mutex\_t \*mutex);

34163 int pthread\_mutex\_unlock(pthread\_mutex\_t \*mutex);

34164

34165 **DESCRIPTION**

34166 The mutex object referenced by *mutex* shall be locked by calling *pthread\_mutex\_lock()*. If the  
34167 mutex is already locked, the calling thread shall block until the mutex becomes available. This  
34168 operation shall return with the mutex object referenced by *mutex* in the locked state with the  
34169 calling thread as its owner.

34170 XSI If the mutex type is PTHREAD\_MUTEX\_NORMAL, deadlock detection shall not be provided.  
34171 Attempting to relock the mutex causes deadlock. If a thread attempts to unlock a mutex that it  
34172 has not locked or a mutex which is unlocked, undefined behavior results.

34173 If the mutex type is PTHREAD\_MUTEX\_ERRORCHECK, then error checking shall be provided.  
34174 If a thread attempts to relock a mutex that it has already locked, an error shall be returned. If a  
34175 thread attempts to unlock a mutex that it has not locked or a mutex which is unlocked, an error  
34176 shall be returned.

34177 If the mutex type is PTHREAD\_MUTEX\_RECURSIVE, then the mutex shall maintain the  
34178 concept of a lock count. When a thread successfully acquires a mutex for the first time, the lock  
34179 count shall be set to one. Every time a thread relocks this mutex, the lock count shall be  
34180 incremented by one. Each time the thread unlocks the mutex, the lock count shall be  
34181 decremented by one. When the lock count reaches zero, the mutex shall become available for  
34182 other threads to acquire. If a thread attempts to unlock a mutex that it has not locked or a mutex  
34183 which is unlocked, an error shall be returned.

34184 If the mutex type is PTHREAD\_MUTEX\_DEFAULT, attempting to recursively lock the mutex  
34185 results in undefined behavior. Attempting to unlock the mutex if it was not locked by the calling  
34186 thread results in undefined behavior. Attempting to unlock the mutex if it is not locked results in  
34187 undefined behavior.

34188 The *pthread\_mutex\_trylock()* function shall be equivalent to *pthread\_mutex\_lock()*, except that if  
34189 the mutex object referenced by *mutex* is currently locked (by any thread, including the current  
34190 thread), the call shall return immediately. If the mutex type is PTHREAD\_MUTEX\_RECURSIVE  
34191 and the mutex is currently owned by the calling thread, the mutex lock count shall be  
34192 incremented by one and the *pthread\_mutex\_trylock()* function shall immediately return success.

34193 XSI The *pthread\_mutex\_unlock()* function shall release the mutex object referenced by *mutex*. The  
34194 manner in which a mutex is released is dependent upon the mutex's type attribute. If there are  
34195 threads blocked on the mutex object referenced by *mutex* when *pthread\_mutex\_unlock()* is called,  
34196 resulting in the mutex becoming available, the scheduling policy shall determine which thread  
34197 shall acquire the mutex.

34198 XSI (In the case of PTHREAD\_MUTEX\_RECURSIVE mutexes, the mutex shall become available  
34199 when the count reaches zero and the calling thread no longer has any locks on this mutex).

34200 If a signal is delivered to a thread waiting for a mutex, upon return from the signal handler the  
34201 thread shall resume waiting for the mutex as if it was not interrupted.

34202 **RETURN VALUE**

34203 If successful, the *pthread\_mutex\_lock()* and *pthread\_mutex\_unlock()* functions shall return zero;  
34204 otherwise, an error number shall be returned to indicate the error.

34205 The *pthread\_mutex\_trylock()* function shall return zero if a lock on the mutex object referenced by  
34206 *mutex* is acquired. Otherwise, an error number is returned to indicate the error.

34207 **ERRORS**

34208 The *pthread\_mutex\_lock()* and *pthread\_mutex\_trylock()* functions shall fail if:

34209 [EINVAL] The *mutex* was created with the protocol attribute having the value  
34210 PTHREAD\_PRIO\_PROTECT and the calling thread's priority is higher than  
34211 the mutex's current priority ceiling.

34212 The *pthread\_mutex\_trylock()* function shall fail if:

34213 [EBUSY] The *mutex* could not be acquired because it was already locked.

34214 The *pthread\_mutex\_lock()*, *pthread\_mutex\_trylock()*, and *pthread\_mutex\_unlock()* functions may  
34215 fail if:

34216 [EINVAL] The value specified by *mutex* does not refer to an initialized mutex object.

34217 XSI [EAGAIN] The mutex could not be acquired because the maximum number of recursive  
34218 locks for *mutex* has been exceeded.

34219 The *pthread\_mutex\_lock()* function may fail if:

34220 [EDEADLK] The current thread already owns the mutex.

34221 The *pthread\_mutex\_unlock()* function may fail if:

34222 [EPERM] The current thread does not own the mutex.

34223 These functions shall not return an error code of [EINTR].

34224 **EXAMPLES**

34225 None.

34226 **APPLICATION USAGE**

34227 None.

34228 **RATIONALE**

34229 Mutex objects are intended to serve as a low-level primitive from which other thread  
34230 synchronization functions can be built. As such, the implementation of mutexes should be as  
34231 efficient as possible, and this has ramifications on the features available at the interface.

34232 The mutex functions and the particular default settings of the mutex attributes have been  
34233 motivated by the desire to not preclude fast, inlined implementations of mutex locking and  
34234 unlocking.

34235 For example, deadlocking on a double-lock is explicitly allowed behavior in order to avoid  
34236 requiring more overhead in the basic mechanism than is absolutely necessary. (More "friendly"  
34237 mutexes that detect deadlock or that allow multiple locking by the same thread are easily  
34238 constructed by the user via the other mechanisms provided. For example, *pthread\_self()* can be  
34239 used to record mutex ownership.) Implementations might also choose to provide such extended  
34240 features as options via special mutex attributes.

34241 Since most attributes only need to be checked when a thread is going to be blocked, the use of  
34242 attributes does not slow the (common) mutex-locking case.

34243 Likewise, while being able to extract the thread ID of the owner of a mutex might be desirable, it  
34244 would require storing the current thread ID when each mutex is locked, and this could incur  
34245 unacceptable levels of overhead. Similar arguments apply to a *mutex\_tryunlock* operation.

34246 **FUTURE DIRECTIONS**

34247 None.

34248 **SEE ALSO**

34249 *pthread\_mutex\_destroy()*, *pthread\_mutex\_timedlock()*, the Base Definitions volume of  
34250 IEEE Std 1003.1-200x, <pthread.h>

34251 **CHANGE HISTORY**

34252 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

34253 **Issue 6**

34254 The *pthread\_mutex\_lock()*, *pthread\_mutex\_trylock()*, and *pthread\_mutex\_unlock()* functions are  
34255 marked as part of the Threads option.

34256 The following new requirements on POSIX implementations derive from alignment with the  
34257 Single UNIX Specification:

- 34258 • The behavior when attempting to relock a mutex is defined.

34259 The *pthread\_mutex\_timedlock()* function is added to the SEE ALSO section for alignment with  
34260 IEEE Std 1003.1d-1999.

34261 **NAME**

34262 pthread\_mutex\_setprioceiling — change the priority ceiling of a mutex (**REALTIME**  
34263 **THREADS**)

34264 **SYNOPSIS**

```
34265 THR TPP #include <pthread.h>
```

```
34266 int pthread_mutex_setprioceiling(pthread_mutex_t *restrict mutex,  
34267 int prioceiling, int *restrict old_ceiling);
```

34268

34269 **DESCRIPTION**

34270 Refer to *pthread\_mutex\_getprioceiling()*.

34271 **NAME**34272 pthread\_mutex\_timedlock — lock a mutex (**ADVANCED REALTIME**)34273 **SYNOPSIS**

34274 THR TMO #include &lt;pthread.h&gt;

34275 #include &lt;time.h&gt;

34276 int pthread\_mutex\_timedlock(pthread\_mutex\_t \*restrict mutex,

34277 const struct timespec \*restrict abs\_timeout);

34278

34279 **DESCRIPTION**

34280 The *pthread\_mutex\_timedlock()* function shall lock the mutex object referenced by *mutex*. If the  
 34281 mutex is already locked, the calling thread shall block until the mutex becomes available as in  
 34282 the *pthread\_mutex\_lock()* function. If the mutex cannot be locked without waiting for another  
 34283 thread to unlock the mutex, this wait shall be terminated when the specified timeout expires.

34284 The timeout shall expire when the absolute time specified by *abs\_timeout* passes, as measured by  
 34285 the clock on which timeouts are based (that is, when the value of that clock equals or exceeds  
 34286 *abs\_timeout*), or if the absolute time specified by *abs\_timeout* has already been passed at the time  
 34287 of the call.

34288 TMR If the Timers option is supported, the timeout shall be based on the `CLOCK_REALTIME` clock; if  
 34289 the Timers option is not supported, the timeout shall be based on the system clock as returned  
 34290 by the *time()* function.

34291 The resolution of the timeout shall be the resolution of the clock on which it is based. The  
 34292 `timespec` data type is defined in the `<time.h>` header.

34293 Under no circumstance shall the function fail with a timeout if the mutex can be locked  
 34294 immediately. The validity of the *abs\_timeout* parameter need not be checked if the mutex can be  
 34295 locked immediately.

34296 As a consequence of the priority inheritance rules (for mutexes initialized with the  
 34297 `PRIO_INHERIT` protocol), if a timed mutex wait is terminated because its timeout expires, the  
 34298 priority of the owner of the mutex shall be adjusted as necessary to reflect the fact that this  
 34299 thread is no longer among the threads waiting for the mutex.

34300 **RETURN VALUE**

34301 If successful, the *pthread\_mutex\_timedlock()* function shall return zero; otherwise, an error  
 34302 number shall be returned to indicate the error.

34303 **ERRORS**

34304 The *pthread\_mutex\_timedlock()* function shall fail if:

34305 [EINVAL] The mutex was created with the protocol attribute having the value  
 34306 `PTHREAD_PRIO_PROTECT` and the calling thread's priority is higher than  
 34307 the mutex' current priority ceiling.

34308 [EINVAL] The process or thread would have blocked, and the *abs\_timeout* parameter  
 34309 specified a nanoseconds field value less than zero or greater than or equal to  
 34310 1 000 million.

34311 [ETIMEDOUT] The mutex could not be locked before the specified timeout expired.

34312 The *pthread\_mutex\_timedlock()* function may fail if:

34313 [EINVAL] The value specified by *mutex* does not refer to an initialized mutex object.

34314 XSI [EAGAIN] The mutex could not be acquired because the maximum number of recursive  
34315 locks for mutex has been exceeded.

34316 [EDEADLK] The current thread already owns the mutex.

34317 This function shall not return an error code of [EINTR].

## 34318 EXAMPLES

34319 None.

## 34320 APPLICATION USAGE

34321 The *pthread\_mutex\_timedlock()* function is part of the Threads and Timeouts options and need  
34322 not be provided on all implementations.

## 34323 RATIONALE

34324 None.

## 34325 FUTURE DIRECTIONS

34326 None.

## 34327 SEE ALSO

34328 *pthread\_mutex\_destroy()*, *pthread\_mutex\_lock()*, *pthread\_mutex\_trylock()*, *time()*, the Base  
34329 Definitions volume of IEEE Std 1003.1-200x, <pthread.h>, <time.h>

## 34330 CHANGE HISTORY

34331 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

34332 **NAME**

34333 pthread\_mutex\_trylock, pthread\_mutex\_unlock — lock and unlock a mutex

34334 **SYNOPSIS**

34335 THR #include &lt;pthread.h&gt;

34336 int pthread\_mutex\_trylock(pthread\_mutex\_t \*mutex);

34337 int pthread\_mutex\_unlock(pthread\_mutex\_t \*mutex);

34338

34339 **DESCRIPTION**34340 Refer to *pthread\_mutex\_lock()*.

## 34341 NAME

34342 pthread\_mutexattr\_destroy, pthread\_mutexattr\_init — destroy and initialize mutex attributes  
34343 object

## 34344 SYNOPSIS

```
34345 THR #include <pthread.h>
```

```
34346 int pthread_mutexattr_destroy(pthread_mutexattr_t *attr);
```

```
34347 int pthread_mutexattr_init(pthread_mutexattr_t *attr);
```

34348

## 34349 DESCRIPTION

34350 The *pthread\_mutexattr\_destroy()* function shall destroy a mutex attributes object; the object  
34351 becomes, in effect, uninitialized. An implementation may cause *pthread\_mutexattr\_destroy()* to  
34352 set the object referenced by *attr* to an invalid value. A destroyed *attr* attributes object can be  
34353 reinitialized using *pthread\_mutexattr\_init()*; the results of otherwise referencing the object after it  
34354 has been destroyed are undefined. |

34355 The *pthread\_mutexattr\_init()* function shall initialize a mutex attributes object *attr* with the  
34356 default value for all of the attributes defined by the implementation.

34357 Results are undefined if *pthread\_mutexattr\_init()* is called specifying an already initialized *attr* |  
34358 attributes object. |

34359 After a mutex attributes object has been used to initialize one or more mutexes, any function  
34360 affecting the attributes object (including destruction) shall not affect any previously initialized |  
34361 mutexes. |

## 34362 RETURN VALUE

34363 Upon successful completion, *pthread\_mutexattr\_destroy()* and *pthread\_mutexattr\_init()* shall  
34364 return zero; otherwise, an error number shall be returned to indicate the error.

## 34365 ERRORS

34366 The *pthread\_mutexattr\_destroy()* function may fail if:

34367 [EINVAL] The value specified by *attr* is invalid.

34368 The *pthread\_mutexattr\_init()* function shall fail if:

34369 [ENOMEM] Insufficient memory exists to initialize the mutex attributes object.

34370 These functions shall not return an error code of [EINTR].

## 34371 EXAMPLES

34372 None.

## 34373 APPLICATION USAGE

34374 None.

## 34375 RATIONALE

34376 See *pthread\_attr\_init()* for a general explanation of attributes. Attributes objects allow  
34377 implementations to experiment with useful extensions and permit extension of this volume of  
34378 IEEE Std 1003.1-200x without changing the existing functions. Thus, they provide for future  
34379 extensibility of this volume of IEEE Std 1003.1-200x and reduce the temptation to standardize  
34380 prematurely on semantics that are not yet widely implemented or understood.

34381 Examples of possible additional mutex attributes that have been discussed are *spin\_only*,  
34382 *limited\_spin*, *no\_spin*, *recursive*, and *metered*. (To explain what the latter attributes might mean:  
34383 recursive mutexes would allow for multiple re-locking by the current owner; metered mutexes  
34384 would transparently keep records of queue length, wait time, and so on.) Since there is not yet

34385 wide agreement on the usefulness of these resulting from shared implementation and usage  
 34386 experience, they are not yet specified in this volume of IEEE Std 1003.1-200x. Mutex attributes  
 34387 objects, however, make it possible to test out these concepts for possible standardization at a  
 34388 later time.

#### 34389 **Mutex Attributes and Performance**

34390 Care has been taken to ensure that the default values of the mutex attributes have been defined  
 34391 such that mutexes initialized with the defaults have simple enough semantics so that the locking  
 34392 and unlocking can be done with the equivalent of a test-and-set instruction (plus possibly a few  
 34393 other basic instructions).

34394 There is at least one implementation method that can be used to reduce the cost of testing at  
 34395 lock-time if a mutex has non-default attributes. One such method that an implementation can  
 34396 employ (and this can be made fully transparent to fully conforming POSIX applications) is to  
 34397 secretly pre-lock any mutexes that are initialized to non-default attributes. Any later attempt to  
 34398 lock such a mutex causes the implementation to branch to the “slow path” as if the mutex were  
 34399 unavailable; then, on the slow path, the implementation can do the “real work” to lock a non-  
 34400 default mutex. The underlying unlock operation is more complicated since the implementation  
 34401 never really wants to release the pre-lock on this kind of mutex. This illustrates that, depending  
 34402 on the hardware, there may be certain optimizations that can be used so that whatever mutex  
 34403 attributes are considered “most frequently used” can be processed most efficiently.

#### 34404 **Process Shared Memory and Synchronization**

34405 The existence of memory mapping functions in this volume of IEEE Std 1003.1-200x leads to the  
 34406 possibility that an application may allocate the synchronization objects from this section in  
 34407 memory that is accessed by multiple processes (and therefore, by threads of multiple processes).

34408 In order to permit such usage, while at the same time keeping the usual case (that is, usage  
 34409 within a single process) efficient, a process-shared option has been defined.

34410 If an implementation supports the `_POSIX_THREAD_PROCESS_SHARED` option, then the  
 34411 *process-shared* attribute can be used to indicate that mutexes or condition variables may be  
 34412 accessed by threads of multiple processes.

34413 The default setting of `PTHREAD_PROCESS_PRIVATE` has been chosen for the *process-shared*  
 34414 attribute so that the most efficient forms of these synchronization objects are created by default.

34415 Synchronization variables that are initialized with the `PTHREAD_PROCESS_PRIVATE` *process-*  
 34416 *shared* attribute may only be operated on by threads in the process that initialized them.  
 34417 Synchronization variables that are initialized with the `PTHREAD_PROCESS_SHARED` *process-*  
 34418 *shared* attribute may be operated on by any thread in any process that has access to it. In  
 34419 particular, these processes may exist beyond the lifetime of the initializing process. For example,  
 34420 the following code implements a simple counting semaphore in a mapped file that may be used  
 34421 by many processes.

```
34422 /* sem.h */
34423 struct semaphore {
34424     pthread_mutex_t lock;
34425     pthread_cond_t nonzero;
34426     unsigned count;
34427 };
34428 typedef struct semaphore semaphore_t;
34429 semaphore_t *semaphore_create(char *semaphore_name);
34430 semaphore_t *semaphore_open(char *semaphore_name);
```

```
34431 void semaphore_post(semaphore_t *semap);
34432 void semaphore_wait(semaphore_t *semap);
34433 void semaphore_close(semaphore_t *semap);

34434 /* sem.c */
34435 #include <sys/types.h>
34436 #include <sys/stat.h>
34437 #include <sys/mman.h>
34438 #include <fcntl.h>
34439 #include <pthread.h>
34440 #include "sem.h"

34441 semaphore_t *
34442 semaphore_create(char *semaphore_name)
34443 {
34444     int fd;
34445     semaphore_t *semap;
34446     pthread_mutexattr_t psharedm;
34447     pthread_condattr_t psharedc;

34448     fd = open(semaphore_name, O_RDWR | O_CREAT | O_EXCL, 0666);
34449     if (fd < 0)
34450         return (NULL);
34451     (void) ftruncate(fd, sizeof(semaphore_t));
34452     (void) pthread_mutexattr_init(&psharedm);
34453     (void) pthread_mutexattr_setpshared(&psharedm,
34454         PTHREAD_PROCESS_SHARED);
34455     (void) pthread_condattr_init(&psharedc);
34456     (void) pthread_condattr_setpshared(&psharedc,
34457         PTHREAD_PROCESS_SHARED);
34458     semap = (semaphore_t *) mmap(NULL, sizeof(semaphore_t),
34459         PROT_READ | PROT_WRITE, MAP_SHARED,
34460         fd, 0);
34461     close (fd);
34462     (void) pthread_mutex_init(&semap->lock, &psharedm);
34463     (void) pthread_cond_init(&semap->nonzero, &psharedc);
34464     semap->count = 0;
34465     return (semap);
34466 }

34467 semaphore_t *
34468 semaphore_open(char *semaphore_name)
34469 {
34470     int fd;
34471     semaphore_t *semap;

34472     fd = open(semaphore_name, O_RDWR, 0666);
34473     if (fd < 0)
34474         return (NULL);
34475     semap = (semaphore_t *) mmap(NULL, sizeof(semaphore_t),
34476         PROT_READ | PROT_WRITE, MAP_SHARED,
34477         fd, 0);
34478     close (fd);
34479     return (semap);
34480 }
```

```

34481 void
34482 semaphore_post(semaphore_t *semap)
34483 {
34484     pthread_mutex_lock(&semap->lock);
34485     if (semap->count == 0)
34486         pthread_cond_signal(&semap->nonzero);
34487     semap->count++;
34488     pthread_mutex_unlock(&semap->lock);
34489 }
34490 void
34491 semaphore_wait(semaphore_t *semap)
34492 {
34493     pthread_mutex_lock(&semap->lock);
34494     while (semap->count == 0)
34495         pthread_cond_wait(&semap->nonzero, &semap->lock);
34496     semap->count--;
34497     pthread_mutex_unlock(&semap->lock);
34498 }
34499 void
34500 semaphore_close(semaphore_t *semap)
34501 {
34502     munmap((void *) semap, sizeof(semaphore_t));
34503 }

```

34504 The following code is for three separate processes that create, post, and wait on a semaphore in  
34505 the file **/tmp/semaphore**. Once the file is created, the post and wait programs increment and  
34506 decrement the counting semaphore (waiting and waking as required) even though they did not  
34507 initialize the semaphore.

```

34508 /* create.c */
34509 #include "pthread.h"
34510 #include "sem.h"
34511 int
34512 main()
34513 {
34514     semaphore_t *semap;
34515     semap = semaphore_create("/tmp/semaphore");
34516     if (semap == NULL)
34517         exit(1);
34518     semaphore_close(semap);
34519     return (0);
34520 }
34521 /* post */
34522 #include "pthread.h"
34523 #include "sem.h"
34524 int
34525 main()
34526 {
34527     semaphore_t *semap;

```

```
34528     semaphore = semaphore_open("/tmp/semaphore");
34529     if (semap == NULL)
34530         exit(1);
34531     semaphore_post(semap);
34532     semaphore_close(semap);
34533     return (0);
34534 }

34535 /* wait */
34536 #include "pthread.h"
34537 #include "sem.h"

34538 int
34539 main()
34540 {
34541     semaphore_t *semap;

34542     semap = semaphore_open("/tmp/semaphore");
34543     if (semap == NULL)
34544         exit(1);
34545     semaphore_wait(semap);
34546     semaphore_close(semap);
34547     return (0);
34548 }
```

**34549 FUTURE DIRECTIONS**

34550 None.

**34551 SEE ALSO**

34552 *pthread\_cond\_destroy()*, *pthread\_create()*, *pthread\_mutex\_destroy()*, *pthread\_mutexattr\_destroy()*, the  
34553 Base Definitions volume of IEEE Std 1003.1-200x, <pthread.h>

**34554 CHANGE HISTORY**

34555 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

**34556 Issue 6**

34557 The *pthread\_mutexattr\_destroy()* and *pthread\_mutexattr\_init()* functions are marked as part of the  
34558 Threads option.

34559 IEEE PASC Interpretation 1003.1c #27 is applied, updating the ERRORS section.

34560 **NAME**

34561 pthread\_mutexattr\_getprioceiling, pthread\_mutexattr\_setprioceiling — get and set prioceiling  
 34562 attribute of mutex attributes object (**REALTIME THREADS**)

34563 **SYNOPSIS**

```
34564 THR TPP #include <pthread.h>
```

```
34565 int pthread_mutexattr_getprioceiling(  
34566     const pthread_mutexattr_t *restrict attr,  
34567     int *restrict prioceiling);  
34568 int pthread_mutexattr_setprioceiling(pthread_mutexattr_t *attr,  
34569     int prioceiling);  
34570
```

34571 **DESCRIPTION**

34572 The *pthread\_mutexattr\_getprioceiling()* and *pthread\_mutexattr\_setprioceiling()* functions,  
 34573 respectively, shall get and set the priority ceiling attribute of a mutex attributes object pointed to  
 34574 by *attr* which was previously created by the function *pthread\_mutexattr\_init()*.

34575 The *prioceiling* attribute contains the priority ceiling of initialized mutexes. The values of  
 34576 *prioceiling* are within the maximum range of priorities defined by SCHED\_FIFO.

34577 The *prioceiling* attribute defines the priority ceiling of initialized mutexes, which is the minimum  
 34578 priority level at which the critical section guarded by the mutex is executed. In order to avoid  
 34579 priority inversion, the priority ceiling of the mutex shall be set to a priority higher than or equal  
 34580 to the highest priority of all the threads that may lock that mutex. The values of *prioceiling* are  
 34581 within the maximum range of priorities defined under the SCHED\_FIFO scheduling policy.

34582 **RETURN VALUE**

34583 Upon successful completion, the *pthread\_mutexattr\_getprioceiling()* and  
 34584 *pthread\_mutexattr\_setprioceiling()* functions shall return zero; otherwise, an error number shall be  
 34585 returned to indicate the error.

34586 **ERRORS**

34587 The *pthread\_mutexattr\_getprioceiling()* and *pthread\_mutexattr\_setprioceiling()* functions may fail if:

34588 [EINVAL] The value specified by *attr* or *prioceiling* is invalid.

34589 [EPERM] The caller does not have the privilege to perform the operation.

34590 These functions shall not return an error code of [EINTR].

34591 **EXAMPLES**

34592 None.

34593 **APPLICATION USAGE**

34594 None.

34595 **RATIONALE**

34596 None.

34597 **FUTURE DIRECTIONS**

34598 None.

34599 **SEE ALSO**

34600 *pthread\_cond\_destroy()*, *pthread\_create()*, *pthread\_mutex\_destroy()*, the Base Definitions volume of  
 34601 IEEE Std 1003.1-200x, <pthread.h>

## 34602 CHANGE HISTORY

34603 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

34604 Marked as part of the Realtime Threads Feature Group.

## 34605 Issue 6

34606 The `pthread_mutexattr_getprioceiling()` and `pthread_mutexattr_setprioceiling()` functions are  
34607 marked as part of the Threads and Thread Priority Protection options.

34608 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
34609 implementation does not support the Thread Priority Protection option.

34610 The [ENOTSUP] error condition has been removed since these functions do not have a *protocol*  
34611 argument.

34612 The `restrict` keyword is added to the `pthread_mutexattr_getprioceiling()` prototype for alignment  
34613 with the ISO/IEC 9899:1999 standard.

## 34614 NAME

34615 pthread\_mutexattr\_getprotocol, pthread\_mutexattr\_setprotocol — get and set protocol attribute  
 34616 of mutex attributes object (**REALTIME THREADS**)

## 34617 SYNOPSIS

```
34618 THR #include <pthread.h>
34619 TPP|TPI int pthread_mutexattr_getprotocol(
34620         const pthread_mutexattr_t *restrict attr,
34621         int *restrict protocol);
34622 int pthread_mutexattr_setprotocol(pthread_mutexattr_t *attr,
34623         int protocol);
34624
```

## 34625 DESCRIPTION

34626 The *pthread\_mutexattr\_getprotocol()* and *pthread\_mutexattr\_setprotocol()* functions, respectively,  
 34627 shall get and set the protocol attribute of a mutex attributes object pointed to by *attr* which was  
 34628 previously created by the function *pthread\_mutexattr\_init()*.

34629 The *protocol* attribute defines the protocol to be followed in utilizing mutexes. The value of  
 34630 *protocol* may be one of:

```
34631 PTHREAD_PRIO_NONE
34632 TPI PTHREAD_PRIO_INHERIT
34633 TPP PTHREAD_PRIO_PROTECT
34634
```

34635 which are defined in the **<pthread.h>** header. |

34636 When a thread owns a mutex with the PTHREAD\_PRIO\_NONE *protocol* attribute, its priority |  
 34637 and scheduling shall not be affected by its mutex ownership. |

34638 TPI When a thread is blocking higher priority threads because of owning one or more mutexes with  
 34639 the PTHREAD\_PRIO\_INHERIT protocol attribute, it shall execute at the higher of its priority or |  
 34640 the priority of the highest priority thread waiting on any of the mutexes owned by this thread |  
 34641 and initialized with this protocol. |

34642 TPP When a thread owns one or more mutexes initialized with the PTHREAD\_PRIO\_PROTECT |  
 34643 protocol, it shall execute at the higher of its priority or the highest of the priority ceilings of all |  
 34644 the mutexes owned by this thread and initialized with this attribute, regardless of whether other |  
 34645 threads are blocked on any of these mutexes or not. |

34646 While a thread is holding a mutex which has been initialized with the  
 34647 PTHREAD\_PRIO\_INHERIT or PTHREAD\_PRIO\_PROTECT protocol attributes, it shall not be  
 34648 subject to being moved to the tail of the scheduling queue at its priority in the event that its  
 34649 original priority is changed, such as by a call to *sched\_setparam()*. Likewise, when a thread  
 34650 unlocks a mutex that has been initialized with the PTHREAD\_PRIO\_INHERIT or  
 34651 PTHREAD\_PRIO\_PROTECT protocol attributes, it shall not be subject to being moved to the tail  
 34652 of the scheduling queue at its priority in the event that its original priority is changed.

34653 If a thread simultaneously owns several mutexes initialized with different protocols, it shall  
 34654 execute at the highest of the priorities that it would have obtained by each of these protocols.

34655 TPI When a thread makes a call to *pthread\_mutex\_lock()*, the mutex was initialized with the protocol  
 34656 attribute having the value PTHREAD\_PRIO\_INHERIT, when the calling thread is blocked  
 34657 because the mutex is owned by another thread, that owner thread shall inherit the priority level  
 34658 of the calling thread as long as it continues to own the mutex. The implementation shall update  
 34659 its execution priority to the maximum of its assigned priority and all its inherited priorities.

34660 Furthermore, if this owner thread itself becomes blocked on another mutex, the same priority  
34661 inheritance effect shall be propagated to this other owner thread, in a recursive manner.

#### 34662 RETURN VALUE

34663 Upon successful completion, the *pthread\_mutexattr\_getprotocol()* and  
34664 *pthread\_mutexattr\_setprotocol()* functions shall return zero; otherwise, an error number shall be  
34665 returned to indicate the error.

#### 34666 ERRORS

34667 The *pthread\_mutexattr\_setprotocol()* function shall fail if:

34668 [ENOTSUP] The value specified by *protocol* is an unsupported value.

34669 The *pthread\_mutexattr\_getprotocol()* and *pthread\_mutexattr\_setprotocol()* functions may fail if:

34670 [EINVAL] The value specified by *attr* or *protocol* is invalid.

34671 [EPERM] The caller does not have the privilege to perform the operation.

34672 These functions shall not return an error code of [EINTR].

#### 34673 EXAMPLES

34674 None.

#### 34675 APPLICATION USAGE

34676 None.

#### 34677 RATIONALE

34678 None.

#### 34679 FUTURE DIRECTIONS

34680 None.

#### 34681 SEE ALSO

34682 *pthread\_cond\_destroy()*, *pthread\_create()*, *pthread\_mutex\_destroy()*, the Base Definitions volume of  
34683 IEEE Std 1003.1-200x, <pthread.h>

#### 34684 CHANGE HISTORY

34685 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

34686 Marked as part of the Realtime Threads Feature Group.

#### 34687 Issue 6

34688 The *pthread\_mutexattr\_getprotocol()* and *pthread\_mutexattr\_setprotocol()* functions are marked as  
34689 part of the Threads option and either the Thread Priority Protection or Thread Priority  
34690 Inheritance options.

34691 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
34692 implementation does not support the Thread Priority Protection or Thread Priority Inheritance  
34693 options.

34694 The **restrict** keyword is added to the *pthread\_mutexattr\_getprotocol()* prototype for alignment  
34695 with the ISO/IEC 9899:1999 standard.

34696 **NAME**

34697 pthread\_mutexattr\_getpshared, pthread\_mutexattr\_setpshared — get and set process-shared  
 34698 attribute

34699 **SYNOPSIS**

```
34700 THR TSH #include <pthread.h>
```

```
34701 int pthread_mutexattr_getpshared(  
34702     const pthread_mutexattr_t *restrict attr,  
34703     int *restrict pshared);  
34704 int pthread_mutexattr_setpshared(pthread_mutexattr_t *attr,  
34705     int pshared);  
34706
```

34707 **DESCRIPTION**

34708 The *pthread\_mutexattr\_getpshared()* function shall obtain the value of the *process-shared* attribute  
 34709 from the attributes object referenced by *attr*. The *pthread\_mutexattr\_setpshared()* function shall  
 34710 set the *process-shared* attribute in an initialized attributes object referenced by *attr*.

34711 The *process-shared* attribute is set to PTHREAD\_PROCESS\_SHARED to permit a mutex to be  
 34712 operated upon by any thread that has access to the memory where the mutex is allocated, even if  
 34713 the mutex is allocated in memory that is shared by multiple processes. If the *process-shared*  
 34714 attribute is PTHREAD\_PROCESS\_PRIVATE, the mutex shall only be operated upon by threads  
 34715 created within the same process as the thread that initialized the mutex; if threads of differing  
 34716 processes attempt to operate on such a mutex, the behavior is undefined. The default value of  
 34717 the attribute shall be PTHREAD\_PROCESS\_PRIVATE.

34718 **RETURN VALUE**

34719 Upon successful completion, *pthread\_mutexattr\_setpshared()* shall return zero; otherwise, an error  
 34720 number shall be returned to indicate the error.

34721 Upon successful completion, *pthread\_mutexattr\_getpshared()* shall return zero and stores the  
 34722 value of the *process-shared* attribute of *attr* into the object referenced by the *pshared* parameter.  
 34723 Otherwise, an error number shall be returned to indicate the error.

34724 **ERRORS**

34725 The *pthread\_mutexattr\_getpshared()* and *pthread\_mutexattr\_setpshared()* functions may fail if:

34726 [EINVAL] The value specified by *attr* is invalid.

34727 The *pthread\_mutexattr\_setpshared()* function may fail if:

34728 [EINVAL] The new value specified for the attribute is outside the range of legal values  
 34729 for that attribute.

34730 These functions shall not return an error code of [EINTR].

34731 **EXAMPLES**

34732 None.

34733 **APPLICATION USAGE**

34734 None.

34735 **RATIONALE**

34736 None.

34737 **FUTURE DIRECTIONS**

34738 None.

34739 **SEE ALSO**

34740 *pthread\_cond\_destroy()*, *pthread\_create()*, *pthread\_mutex\_destroy()*, *pthread\_mutexattr\_destroy()*, the  
34741 Base Definitions volume of IEEE Std 1003.1-200x, <**pthread.h**>

34742 **CHANGE HISTORY**

34743 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

34744 **Issue 6**

34745 The *pthread\_mutexattr\_getpshared()* and *pthread\_mutexattr\_setpshared()* functions are marked as  
34746 part of the Threads and Thread Process-Shared Synchronization options.

34747 The **restrict** keyword is added to the *pthread\_mutexattr\_getpshared()* prototype for alignment  
34748 with the ISO/IEC 9899:1999 standard.

34749 **NAME**

34750 pthread\_mutexattr\_gettype, pthread\_mutexattr\_settype — get and set a mutex type attribute

34751 **SYNOPSIS**

```
34752 XSI #include <pthread.h>
34753
34754 int pthread_mutexattr_gettype(const pthread_mutexattr_t *restrict attr,
34755 int *restrict type);
34756 int pthread_mutexattr_settype(pthread_mutexattr_t *attr, int type);
```

34757 **DESCRIPTION**

34758 The *pthread\_mutexattr\_gettype()* and *pthread\_mutexattr\_settype()* functions, respectively, shall get  
 34759 and set the mutex *type* attribute. This attribute is set in the *type* parameter to these functions. The  
 34760 default value of the *type* attribute is PTHREAD\_MUTEX\_DEFAULT.

34761 The type of mutex is contained in the *type* attribute of the mutex attributes. Valid mutex types  
 34762 include:

## 34763 PTHREAD\_MUTEX\_NORMAL

34764 This type of mutex does not detect deadlock. A thread attempting to relock this mutex  
 34765 without first unlocking it shall deadlock. Attempting to unlock a mutex locked by a  
 34766 different thread results in undefined behavior. Attempting to unlock an unlocked mutex  
 34767 results in undefined behavior.

## 34768 PTHREAD\_MUTEX\_ERRORCHECK

34769 This type of mutex provides error checking. A thread attempting to relock this mutex  
 34770 without first unlocking it shall return with an error. A thread attempting to unlock a mutex  
 34771 which another thread has locked shall return with an error. A thread attempting to unlock  
 34772 an unlocked mutex shall return with an error.

## 34773 PTHREAD\_MUTEX\_RECURSIVE

34774 A thread attempting to relock this mutex without first unlocking it shall succeed in locking  
 34775 the mutex. The relocking deadlock which can occur with mutexes of type  
 34776 PTHREAD\_MUTEX\_NORMAL cannot occur with this type of mutex. Multiple locks of this  
 34777 mutex shall require the same number of unlocks to release the mutex before another thread  
 34778 can acquire the mutex. A thread attempting to unlock a mutex which another thread has  
 34779 locked shall return with an error. A thread attempting to unlock an unlocked mutex shall  
 34780 return with an error.

## 34781 PTHREAD\_MUTEX\_DEFAULT

34782 Attempting to recursively lock a mutex of this type results in undefined behavior.  
 34783 Attempting to unlock a mutex of this type which was not locked by the calling thread  
 34784 results in undefined behavior. Attempting to unlock a mutex of this type which is not  
 34785 locked results in undefined behavior. An implementation may map this mutex to one of the  
 34786 other mutex types.

34787 **RETURN VALUE**

34788 Upon successful completion, the *pthread\_mutexattr\_gettype()* function shall return zero and store  
 34789 the value of the *type* attribute of *attr* into the object referenced by the *type* parameter. Otherwise,  
 34790 an error shall be returned to indicate the error.

34791 If successful, the *pthread\_mutexattr\_settype()* function shall return zero; otherwise, an error  
 34792 number shall be returned to indicate the error.

34793 **ERRORS**

34794 The *pthread\_mutexattr\_settype()* function shall fail if:

34795 [EINVAL] The value *type* is invalid.

34796 The *pthread\_mutexattr\_gettype()* and *pthread\_mutexattr\_settype()* functions may fail if:

34797 [EINVAL] The value specified by *attr* is invalid.

34798 These functions shall not return an error code of [EINTR].

34799 **EXAMPLES**

34800 None.

34801 **APPLICATION USAGE**

34802 It is advised that an application should not use a PTHREAD\_MUTEX\_RECURSIVE mutex with  
34803 condition variables because the implicit unlock performed for a *pthread\_cond\_timedwait()* or  
34804 *pthread\_cond\_wait()* may not actually release the mutex (if it had been locked multiple times). If  
34805 this happens, no other thread can satisfy the condition of the predicate.

34806 **RATIONALE**

34807 None.

34808 **FUTURE DIRECTIONS**

34809 None.

34810 **SEE ALSO**

34811 *pthread\_cond\_timedwait()*, *pthread\_cond\_wait()*, the Base Definitions volume of  
34812 IEEE Std 1003.1-200x, <pthread.h>

34813 **CHANGE HISTORY**

34814 First released in Issue 5.

34815 **Issue 6**

34816 The Open Group Corrigendum U033/3 is applied. The SYNOPSIS for  
34817 *pthread\_mutexattr\_gettype()* is updated so that the first argument is of type **const**  
34818 **pthread\_mutexattr\_t\***.

34819 The **restrict** keyword is added to the *pthread\_mutexattr\_gettype()* prototype for alignment with  
34820 the ISO/IEC 9899:1999 standard.

34821 **NAME**

34822 pthread\_mutexattr\_init — initialize mutex attributes object

34823 **SYNOPSIS**

34824 THR #include &lt;pthread.h&gt;

34825 int pthread\_mutexattr\_init(pthread\_mutexattr\_t \*attr);

34826

34827 **DESCRIPTION**34828 Refer to *pthread\_mutexattr\_destroy()*.

34829 **NAME**

34830 pthread\_mutexattr\_setprioceiling — set prioceiling attribute of mutex attributes object  
34831 (**REALTIME THREADS**)

34832 **SYNOPSIS**

34833 THR TPP #include <pthread.h>

```
34834 int pthread_mutexattr_setprioceiling(pthread_mutexattr_t *attr,  
34835 int prioceiling);
```

34836

34837 **DESCRIPTION**

34838 Refer to *pthread\_mutexattr\_getprioceiling()*.

34839 **NAME**

34840 pthread\_mutexattr\_setprotocol — set protocol attribute of mutex attributes object (**REALTIME**  
34841 **THREADS**)

34842 **SYNOPSIS**

34843 THR #include <pthread.h>

34844 TPP|TPI int pthread\_mutexattr\_setprotocol(pthread\_mutexattr\_t \*attr,  
34845 int protocol);

34846

34847 **DESCRIPTION**

34848 Refer to *pthread\_mutexattr\_setprotocol()*.

34849 **NAME**

34850 pthread\_mutexattr\_setpshared — set process-shared attribute

34851 **SYNOPSIS**

34852 THR TSH #include <pthread.h>

```
34853 int pthread_mutexattr_setpshared(pthread_mutexattr_t *attr,  
34854 int pshared);
```

34855

34856 **DESCRIPTION**

34857 Refer to *pthread\_mutexattr\_getpshared()*.

34858 **NAME**

34859 pthread\_mutexattr\_settype — set a mutex type attribute

34860 **SYNOPSIS**

34861 xSI #include &lt;pthread.h&gt;

34862 int pthread\_mutexattr\_settype(pthread\_mutexattr\_t \*attr, int type);

34863

34864 **DESCRIPTION**34865 Refer to *pthread\_mutexattr\_gettype()*.

## 34866 NAME

34867 pthread\_once — dynamic package initialization

## 34868 SYNOPSIS

34869 THR #include &lt;pthread.h&gt;

```

34870 int pthread_once(pthread_once_t *once_control,
34871                 void (*init_routine)(void));
34872 pthread_once_t once_control = PTHREAD_ONCE_INIT;
34873

```

## 34874 DESCRIPTION

34875 The first call to *pthread\_once()* by any thread in a process, with a given *once\_control*, shall call the  
 34876 *init\_routine* with no arguments. Subsequent calls of *pthread\_once()* with the same *once\_control*  
 34877 shall not call the *init\_routine*. On return from *pthread\_once()*, *init\_routine* shall have completed. |  
 34878 The *once\_control* parameter shall determine whether the associated initialization routine has |  
 34879 been called.

34880 The *pthread\_once()* function is not a cancellation point. However, if *init\_routine* is a cancellation  
 34881 point and is canceled, the effect on *once\_control* shall be as if *pthread\_once()* was never called.

34882 The constant PTHREAD\_ONCE\_INIT is defined in the <pthread.h> header. |

34883 The behavior of *pthread\_once()* is undefined if *once\_control* has automatic storage duration or is  
 34884 not initialized by PTHREAD\_ONCE\_INIT.

## 34885 RETURN VALUE

34886 Upon successful completion, *pthread\_once()* shall return zero; otherwise, an error number shall  
 34887 be returned to indicate the error.

## 34888 ERRORS

34889 The *pthread\_once()* function may fail if:

34890 [EINVAL] If either *once\_control* or *init\_routine* is invalid.

34891 The *pthread\_once()* function shall not return an error code of [EINTR].

## 34892 EXAMPLES

34893 None.

## 34894 APPLICATION USAGE

34895 None.

## 34896 RATIONALE

34897 Some C libraries are designed for dynamic initialization. That is, the global initialization for the  
 34898 library is performed when the first procedure in the library is called. In a single-threaded  
 34899 program, this is normally implemented using a static variable whose value is checked on entry  
 34900 to a routine, as follows:

```

34901 static int random_is_initialized = 0;
34902 extern int initialize_random();
34903
34904 int random_function()
34905 {
34906     if (random_is_initialized == 0) {
34907         initialize_random();
34908         random_is_initialized = 1;
34909     }
34910     ... /* Operations performed after initialization. */
34911 }

```

34911 To keep the same structure in a multi-threaded program, a new primitive is needed. Otherwise,  
34912 library initialization has to be accomplished by an explicit call to a library-exported initialization  
34913 function prior to any use of the library.

34914 For dynamic library initialization in a multi-threaded process, a simple initialization flag is not  
34915 sufficient; the flag needs to be protected against modification by multiple threads  
34916 simultaneously calling into the library. Protecting the flag requires the use of a mutex; however,  
34917 mutexes have to be initialized before they are used. Ensuring that the mutex is only initialized  
34918 once requires a recursive solution to this problem.

34919 The use of *pthread\_once()* not only supplies an implementation-guaranteed means of dynamic  
34920 initialization, it provides an aid to the reliable construction of multi-threaded and realtime  
34921 systems. The preceding example then becomes:

```
34922 #include <pthread.h>
34923 static pthread_once_t random_is_initialized = PTHREAD_ONCE_INIT;
34924 extern int initialize_random();

34925 int random_function()
34926 {
34927     (void) pthread_once(&random_is_initialized, initialize_random);
34928     ... /* Operations performed after initialization. */
34929 }
```

34930 Note that a **pthread\_once\_t** cannot be an array because some compilers do not accept the  
34931 construct **&<array\_name>**.

#### 34932 **FUTURE DIRECTIONS**

34933 None.

#### 34934 **SEE ALSO**

34935 The Base Definitions volume of IEEE Std 1003.1-200x, **<pthread.h>**

#### 34936 **CHANGE HISTORY**

34937 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

#### 34938 **Issue 6**

34939 The *pthread\_once()* function is marked as part of the Threads option.

34940 The [EINVAL] error is added as a may fail case for if either argument is invalid.

## 34941 NAME

34942 pthread\_rwlock\_destroy, pthread\_rwlock\_init — destroy and initialize a read-write lock object

## 34943 SYNOPSIS

34944 THR #include &lt;pthread.h&gt;

```
34945 int pthread_rwlock_destroy(pthread_rwlock_t *rwlock);
34946 int pthread_rwlock_init(pthread_rwlock_t *restrict rwlock,
34947 const pthread_rwlockattr_t *restrict attr);
34948
```

## 34949 DESCRIPTION

34950 The *pthread\_rwlock\_destroy()* function shall destroy the read-write lock object referenced by  
 34951 *rwlock* and release any resources used by the lock. The effect of subsequent use of the lock is  
 34952 undefined until the lock is reinitialized by another call to *pthread\_rwlock\_init()*. An  
 34953 implementation may cause *pthread\_rwlock\_destroy()* to set the object referenced by *rwlock* to an  
 34954 invalid value. Results are undefined if *pthread\_rwlock\_destroy()* is called when any thread holds  
 34955 *rwlock*. Attempting to destroy an uninitialized read-write lock results in undefined behavior.

34956 The *pthread\_rwlock\_init()* function shall allocate any resources required to use the read-write  
 34957 lock referenced by *rwlock* and initializes the lock to an unlocked state with attributes referenced  
 34958 by *attr*. If *attr* is NULL, the default read-write lock attributes shall be used; the effect is the same  
 34959 as passing the address of a default read-write lock attributes object. Once initialized, the lock can  
 34960 be used any number of times without being reinitialized. Results are undefined if  
 34961 *pthread\_rwlock\_init()* is called specifying an already initialized read-write lock. Results are  
 34962 undefined if a read-write lock is used without first being initialized.

34963 If the *pthread\_rwlock\_init()* function fails, *rwlock* shall not be initialized and the contents of *rwlock*  
 34964 are undefined.

34965 Only the object referenced by *rwlock* may be used for performing synchronization. The result of  
 34966 referring to copies of that object in calls to *pthread\_rwlock\_destroy()*, *pthread\_rwlock\_rdlock()*,  
 34967 *pthread\_rwlock\_timedrdlock()*, *pthread\_rwlock\_timedwrlock()*, *pthread\_rwlock\_tryrdlock()*,  
 34968 *pthread\_rwlock\_trywrlock()*, *pthread\_rwlock\_unlock()*, or *pthread\_rwlock\_wrlock()* is undefined.

## 34969 RETURN VALUE

34970 If successful, the *pthread\_rwlock\_destroy()* and *pthread\_rwlock\_init()* functions shall return zero;  
 34971 otherwise, an error number shall be returned to indicate the error.

34972 The [EBUSY] and [EINVAL] error checks, if implemented, act as if they were performed  
 34973 immediately at the beginning of processing for the function and caused an error return prior to  
 34974 modifying the state of the read-write lock specified by *rwlock*.

## 34975 ERRORS

34976 The *pthread\_rwlock\_destroy()* function may fail if:

34977 [EBUSY] The implementation has detected an attempt to destroy the object referenced  
 34978 by *rwlock* while it is locked.

34979 [EINVAL] The value specified by *rwlock* is invalid.

34980 The *pthread\_rwlock\_init()* function shall fail if:

34981 [EAGAIN] The system lacked the necessary resources (other than memory) to initialize  
 34982 another read-write lock.

34983 [ENOMEM] Insufficient memory exists to initialize the read-write lock.

34984 [EPERM] The caller does not have the privilege to perform the operation.

- 34985 The *pthread\_rwlock\_init()* function may fail if:
- 34986 [EBUSY] The implementation has detected an attempt to reinitialize the object  
34987 referenced by *rwlock*, a previously initialized but not yet destroyed read-write  
34988 lock.
- 34989 [EINVAL] The value specified by *attr* is invalid.
- 34990 These functions shall not return an error code of [EINTR].
- 34991 **EXAMPLES**
- 34992 None.
- 34993 **APPLICATION USAGE**
- 34994 None.
- 34995 **RATIONALE**
- 34996 None.
- 34997 **FUTURE DIRECTIONS**
- 34998 None.
- 34999 **SEE ALSO**
- 35000 *pthread\_rwlock\_rdlock()*, *pthread\_rwlock\_timedrdlock()*, *pthread\_rwlock\_timedwrlock()*,  
35001 *pthread\_rwlock\_tryrdlock()*, *pthread\_rwlock\_trywrlock()*, *pthread\_rwlock\_unlock()*,  
35002 *pthread\_rwlock\_wrlock()*, the Base Definitions volume of IEEE Std 1003.1-200x, <pthread.h>
- 35003 **CHANGE HISTORY**
- 35004 First released in Issue 5.
- 35005 **Issue 6**
- 35006 The following changes are made for alignment with IEEE Std 1003.1j-2000:
- 35007 • The margin code in the SYNOPSIS is changed to THR to indicate that the functionality is  
35008 now part of the Threads option (previously it was part of the Read-Write Locks option in  
35009 IEEE Std 1003.1j-2000 and also part of the XSI extension). The initializer macro is also deleted  
35010 from the SYNOPSIS.
- 35011 • The DESCRIPTION is updated as follows:
- 35012 — It explicitly notes allocation of resources upon initialization of a read-write lock object.
- 35013 — A paragraph is added specifying that copies of read-write lock objects may not be used.
- 35014 • An [EINVAL] error is added to the ERRORS section for *pthread\_rwlock\_init()*, indicating that  
35015 the *rwlock* value is invalid.
- 35016 • The SEE ALSO section is updated.
- 35017 The **restrict** keyword is added to the *pthread\_rwlock\_init()* prototype for alignment with the  
35018 ISO/IEC 9899:1999 standard.

35019 **NAME**

35020 pthread\_rwlock\_init — initialize a read-write lock object

35021 **SYNOPSIS**

35022 THR #include <pthread.h>

```
35023 int pthread_rwlock_init(pthread_rwlock_t *restrict rwlock,  
35024     const pthread_rwlockattr_t *restrict attr);  
35025
```

35026 **DESCRIPTION**

35027 Refer to *pthread\_rwlock\_destroy()*.

35028 **NAME**

35029 pthread\_rwlock\_rdlock, pthread\_rwlock\_tryrdlock — lock a read-write lock object for reading

35030 **SYNOPSIS**

35031 THR #include &lt;pthread.h&gt;

35032 int pthread\_rwlock\_rdlock(pthread\_rwlock\_t \*rwlock);

35033 int pthread\_rwlock\_tryrdlock(pthread\_rwlock\_t \*rwlock);

35034

35035 **DESCRIPTION**

35036 The *pthread\_rwlock\_rdlock()* function shall apply a read lock to the read-write lock referenced by  
 35037 *rwlock*. The calling thread acquires the read lock if a writer does not hold the lock and there are  
 35038 no writers blocked on the lock.

35039 TPS If the Thread Execution Scheduling option is supported, and the threads involved in the lock are  
 35040 executing with the scheduling policies SCHED\_FIFO or SCHED\_RR, the calling thread shall not  
 35041 acquire the lock if a writer holds the lock or if writers of higher or equal priority are blocked on  
 35042 the lock; otherwise, the calling thread shall acquire the lock.

35043 TPS TSP If the Threads Execution Scheduling option is supported, and the threads involved in the lock  
 35044 are executing with the SCHED\_SPORADIC scheduling policy, the calling thread shall not  
 35045 acquire the lock if a writer holds the lock or if writers of higher or equal priority are blocked on  
 35046 the lock; otherwise, the calling thread shall acquire the lock.

35047 If the Thread Execution Scheduling option is not supported, it is implementation-defined  
 35048 whether the calling thread acquires the lock when a writer does not hold the lock and there are  
 35049 writers blocked on the lock. If a writer holds the lock, the calling thread shall not acquire the  
 35050 read lock. If the read lock is not acquired, the calling thread shall block until it can acquire the  
 35051 lock. The calling thread may deadlock if at the time the call is made it holds a write lock.

35052 A thread may hold multiple concurrent read locks on *rwlock* (that is, successfully call the  
 35053 *pthread\_rwlock\_rdlock()* function *n* times). If so, the application shall ensure that the thread  
 35054 performs matching unlocks (that is, it calls the *pthread\_rwlock\_unlock()* function *n* times).

35055 The maximum number of simultaneous read locks that an implementation guarantees can be  
 35056 applied to a read-write lock shall be implementation-defined. The *pthread\_rwlock\_rdlock()*  
 35057 function may fail if this maximum would be exceeded.

35058 The *pthread\_rwlock\_tryrdlock()* function shall apply a read lock as in the *pthread\_rwlock\_rdlock()*  
 35059 function, with the exception that the function shall fail if the equivalent *pthread\_rwlock\_rdlock()*  
 35060 call would have blocked the calling thread. In no case shall the *pthread\_rwlock\_tryrdlock()*  
 35061 function ever block; it always either acquires the lock or fails and returns immediately.

35062 Results are undefined if any of these functions are called with an uninitialized read-write lock.

35063 If a signal is delivered to a thread waiting for a read-write lock for reading, upon return from the  
 35064 signal handler the thread resumes waiting for the read-write lock for reading as if it was not  
 35065 interrupted.

35066 **RETURN VALUE**

35067 If successful, the *pthread\_rwlock\_rdlock()* function shall return zero; otherwise, an error number  
 35068 shall be returned to indicate the error.

35069 The *pthread\_rwlock\_tryrdlock()* function shall return zero if the lock for reading on the read-write  
 35070 lock object referenced by *rwlock* is acquired. Otherwise, an error number shall be returned to  
 35071 indicate the error.

35072 **ERRORS**

35073 The *pthread\_rwlock\_tryrdlock()* function shall fail if:

35074 [EBUSY] The read-write lock could not be acquired for reading because a writer holds  
35075 the lock or a writer with the appropriate priority was blocked on it.

35076 The *pthread\_rwlock\_rdlock()* and *pthread\_rwlock\_tryrdlock()* functions may fail if:

35077 [EINVAL] The value specified by *rwlock* does not refer to an initialized read-write lock  
35078 object.

35079 [EAGAIN] The read lock could not be acquired because the maximum number of read  
35080 locks for *rwlock* has been exceeded.

35081 The *pthread\_rwlock\_rdlock()* function may fail if:

35082 [EDEADLK] The current thread already owns the read-write lock for writing.

35083 These functions shall not return an error code of [EINTR].

35084 **EXAMPLES**

35085 None.

35086 **APPLICATION USAGE**

35087 Applications using these functions may be subject to priority inversion, as discussed in the Base  
35088 Definitions volume of IEEE Std 1003.1-200x, Section 3.285, Priority Inversion.

35089 **RATIONALE**

35090 None.

35091 **FUTURE DIRECTIONS**

35092 None.

35093 **SEE ALSO**

35094 *pthread\_rwlock\_destroy()*, *pthread\_rwlock\_init()*, *pthread\_rwlock\_timedrdlock()*,  
35095 *pthread\_rwlock\_timedwrlock()*, *pthread\_rwlock\_trywrlock()*, *pthread\_rwlock\_unlock()*,  
35096 *pthread\_rwlock\_wrlock()*, the Base Definitions volume of IEEE Std 1003.1-200x, <pthread.h>

35097 **CHANGE HISTORY**

35098 First released in Issue 5.

35099 **Issue 6**

35100 The following changes are made for alignment with IEEE Std 1003.1j-2000:

- 35101 • The margin code in the SYNOPSIS is changed to THR to indicate that the functionality is  
35102 now part of the Threads option (previously it was part of the Read-Write Locks option in  
35103 IEEE Std 1003.1j-2000 and also part of the XSI extension).
- 35104 • The DESCRIPTION is updated as follows:
- 35105 — Conditions under which writers have precedence over readers are specified.
  - 35106 — Failure of *pthread\_rwlock\_tryrdlock()* is clarified.
  - 35107 — A paragraph on the maximum number of read locks is added.
- 35108 • In the ERRORS sections, [EBUSY] is modified to take into account write priority, and  
35109 [EDEADLK] is deleted as a *pthread\_rwlock\_tryrdlock()* error.
- 35110 • The SEE ALSO section is updated.

35111 **NAME**

35112 pthread\_rwlock\_timedrdlock — lock a read-write lock for reading

35113 **SYNOPSIS**

35114 THR TMO #include &lt;pthread.h&gt;

35115 #include &lt;time.h&gt;

35116 int pthread\_rwlock\_timedrdlock(pthread\_rwlock\_t \*restrict *rwlock*,35117 const struct timespec \*restrict *abs\_timeout*);

35118

35119 **DESCRIPTION**

35120 The *pthread\_rwlock\_timedrdlock()* function shall apply a read lock to the read-write lock  
 35121 referenced by *rwlock* as in the *pthread\_rwlock\_rdlock()* function. However, if the lock cannot be  
 35122 acquired without waiting for other threads to unlock the lock, this wait shall be terminated  
 35123 when the specified timeout expires. The timeout shall expire when the absolute time specified  
 35124 by *abs\_timeout* passes, as measured by the clock on which timeouts are based (that is, when the  
 35125 value of that clock equals or exceeds *abs\_timeout*), or if the absolute time specified by *abs\_timeout*  
 35126 has already been passed at the time of the call.

35127 TMR If the Timers option is supported, the timeout shall be based on the `CLOCK_REALTIME` clock. If  
 35128 the Timers option is not supported, the timeout shall be based on the system clock as returned  
 35129 by the *time()* function. The resolution of the timeout shall be the resolution of the clock on which  
 35130 it is based. The **timespec** data type is defined in the `<time.h>` header. Under no circumstances  
 35131 shall the function fail with a timeout if the lock can be acquired immediately. The validity of the  
 35132 *abs\_timeout* parameter need not be checked if the lock can be immediately acquired.

35133 If a signal that causes a signal handler to be executed is delivered to a thread blocked on a read-  
 35134 write lock via a call to *pthread\_rwlock\_timedrdlock()*, upon return from the signal handler the  
 35135 thread shall resume waiting for the lock as if it was not interrupted.

35136 The calling thread may deadlock if at the time the call is made it holds a write lock on *rwlock*.  
 35137 The results are undefined if this function is called with an uninitialized read-write lock.

35138 **RETURN VALUE**

35139 The *pthread\_rwlock\_timedrdlock()* function shall return zero if the lock for reading on the read-  
 35140 write lock object referenced by *rwlock* is acquired. Otherwise, an error number shall be returned  
 35141 to indicate the error.

35142 **ERRORS**35143 The *pthread\_rwlock\_timedrdlock()* function shall fail if:

35144 [ETIMEDOUT] The lock could not be acquired before the specified timeout expired.

35145 The *pthread\_rwlock\_timedrdlock()* function may fail if:35146 [EAGAIN] The read lock could not be acquired because the maximum number of read  
35147 locks for lock would be exceeded.35148 [EDEADLK] The calling thread already holds a write lock on *rwlock*.35149 [EINVAL] The value specified by *rwlock* does not refer to an initialized read-write lock  
35150 object, or the *abs\_timeout* nanosecond value is less than zero or greater than or  
35151 equal to 1 000 million.

35152 This function shall not return an error code of [EINTR].

## 35153 EXAMPLES

35154 None.

## 35155 APPLICATION USAGE

35156 Applications using this function may be subject to priority inversion, as discussed in the Base  
35157 Definitions volume of IEEE Std 1003.1-200x, Section 3.285, Priority Inversion.

35158 The *pthread\_rwlock\_timedrdlock()* function is part of the Threads and Timeouts options and need  
35159 not be provided on all implementations.

## 35160 RATIONALE

35161 None.

## 35162 FUTURE DIRECTIONS

35163 None.

## 35164 SEE ALSO

35165 *pthread\_rwlock\_destroy()*, *pthread\_rwlock\_init()*, *pthread\_rwlock\_rdlock()*,  
35166 *pthread\_rwlock\_timedwrlock()*, *pthread\_rwlock\_tryrdlock()*, *pthread\_rwlock\_trywrlock()*,  
35167 *pthread\_rwlock\_unlock()*, *pthread\_rwlock\_wrlock()*, the Base Definitions volume of  
35168 IEEE Std 1003.1-200x, <pthread.h>, <time.h>

## 35169 CHANGE HISTORY

35170 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

35171 **NAME**

35172 pthread\_rwlock\_timedwrlock — lock a read-write lock for writing

35173 **SYNOPSIS**

35174 THR TMO #include &lt;pthread.h&gt;

35175 #include &lt;time.h&gt;

35176 int pthread\_rwlock\_timedwrlock(pthread\_rwlock\_t \*restrict *rwlock*,35177 const struct timespec \*restrict *abs\_timeout*);

35178

35179 **DESCRIPTION**

35180 The *pthread\_rwlock\_timedwrlock()* function shall apply a write lock to the read-write lock  
 35181 referenced by *rwlock* as in the *pthread\_rwlock\_wrlock()* function. However, if the lock cannot be  
 35182 acquired without waiting for other threads to unlock the lock, this wait shall be terminated  
 35183 when the specified timeout expires. The timeout shall expire when the absolute time specified  
 35184 by *abs\_timeout* passes, as measured by the clock on which timeouts are based (that is, when the  
 35185 value of that clock equals or exceeds *abs\_timeout*), or if the absolute time specified by *abs\_timeout*  
 35186 has already been passed at the time of the call.

35187 TMR If the Timers option is supported, the timeout shall be based on the `CLOCK_REALTIME` clock. If  
 35188 the Timers option is not supported, the timeout shall be based on the system clock as returned  
 35189 by the *time()* function. The resolution of the timeout shall be the resolution of the clock on which  
 35190 it is based. The `timespec` data type is defined in the `<time.h>` header. Under no circumstances  
 35191 shall the function fail with a timeout if the lock can be acquired immediately. The validity of the  
 35192 *abs\_timeout* parameter need not be checked if the lock can be immediately acquired.

35193 If a signal that causes a signal handler to be executed is delivered to a thread blocked on a read-  
 35194 write lock via a call to *pthread\_rwlock\_timedwrlock()*, upon return from the signal handler the  
 35195 thread shall resume waiting for the lock as if it was not interrupted.

35196 The calling thread may deadlock if at the time the call is made it holds the read-write lock. The  
 35197 results are undefined if this function is called with an uninitialized read-write lock.

35198 **RETURN VALUE**

35199 The *pthread\_rwlock\_timedwrlock()* function shall return zero if the lock for writing on the read-  
 35200 write lock object referenced by *rwlock* is acquired. Otherwise, an error number shall be returned  
 35201 to indicate the error.

35202 **ERRORS**35203 The *pthread\_rwlock\_timedwrlock()* function shall fail if:

35204 [ETIMEDOUT] The lock could not be acquired before the specified timeout expired.

35205 The *pthread\_rwlock\_timedwrlock()* function may fail if:35206 [EDEADLK] The calling thread already holds the *rwlock*.

35207 [EINVAL] The value specified by *rwlock* does not refer to an initialized read-write lock  
 35208 object, or the *abs\_timeout* nanosecond value is less than zero or greater than or  
 35209 equal to 1 000 million.

35210 This function shall not return an error code of [EINTR].

## 35211 EXAMPLES

35212 None.

## 35213 APPLICATION USAGE

35214 Applications using this function may be subject to priority inversion, as discussed in the Base  
35215 Definitions volume of IEEE Std 1003.1-200x, Section 3.285, Priority Inversion.

35216 The *pthread\_rwlock\_timedwrlock()* function is part of the Threads and Timeouts options and need  
35217 not be provided on all implementations.

## 35218 RATIONALE

35219 None.

## 35220 FUTURE DIRECTIONS

35221 None.

## 35222 SEE ALSO

35223 *pthread\_rwlock\_destroy()*, *pthread\_rwlock\_init()*, *pthread\_rwlock\_rdlock()*,  
35224 *pthread\_rwlock\_timedrdlock()*, *pthread\_rwlock\_tryrdlock()*, *pthread\_rwlock\_trywrlock()*,  
35225 *pthread\_rwlock\_unlock()*, *pthread\_rwlock\_wrlock()*, the Base Definitions volume of  
35226 IEEE Std 1003.1-200x, <pthread.h>, <time.h>

## 35227 CHANGE HISTORY

35228 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

35229 **NAME**

35230 pthread\_rwlock\_tryrdlock — lock a read-write lock object for reading

35231 **SYNOPSIS**

35232 THR #include &lt;pthread.h&gt;

35233 int pthread\_rwlock\_tryrdlock(pthread\_rwlock\_t \*rwlock);

35234

35235 **DESCRIPTION**35236 Refer to *pthread\_rwlock\_rdlock()*.

35237 **NAME**

35238 pthread\_rwlock\_trywrlock, pthread\_rwlock\_wrlock — lock a read-write lock object for writing

35239 **SYNOPSIS**

35240 THR #include <pthread.h>

35241 int pthread\_rwlock\_trywrlock(pthread\_rwlock\_t \*rwlock);

35242 int pthread\_rwlock\_wrlock(pthread\_rwlock\_t \*rwlock);

35243

35244 **DESCRIPTION**

35245 The *pthread\_rwlock\_trywrlock()* function shall apply a write lock like the *pthread\_rwlock\_wrlock()*  
 35246 function, with the exception that the function shall fail if any thread currently holds *rwlock* (for  
 35247 reading or writing).

35248 The *pthread\_rwlock\_wrlock()* function shall apply a write lock to the read-write lock referenced  
 35249 by *rwlock*. The calling thread acquires the write lock if no other thread (reader or writer) holds  
 35250 the read-write lock *rwlock*. Otherwise, the thread shall block until it can acquire the lock. The  
 35251 calling thread may deadlock if at the time the call is made it holds the read-write lock (whether a  
 35252 read or write lock).

35253 Implementations may favor writers over readers to avoid writer starvation.

35254 Results are undefined if any of these functions are called with an uninitialized read-write lock.

35255 If a signal is delivered to a thread waiting for a read-write lock for writing, upon return from the  
 35256 signal handler the thread resumes waiting for the read-write lock for writing as if it was not  
 35257 interrupted.

35258 **RETURN VALUE**

35259 The *pthread\_rwlock\_trywrlock()* function shall return zero if the lock for writing on the read-write  
 35260 lock object referenced by *rwlock* is acquired. Otherwise, an error number shall be returned to  
 35261 indicate the error.

35262 If successful, the *pthread\_rwlock\_wrlock()* function shall return zero; otherwise, an error number  
 35263 shall be returned to indicate the error.

35264 **ERRORS**

35265 The *pthread\_rwlock\_trywrlock()* function shall fail if:

35266 [EBUSY] The read-write lock could not be acquired for writing because it was already  
 35267 locked for reading or writing.

35268 The *pthread\_rwlock\_trywrlock()* and *pthread\_rwlock\_wrlock()* functions may fail if:

35269 [EINVAL] The value specified by *rwlock* does not refer to an initialized read-write lock  
 35270 object.

35271 The *pthread\_rwlock\_wrlock()* function may fail if:

35272 [EDEADLK] The current thread already owns the read-write lock for writing or reading.

35273 These functions shall not return an error code of [EINTR].

35274 **EXAMPLES**

35275 None.

35276 **APPLICATION USAGE**

35277 Applications using these functions may be subject to priority inversion, as discussed in the Base  
35278 Definitions volume of IEEE Std 1003.1-200x, Section 3.285, Priority Inversion.

35279 **RATIONALE**

35280 None.

35281 **FUTURE DIRECTIONS**

35282 None.

35283 **SEE ALSO**

35284 *pthread\_rwlock\_destroy()*, *pthread\_rwlock\_init()*, *pthread\_rwlock\_rdlock()*,  
35285 *pthread\_rwlock\_timedrdlock()*, *pthread\_rwlock\_timedwrlock()*, *pthread\_rwlock\_tryrdlock()*,  
35286 *pthread\_rwlock\_unlock()*, the Base Definitions volume of IEEE Std 1003.1-200x, <pthread.h>

35287 **CHANGE HISTORY**

35288 First released in Issue 5.

35289 **Issue 6**

35290 The following changes are made for alignment with IEEE Std 1003.1j-2000:

- 35291 • The margin code in the SYNOPSIS is changed to THR to indicate that the functionality is  
35292 now part of the Threads option (previously it was part of the Read-Write Locks option in  
35293 IEEE Std 1003.1j-2000 and also part of the XSI extension).
- 35294 • The [EDEADLK] error is deleted as a *pthread\_rwlock\_trywrlock()* error.
- 35295 • The SEE ALSO section is updated.

35296 **NAME**

35297 pthread\_rwlock\_unlock — unlock a read-write lock object

35298 **SYNOPSIS**

35299 THR #include &lt;pthread.h&gt;

35300 int pthread\_rwlock\_unlock(pthread\_rwlock\_t \*rwlock);

35301

35302 **DESCRIPTION**

35303 The *pthread\_rwlock\_unlock()* function shall release a lock held on the read-write lock object |  
35304 referenced by *rwlock*. Results are undefined if the read-write lock *rwlock* is not held by the |  
35305 calling thread.

35306 If this function is called to release a read lock from the read-write lock object and there are other |  
35307 read locks currently held on this read-write lock object, the read-write lock object remains in the |  
35308 read locked state. If this function releases the last read lock for this read-write lock object, the |  
35309 read-write lock object shall be put in the unlocked state with no owners.

35310 If this function is called to release a write lock for this read-write lock object, the read-write lock |  
35311 object shall be put in the unlocked state.

35312 If there are threads blocked on the lock when it becomes available, the scheduling policy shall |  
35313 TPS determine which thread(s) shall acquire the lock. If the Thread Execution Scheduling option is |  
35314 supported, when threads executing with the scheduling policies SCHED\_FIFO, SCHED\_RR, or |  
35315 SCHED\_SPORADIC are waiting on the lock, they shall acquire the lock in priority order when |  
35316 the lock becomes available. For equal priority threads, write locks shall take precedence over |  
35317 read locks. If the Thread Execution Scheduling option is not supported, it is implementation- |  
35318 defined whether write locks take precedence over read locks.

35319 Results are undefined if any of these functions are called with an uninitialized read-write lock.

35320 **RETURN VALUE**

35321 If successful, the *pthread\_rwlock\_unlock()* function shall return zero; otherwise, an error number |  
35322 shall be returned to indicate the error.

35323 **ERRORS**

35324 The *pthread\_rwlock\_unlock()* function may fail if:

35325 [EINVAL] The value specified by *rwlock* does not refer to an initialized read-write lock |  
35326 object.

35327 [EPERM] The current thread does not hold a lock on the read-write lock.

35328 The *pthread\_rwlock\_unlock()* function shall not return an error code of [EINTR].

35329 **EXAMPLES**

35330 None.

35331 **APPLICATION USAGE**

35332 None.

35333 **RATIONALE**

35334 None.

35335 **FUTURE DIRECTIONS**

35336 None.

35337 **SEE ALSO**

35338 *pthread\_rwlock\_destroy()*, *pthread\_rwlock\_init()*, *pthread\_rwlock\_rdlock()*,  
35339 *pthread\_rwlock\_timedrdlock()*, *pthread\_rwlock\_timedwrlock()*, *pthread\_rwlock\_tryrdlock()*,  
35340 *pthread\_rwlock\_trywrlock()*, *pthread\_rwlock\_wrlock()*, the Base Definitions volume of  
35341 IEEE Std 1003.1-200x, <**pthread.h**>

35342 **CHANGE HISTORY**

35343 First released in Issue 5.

35344 **Issue 6**

35345 The following changes are made for alignment with IEEE Std 1003.1j-2000:

- 35346 • The margin code in the SYNOPSIS is changed to THR to indicate that the functionality is  
35347 now part of the Threads option (previously it was part of the Read-Write Locks option in  
35348 IEEE Std 1003.1j-2000 and also part of the XSI extension).
- 35349 • The DESCRIPTION is updated as follows:
  - 35350 — The conditions under which writers have precedence over readers are specified.
  - 35351 — The concept of read-write lock owner is deleted.
- 35352 • The SEE ALSO section is updated.

35353 **NAME**

35354 pthread\_rwlock\_wrlock — lock a read-write lock object for writing

35355 **SYNOPSIS**

35356 THR #include <pthread.h>

35357 int pthread\_rwlock\_wrlock(pthread\_rwlock\_t \**rwlock*);

35358

35359 **DESCRIPTION**

35360 Refer to *pthread\_rwlock\_trywrlock()*.

35361 **NAME**

35362 pthread\_rwlockattr\_destroy, pthread\_rwlockattr\_init — destroy and initialize read-write lock  
 35363 attributes object

35364 **SYNOPSIS**

```
35365 THR #include <pthread.h>
```

```
35366 int pthread_rwlockattr_destroy(pthread_rwlockattr_t *attr);
```

```
35367 int pthread_rwlockattr_init(pthread_rwlockattr_t *attr);
```

35368

35369 **DESCRIPTION**

35370 The *pthread\_rwlockattr\_destroy()* function shall destroy a read-write lock attributes object. A  
 35371 destroyed *attr* attributes object can be reinitialized using *pthread\_rwlockattr\_init()*; the results of  
 35372 otherwise referencing the object after it has been destroyed are undefined. An implementation  
 35373 may cause *pthread\_rwlockattr\_destroy()* to set the object referenced by *attr* to an invalid value.

35374 The *pthread\_rwlockattr\_init()* function shall initialize a read-write lock attributes object *attr* with  
 35375 the default value for all of the attributes defined by the implementation.

35376 Results are undefined if *pthread\_rwlockattr\_init()* is called specifying an already initialized *attr*  
 35377 attributes object.

35378 After a read-write lock attributes object has been used to initialize one or more read-write locks,  
 35379 any function affecting the attributes object (including destruction) shall not affect any previously  
 35380 initialized read-write locks.

35381 **RETURN VALUE**

35382 If successful, the *pthread\_rwlockattr\_destroy()* and *pthread\_rwlockattr\_init()* functions shall return  
 35383 zero; otherwise, an error number shall be returned to indicate the error.

35384 **ERRORS**

35385 The *pthread\_rwlockattr\_destroy()* function may fail if:

35386 [EINVAL] The value specified by *attr* is invalid.

35387 The *pthread\_rwlockattr\_init()* function shall fail if:

35388 [ENOMEM] Insufficient memory exists to initialize the read-write lock attributes object.

35389 These functions shall not return an error code of [EINTR].

35390 **EXAMPLES**

35391 None.

35392 **APPLICATION USAGE**

35393 None.

35394 **RATIONALE**

35395 None.

35396 **FUTURE DIRECTIONS**

35397 None.

35398 **SEE ALSO**

35399 *pthread\_rwlock\_init()*, *pthread\_rwlockattr\_getpshared()*, *pthread\_rwlockattr\_setpshared()*, the Base  
 35400 Definitions volume of IEEE Std 1003.1-200x, <pthread.h>

35401 **CHANGE HISTORY**

35402 First released in Issue 5.

35403 **Issue 6**

35404 The following changes are made for alignment with IEEE Std 1003.1j-2000:

- 35405 • The margin code in the SYNOPSIS is changed to THR to indicate that the functionality is
- 35406 now part of the Threads option (previously it was part of the Read-Write Locks option in
- 35407 IEEE Std 1003.1j-2000 and also part of the XSI extension).
- 35408 • The SEE ALSO section is updated.

35409 **NAME**

35410 pthread\_rwlockattr\_getpshared, pthread\_rwlockattr\_setpshared — get and set process-shared  
 35411 attribute of read-write lock attributes object

35412 **SYNOPSIS**

```
35413 THR TSH #include <pthread.h>
35414
35414 int pthread_rwlockattr_getpshared(
35415     const pthread_rwlockattr_t *restrict attr,
35416     int *restrict pshared);
35417 int pthread_rwlockattr_setpshared(pthread_rwlockattr_t *attr,
35418     int pshared);
35419
```

35420 **DESCRIPTION**

35421 The *pthread\_rwlockattr\_getpshared()* function shall obtain the value of the *process-shared* attribute  
 35422 from the initialized attributes object referenced by *attr*. The *pthread\_rwlockattr\_setpshared()* |  
 35423 function shall set the *process-shared* attribute in an initialized attributes object referenced by *attr*. |

35424 The *process-shared* attribute shall be set to PTHREAD\_PROCESS\_SHARED to permit a read-  
 35425 write lock to be operated upon by any thread that has access to the memory where the read-  
 35426 write lock is allocated, even if the read-write lock is allocated in memory that is shared by  
 35427 multiple processes. If the *process-shared* attribute is PTHREAD\_PROCESS\_PRIVATE, the read-  
 35428 write lock shall only be operated upon by threads created within the same process as the thread  
 35429 that initialized the read-write lock; if threads of differing processes attempt to operate on such a  
 35430 read-write lock, the behavior is undefined. The default value of the *process-shared* attribute shall  
 35431 be PTHREAD\_PROCESS\_PRIVATE.

35432 Additional attributes, their default values, and the names of the associated functions to get and  
 35433 set those attribute values are implementation-defined.

35434 **RETURN VALUE**

35435 Upon successful completion, the *pthread\_rwlockattr\_getpshared()* shall return zero and store the  
 35436 value of the *process-shared* attribute of *attr* into the object referenced by the *pshared* parameter.  
 35437 Otherwise, an error number shall be returned to indicate the error.

35438 If successful, the *pthread\_rwlockattr\_setpshared()* function shall return zero; otherwise, an error  
 35439 number shall be returned to indicate the error.

35440 **ERRORS**

35441 The *pthread\_rwlockattr\_getpshared()* and *pthread\_rwlockattr\_setpshared()* functions may fail if:

35442 [EINVAL] The value specified by *attr* is invalid.

35443 The *pthread\_rwlockattr\_setpshared()* function may fail if:

35444 [EINVAL] The new value specified for the attribute is outside the range of legal values  
 35445 for that attribute.

35446 These functions shall not return an error code of [EINTR].

35447 **EXAMPLES**

35448 None.

35449 **APPLICATION USAGE**

35450 None.

35451 **RATIONALE**

35452 None.

35453 **FUTURE DIRECTIONS**

35454 None.

35455 **SEE ALSO**

35456 *pthread\_rwlock\_init()*, *pthread\_rwlockattr\_destroy()*, *pthread\_rwlockattr\_init()*, the Base Definitions  
35457 volume of IEEE Std 1003.1-200x, <pthread.h>

35458 **CHANGE HISTORY**

35459 First released in Issue 5.

35460 **Issue 6**

35461 The following changes are made for alignment with IEEE Std 1003.1j-2000:

- 35462 • The margin code in the SYNOPSIS is changed to THR TSH to indicate that the functionality  
35463 is now part of the Threads option (previously it was part of the Read-Write Locks option in  
35464 IEEE Std 1003.1j-2000 and also part of the XSI extension).
- 35465 • The DESCRIPTION notes that additional attributes are implementation-defined.
- 35466 • The SEE ALSO section is updated.

35467 The **restrict** keyword is added to the *pthread\_rwlockattr\_getpshared()* prototype for alignment  
35468 with the ISO/IEC 9899:1999 standard.

35469 **NAME**

35470 pthread\_rwlockattr\_init — initialize read-write lock attributes object

35471 **SYNOPSIS**

35472 XSI #include &lt;pthread.h&gt;

35473 int pthread\_rwlockattr\_init(pthread\_rwlockattr\_t \*attr);

35474

35475 **DESCRIPTION**35476 Refer to *pthread\_rwlockattr\_destroy()*.

35477 **NAME**

35478 pthread\_rwlockattr\_setpshared — set process-shared attribute of read-write lock attributes  
35479 object

35480 **SYNOPSIS**

35481 XSI #include <pthread.h>

```
35482 int pthread_rwlockattr_setpshared(pthread_rwlockattr_t *attr,  
35483 int pshared);  
35484
```

35485 **DESCRIPTION**

35486 Refer to *pthread\_rwlockattr\_getpshared()*.

35487 **NAME**

35488 pthread\_self — get calling thread's ID

35489 **SYNOPSIS**

35490 THR #include &lt;pthread.h&gt;

35491 pthread\_t pthread\_self(void);

35492

35493 **DESCRIPTION**35494 The *pthread\_self()* function shall return the thread ID of the calling thread.35495 **RETURN VALUE**

35496 Refer to the DESCRIPTION.

35497 **ERRORS**

35498 No errors are defined.

35499 The *pthread\_self()* function shall not return an error code of [EINTR].35500 **EXAMPLES**

35501 None.

35502 **APPLICATION USAGE**

35503 None.

35504 **RATIONALE**35505 The *pthread\_self()* function provides a capability similar to the *getpid()* function for processes  
35506 and the rationale is the same: the creation call does not provide the thread ID to the created  
35507 thread.35508 **FUTURE DIRECTIONS**

35509 None.

35510 **SEE ALSO**35511 *pthread\_create()*, *pthread\_equal()*, the Base Definitions volume of IEEE Std 1003.1-200x,  
35512 <pthread.h>35513 **CHANGE HISTORY**

35514 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

35515 **Issue 6**35516 The *pthread\_self()* function is marked as part of the Threads option.

35517 **NAME**

35518 pthread\_setcancelstate, pthread\_setcanceltype, pthread\_testcancel — set cancelability state

35519 **SYNOPSIS**

35520 THR #include &lt;pthread.h&gt;

35521 int pthread\_setcancelstate(int *state*, int \**oldstate*);35522 int pthread\_setcanceltype(int *type*, int \**oldtype*);

35523 void pthread\_testcancel(void);

35524

35525 **DESCRIPTION**

35526 The *pthread\_setcancelstate()* function shall atomically both set the calling thread's cancelability  
35527 state to the indicated *state* and return the previous cancelability state at the location referenced  
35528 by *oldstate*. Legal values for *state* are PTHREAD\_CANCEL\_ENABLE and  
35529 PTHREAD\_CANCEL\_DISABLE.

35530 The *pthread\_setcanceltype()* function shall atomically both set the calling thread's cancelability  
35531 type to the indicated *type* and return the previous cancelability type at the location referenced by  
35532 *oldtype*. Legal values for *type* are PTHREAD\_CANCEL\_DEFERRED and  
35533 PTHREAD\_CANCEL\_ASYNCHRONOUS.

35534 The cancelability state and type of any newly created threads, including the thread in which  
35535 *main()* was first invoked, shall be PTHREAD\_CANCEL\_ENABLE and  
35536 PTHREAD\_CANCEL\_DEFERRED respectively.

35537 The *pthread\_testcancel()* function shall create a cancellation point in the calling thread. The  
35538 *pthread\_testcancel()* function shall have no effect if cancelability is disabled.

35539 **RETURN VALUE**

35540 If successful, the *pthread\_setcancelstate()* and *pthread\_setcanceltype()* functions shall return zero;  
35541 otherwise, an error number shall be returned to indicate the error.

35542 **ERRORS**35543 The *pthread\_setcancelstate()* function may fail if:

35544 [EINVAL] The specified *state* is not PTHREAD\_CANCEL\_ENABLE or  
35545 PTHREAD\_CANCEL\_DISABLE.

35546 The *pthread\_setcanceltype()* function may fail if:

35547 [EINVAL] The specified *type* is not PTHREAD\_CANCEL\_DEFERRED or  
35548 PTHREAD\_CANCEL\_ASYNCHRONOUS.

35549 These functions shall not return an error code of [EINTR].

35550 **EXAMPLES**

35551 None.

35552 **APPLICATION USAGE**

35553 None.

35554 **RATIONALE**

35555 The *pthread\_setcancelstate()* and *pthread\_setcanceltype()* functions control the points at which a |  
35556 thread may be asynchronously canceled. For cancellation control to be usable in modular fashion, |  
35557 some rules need to be followed.

35558 An object can be considered to be a generalization of a procedure. It is a set of procedures and  
35559 global variables written as a unit and called by clients not known by the object. Objects may  
35560 depend on other objects.

35561 First, cancelability should only be disabled on entry to an object, never explicitly enabled. On  
35562 exit from an object, the cancelability state should always be restored to its value on entry to the  
35563 object.

35564 This follows from a modularity argument: if the client of an object (or the client of an object that  
35565 uses that object) has disabled cancelability, it is because the client does not want to be concerned  
35566 about cleaning up if the thread is canceled while executing some sequence of actions. If an object  
35567 is called in such a state and it enables cancelability and a cancelation request is pending for that  
35568 thread, then the thread is canceled, contrary to the wish of the client that disabled.

35569 Second, the cancelability type may be explicitly set to either *deferred* or *asynchronous* upon entry  
35570 to an object. But as with the cancelability state, on exit from an object the cancelability type  
35571 should always be restored to its value on entry to the object.

35572 Finally, only functions that are cancel-safe may be called from a thread that is asynchronously  
35573 cancelable.

#### 35574 **FUTURE DIRECTIONS**

35575 None.

#### 35576 **SEE ALSO**

35577 *pthread\_cancel()*, the Base Definitions volume of IEEE Std 1003.1-200x, <pthread.h>

#### 35578 **CHANGE HISTORY**

35579 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

#### 35580 **Issue 6**

35581 The *pthread\_setcancelstate()*, *pthread\_setcanceltype()*, and *pthread\_testcancel()* functions are marked  
35582 as part of the Threads option.

35583 **NAME**

35584 pthread\_setconcurrency — set level of concurrency

35585 **SYNOPSIS**

35586 XSI #include <pthread.h>

35587 int pthread\_setconcurrency(int new\_level);

35588

35589 **DESCRIPTION**

35590 Refer to *pthread\_getconcurrency()*.

35591 **NAME**

35592 pthread\_setschedparam — dynamic thread scheduling parameters access (**REALTIME**  
35593 **THREADS**)

35594 **SYNOPSIS**

```
35595 THR TPS #include <pthread.h>
```

```
35596 int pthread_setschedparam(pthread_t thread, int policy,  
35597     const struct sched_param *param);
```

35598

35599 **DESCRIPTION**

35600 Refer to *pthread\_getschedparam()*.

## 35601 NAME

35602 pthread\_setschedprio — dynamic thread scheduling parameters access (**REALTIME**  
35603 **THREADS**)

## 35604 SYNOPSIS

```
35605 THR TPS #include <pthread.h>
```

```
35606 int pthread_setschedprio(pthread_t thread, int prio);
```

35607

## 35608 DESCRIPTION

35609 The *pthread\_setschedprio()* function shall set the scheduling priority for the thread whose thread  
35610 ID is given by *thread* to the value given by *prio*. See **Scheduling Policies** (on page 494) for a  
35611 description on how this function call affects the ordering of the thread in the thread list for its  
35612 new priority.

35613 If the *pthread\_setschedprio()* function fails, the scheduling priority of the target thread shall not be  
35614 changed.

## 35615 RETURN VALUE

35616 If successful, the *pthread\_setschedprio()* function shall return zero; otherwise, an error number  
35617 shall be returned to indicate the error.

## 35618 ERRORS

35619 The *pthread\_setschedprio()* function may fail if:

35620 [EINVAL] The value of *prio* is invalid for the scheduling policy of the specified thread.

35621 [ENOTSUP] An attempt was made to set the priority to an unsupported value.

35622 [EPERM] The caller does not have the appropriate permission to set the scheduling  
35623 policy of the specified thread.

35624 [EPERM] The implementation does not allow the application to modify the priority to  
35625 the value specified.

35626 [ESRCH] The value specified by *thread* does not refer to an existing thread.

35627 The *pthread\_setschedprio()* function shall not return an error code of [EINTR].

## 35628 EXAMPLES

35629 None.

## 35630 APPLICATION USAGE

35631 None.

## 35632 RATIONALE

35633 The *pthread\_setschedprio()* function provides a way for an application to temporarily raise its  
35634 priority and then lower it again, without having the undesired side effect of yielding to other  
35635 threads of the same priority. This is necessary if the application is to implement its own  
35636 strategies for bounding priority inversion, such as priority inheritance or priority ceilings. This  
35637 capability is especially important if the implementation does not support the Thread Priority  
35638 Protection or Thread Priority Inheritance options, but even if those options are supported it is  
35639 needed if the application is to bound priority inheritance for other resources, such as  
35640 semaphores.

35641 The standard developers considered that while it might be preferable conceptually to solve this  
35642 problem by modifying the specification of *pthread\_setschedparam()*, it was too late to make such a  
35643 change, as there may be implementations that would need to be changed. Therefore, this new  
35644 function was introduced.

35645 **FUTURE DIRECTIONS**

35646       None.

35647 **SEE ALSO**

35648       *pthread\_getschedparam()*, the Base Definitions volume of IEEE Std 1003.1-200x, <pthread.h>

35649 **CHANGE HISTORY**

35650       First released in Issue 6. Included as a response to IEEE PASC Interpretation 1003.1 #96.

35651 **NAME**

35652 pthread\_setspecific — thread-specific data management

35653 **SYNOPSIS**

35654 THR #include <pthread.h>

35655 int pthread\_setspecific(pthread\_key\_t key, const void \*value);

35656

35657 **DESCRIPTION**

35658 Refer to *pthread\_getspecific()*.

35659 **NAME**

35660 pthread\_sigmask, sigprocmask — examine and change blocked signals

35661 **SYNOPSIS**

35662 #include &lt;signal.h&gt;

35663 THR int pthread\_sigmask(int how, const sigset\_t \*restrict set,  
35664 sigset\_t \*restrict oset);35665 CX int sigprocmask(int how, const sigset\_t \*restrict set,  
35666 sigset\_t \*restrict oset);

35667

35668 **DESCRIPTION**35669 THR The *pthread\_sigmask()* function shall examine or change (or both) the calling thread's signal  
35670 mask, regardless of the number of threads in the process. The function shall be equivalent to  
35671 *sigprocmask()*, without the restriction that the call be made in a single-threaded process.35672 In a single-threaded process, the *sigprocmask()* function shall examine or change (or both) the  
35673 signal mask of the calling thread.35674 If the argument *set* is not a null pointer, it points to a set of signals to be used to change the  
35675 currently blocked set.35676 The argument *how* indicates the way in which the set is changed, and the application shall  
35677 ensure it consists of one of the following values:35678 SIG\_BLOCK The resulting set shall be the union of the current set and the signal set  
35679 pointed to by *set*.35680 SIG\_SETMASK The resulting set shall be the signal set pointed to by *set*.35681 SIG\_UNBLOCK The resulting set shall be the intersection of the current set and the  
35682 complement of the signal set pointed to by *set*.35683 If the argument *oset* is not a null pointer, the previous mask shall be stored in the location  
35684 pointed to by *oset*. If *set* is a null pointer, the value of the argument *how* is not significant and the  
35685 process' signal mask shall be unchanged; thus the call can be used to enquire about currently  
35686 blocked signals.35687 If there are any pending unblocked signals after the call to *sigprocmask()*, at least one of those  
35688 signals shall be delivered before the call to *sigprocmask()* returns.35689 It is not possible to block those signals which cannot be ignored. This shall be enforced by the  
35690 system without causing an error to be indicated.35691 If any of the SIGFPE, SIGILL, SIGSEGV, or SIGBUS signals are generated while they are blocked,  
35692 the result is undefined, unless the signal was generated by the *kill()* function, the *sigqueue()*  
35693 function, or the *raise()* function.35694 If *sigprocmask()* fails, the thread's signal mask shall not be changed.35695 The use of the *sigprocmask()* function is unspecified in a multi-threaded process.35696 **RETURN VALUE**35697 THR Upon successful completion *pthread\_sigmask()* shall return 0; otherwise, it shall return the  
35698 corresponding error number.35699 Upon successful completion, *sigprocmask()* shall return 0; otherwise, -1 shall be returned, *errno*  
35700 shall be set to indicate the error, and the process' signal mask shall be unchanged.

35701 **ERRORS**

35702 THR The `pthread_sigmask()` and `sigprocmask()` functions shall fail if:

35703 [EINVAL] The value of the *how* argument is not equal to one of the defined values.

35704 THR The `pthread_sigmask()` function shall not return an error code of [EINTR].

35705 **EXAMPLES**

35706 None.

35707 **APPLICATION USAGE**

35708 None.

35709 **RATIONALE**

35710 When a process' signal mask is changed in a signal-catching function that is installed by  
 35711 `sigaction()`, the restoration of the signal mask on return from the signal-catching function  
 35712 overrides that change (see `sigaction()`). If the signal-catching function was installed with  
 35713 `signal()`, it is unspecified whether this occurs.

35714 See `kill()` for a discussion of the requirement on delivery of signals.

35715 **FUTURE DIRECTIONS**

35716 None.

35717 **SEE ALSO**

35718 `sigaction()`, `sigaddset()`, `sigdelset()`, `sigemptyset()`, `sigfillset()`, `sigismember()`, `sigpending()`,  
 35719 `sigqueue()`, `sigsuspend()`, the Base Definitions volume of IEEE Std 1003.1-200x, <**signal.h**>

35720 **CHANGE HISTORY**

35721 First released in Issue 3.

35722 Entry included for alignment with the POSIX.1-1988 standard.

35723 **Issue 5**

35724 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

35725 The `pthread_sigmask()` function is added for alignment with the POSIX Threads Extension.

35726 **Issue 6**

35727 The `pthread_sigmask()` function is marked as part of the Threads option.

35728 The SYNOPSIS for `sigprocmask()` is marked as a CX extension to note that the presence of this  
 35729 function in the <**signal.h**> header is an extension to the ISO C standard. |

35730 The following changes are made for alignment with the ISO POSIX-1: 1996 standard: |

35731 • The DESCRIPTION is updated to explicitly state the functions which may generate the  
 35732 signal.

35733 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

35734 The **restrict** keyword is added to the `pthread_sigmask()` and `sigprocmask()` prototypes for  
 35735 alignment with the ISO/IEC 9899: 1999 standard.

35736 **NAME**

35737 pthread\_spin\_destroy, pthread\_spin\_init — destroy or initialize a spin lock object (**ADVANCED**  
 35738 **REALTIME THREADS**)

35739 **SYNOPSIS**

35740 THR SPI #include <pthread.h>

35741 int pthread\_spin\_destroy(pthread\_spinlock\_t \*lock);

35742 int pthread\_spin\_init(pthread\_spinlock\_t \*lock, int pshared);

35743

35744 **DESCRIPTION**

35745 The *pthread\_spin\_destroy()* function shall destroy the spin lock referenced by *lock* and release any  
 35746 resources used by the lock. The effect of subsequent use of the lock is undefined until the lock is  
 35747 reinitialized by another call to *pthread\_spin\_init()*. The results are undefined if  
 35748 *pthread\_spin\_destroy()* is called when a thread holds the lock, or if this function is called with an  
 35749 uninitialized thread spin lock.

35750 The *pthread\_spin\_init()* function shall allocate any resources required to use the spin lock  
 35751 referenced by *lock* and initialize the lock to an unlocked state.

35752 TSH If the Thread Process-Shared Synchronization option is supported and the value of *pshared* is  
 35753 PTHREAD\_PROCESS\_SHARED, the implementation shall permit the spin lock to be operated  
 35754 upon by any thread that has access to the memory where the spin lock is allocated, even if it is  
 35755 allocated in memory that is shared by multiple processes.

35756 If the Thread Process-Shared Synchronization option is supported and the value of *pshared* is  
 35757 PTHREAD\_PROCESS\_PRIVATE, or if the option is not supported, the spin lock shall only be  
 35758 operated upon by threads created within the same process as the thread that initialized the spin  
 35759 lock. If threads of differing processes attempt to operate on such a spin lock, the behavior is  
 35760 undefined.

35761 The results are undefined if *pthread\_spin\_init()* is called specifying an already initialized spin  
 35762 lock. The results are undefined if a spin lock is used without first being initialized.

35763 If the *pthread\_spin\_init()* function fails, the lock is not initialized and the contents of *lock* are  
 35764 undefined.

35765 Only the object referenced by *lock* may be used for performing synchronization.

35766 The result of referring to copies of that object in calls to *pthread\_spin\_destroy()*,  
 35767 *pthread\_spin\_lock()*, *pthread\_spin\_trylock()*, or *pthread\_spin\_unlock()* is undefined.

35768 **RETURN VALUE**

35769 Upon successful completion, these functions shall return zero; otherwise, an error number shall  
 35770 be returned to indicate the error.

35771 **ERRORS**

35772 These functions may fail if:

35773 [EBUSY] The implementation has detected an attempt to initialize or destroy a spin  
 35774 lock while it is in use (for example, while being used in a *pthread\_spin\_lock()*  
 35775 call) by another thread.

35776 [EINVAL] The value specified by *lock* is invalid.

35777 The *pthread\_spin\_init()* function shall fail if:

35778 [EAGAIN] The system lacks the necessary resources to initialize another spin lock.

35779 [ENOMEM] Insufficient memory exists to initialize the lock.

35780 These functions shall not return an error code of [EINTR].

## 35781 EXAMPLES

35782 None.

## 35783 APPLICATION USAGE

35784 The *pthread\_spin\_destroy()* and *pthread\_spin\_init()* functions are part of the Spin Locks option  
35785 and need not be provided on all implementations.

## 35786 RATIONALE

35787 None.

## 35788 FUTURE DIRECTIONS

35789 None.

## 35790 SEE ALSO

35791 *pthread\_spin\_lock()*, *pthread\_spin\_trylock()*, *pthread\_spin\_unlock()*, the Base Definitions volume of  
35792 IEEE Std 1003.1-200x, <<pthread.h>>

## 35793 CHANGE HISTORY

35794 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

35795 In the SYNOPSIS, the inclusion of <sys/types.h> is no longer required.

35796 **NAME**

35797 pthread\_spin\_init — initialize a spin lock object (**ADVANCED REALTIME THREADS**)

35798 **SYNOPSIS**

35799 THR SPI #include <pthread.h>

35800 int pthread\_spin\_init(pthread\_spinlock\_t \*lock, int pshared);

35801

35802 **DESCRIPTION**

35803 Refer to *pthread\_spin\_destroy()*.

## 35804 NAME

35805 pthread\_spin\_lock, pthread\_spin\_trylock — lock a spin lock object (ADVANCED REALTIME  
35806 THREADS)

## 35807 SYNOPSIS

```
35808 THR SPI #include <pthread.h>
```

```
35809 int pthread_spin_lock(pthread_spinlock_t *lock);  
35810 int pthread_spin_trylock(pthread_spinlock_t *lock);  
35811
```

## 35812 DESCRIPTION

35813 The *pthread\_spin\_lock()* function shall lock the spin lock referenced by *lock*. The calling thread  
35814 shall acquire the lock if it is not held by another thread. Otherwise, the thread shall spin (that is, |  
35815 shall not return from the *pthread\_spin\_lock()* call) until the lock becomes available. The results |  
35816 are undefined if the calling thread holds the lock at the time the call is made. The  
35817 *pthread\_spin\_trylock()* function shall lock the spin lock referenced by *lock* if it is not held by any |  
35818 thread. Otherwise, the function shall fail. |

35819 The results are undefined if any of these functions is called with an uninitialized spin lock.

## 35820 RETURN VALUE

35821 Upon successful completion, these functions shall return zero; otherwise, an error number shall  
35822 be returned to indicate the error.

## 35823 ERRORS

35824 These functions may fail if:

35825 [EINVAL] The value specified by *lock* does not refer to an initialized spin lock object.

35826 The *pthread\_spin\_lock()* function may fail if:

35827 [EDEADLK] The calling thread already holds the lock.

35828 The *pthread\_spin\_trylock()* function shall fail if:

35829 [EBUSY] A thread currently holds the lock.

35830 These functions shall not return an error code of [EINTR].

## 35831 EXAMPLES

35832 None.

## 35833 APPLICATION USAGE

35834 Applications using this function may be subject to priority inversion, as discussed in the Base  
35835 Definitions volume of IEEE Std 1003.1-200x, Section 3.285, Priority Inversion.

35836 The *pthread\_spin\_lock()* and *pthread\_spin\_trylock()* functions are part of the Spin Locks option  
35837 and need not be provided on all implementations.

## 35838 RATIONALE

35839 None.

## 35840 FUTURE DIRECTIONS

35841 None.

## 35842 SEE ALSO

35843 *pthread\_spin\_init()*, *pthread\_spin\_destroy()*, *pthread\_spin\_unlock()*, the Base Definitions volume of  
35844 IEEE Std 1003.1-200x, <pthread.h>

35845 **CHANGE HISTORY**

35846 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

35847 In the SYNOPSIS, the inclusion of `<sys/types.h>` is no longer required.

35848 **NAME**

35849 pthread\_spin\_trylock — lock a spin lock object (**ADVANCED REALTIME THREADS**)

35850 **SYNOPSIS**

35851 THR SPI #include <pthread.h>

35852 int pthread\_spin\_trylock(pthread\_spinlock\_t \*lock);

35853

35854 **DESCRIPTION**

35855 Refer to *pthread\_spin\_lock()*.

35856 **NAME**

35857 pthread\_spin\_unlock — unlock a spin lock object (**ADVANCED REALTIME THREADS**)

35858 **SYNOPSIS**

35859 THR SPI #include <pthread.h>

35860 int pthread\_spin\_unlock(pthread\_spinlock\_t \*lock);

35861

35862 **DESCRIPTION**

35863 The *pthread\_spin\_unlock()* function shall release the spin lock referenced by *lock* which was  
35864 locked via the *pthread\_spin\_lock()* or *pthread\_spin\_trylock()* functions. The results are undefined if  
35865 the lock is not held by the calling thread. If there are threads spinning on the lock when  
35866 *pthread\_spin\_unlock()* is called, the lock becomes available and an unspecified spinning thread  
35867 shall acquire the lock.

35868 The results are undefined if this function is called with an uninitialized thread spin lock.

35869 **RETURN VALUE**

35870 Upon successful completion, the *pthread\_spin\_unlock()* function shall return zero; otherwise, an  
35871 error number shall be returned to indicate the error.

35872 **ERRORS**

35873 The *pthread\_spin\_unlock()* function may fail if:

35874 [EINVAL] An invalid argument was specified.

35875 [EPERM] The calling thread does not hold the lock.

35876 This function shall not return an error code of [EINTR].

35877 **EXAMPLES**

35878 None.

35879 **APPLICATION USAGE**

35880 The *pthread\_spin\_unlock()* function is part of the Spin Locks option and need not be provided on  
35881 all implementations.

35882 **RATIONALE**

35883 None.

35884 **FUTURE DIRECTIONS**

35885 None.

35886 **SEE ALSO**

35887 *pthread\_spin\_init()*, *pthread\_spin\_destroy()*, *pthread\_spin\_lock()*, *pthread\_spin\_trylock()*, the Base  
35888 Definitions volume of IEEE Std 1003.1-200x, <pthread.h>

35889 **CHANGE HISTORY**

35890 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

35891 In the SYNOPSIS, the inclusion of <sys/types.h> is no longer required.

35892 **NAME**

35893 pthread\_testcancel — set cancelability state

35894 **SYNOPSIS**

35895 THR #include <pthread.h>

35896 void pthread\_testcancel(void);

35897

35898 **DESCRIPTION**

35899 Refer to *pthread\_setcancelstate()*.

35900 **NAME**

35901 ptsname — get name of the slave pseudo-terminal device

35902 **SYNOPSIS**

35903 XSI #include &lt;stdlib.h&gt;

35904 char \*ptsname(int *fildev*);

35905

35906 **DESCRIPTION**

35907 The *ptsname()* function shall return the name of the slave pseudo-terminal device associated  
35908 with a master pseudo-terminal device. The *fildev* argument is a file descriptor that refers to the  
35909 master device. The *ptsname()* function shall return a pointer to a string containing the pathname  
35910 of the corresponding slave device.

35911 The *ptsname()* function need not be reentrant. A function that is not required to be reentrant is  
35912 not required to be thread-safe.

35913 **RETURN VALUE**

35914 Upon successful completion, *ptsname()* shall return a pointer to a string which is the name of the  
35915 pseudo-terminal slave device. Upon failure, *ptsname()* shall return a null pointer. This could  
35916 occur if *fildev* is an invalid file descriptor or if the slave device name does not exist in the file  
35917 system.

35918 **ERRORS**

35919 No errors are defined.

35920 **EXAMPLES**

35921 None.

35922 **APPLICATION USAGE**35923 The value returned may point to a static data area that is overwritten by each call to *ptsname()*.35924 **RATIONALE**

35925 None.

35926 **FUTURE DIRECTIONS**

35927 None.

35928 **SEE ALSO**

35929 *grantpt()*, *open()*, *ttyname()*, *unlockpt()*, the Base Definitions volume of IEEE Std 1003.1-200x,  
35930 <stdlib.h>

35931 **CHANGE HISTORY**

35932 First released in Issue 4, Version 2.

35933 **Issue 5**

35934 Moved from X/OPEN UNIX extension to BASE.

35935 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

35936 **NAME**

35937       putc — put byte on a stream

35938 **SYNOPSIS**

35939       #include &lt;stdio.h&gt;

35940       int putc(int *c*, FILE \**stream*);35941 **DESCRIPTION**

35942 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
35943 conflict between the requirements described here and the ISO C standard is unintentional. This  
35944 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

35945       The *putc()* function shall be equivalent to *fputc()*, except that if it is implemented as a macro it  
35946 may evaluate *stream* more than once, so the argument should never be an expression with side  
35947 effects.

35948 **RETURN VALUE**35949       Refer to *fputc()*.35950 **ERRORS**35951       Refer to *fputc()*.35952 **EXAMPLES**

35953       None.

35954 **APPLICATION USAGE**

35955       Since it may be implemented as a macro, *putc()* may treat a *stream* argument with side effects |  
35956 incorrectly. In particular, *putc(c,\*f++)* does not necessarily work correctly. Therefore, use of this  
35957 function is not recommended in such situations; *fputc()* should be used instead.

35958 **RATIONALE**

35959       None.

35960 **FUTURE DIRECTIONS**

35961       None.

35962 **SEE ALSO**35963       *fputc()*, the Base Definitions volume of IEEE Std 1003.1-200x, <stdio.h>35964 **CHANGE HISTORY**

35965       First released in Issue 1. Derived from Issue 1 of the SVID.

35966 **NAME**35967        `putc_unlocked` — stdio with explicit client locking35968 **SYNOPSIS**35969 TSF        `#include <stdio.h>`35970        `int putc_unlocked(int c, FILE *stream);`

35971

35972 **DESCRIPTION**35973        Refer to `getc_unlocked()`.

35974 **NAME**

35975 putchar — put byte on stdout stream

35976 **SYNOPSIS**

35977 #include <stdio.h>

35978 int putchar(int c);

35979 **DESCRIPTION**

35980 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
35981 conflict between the requirements described here and the ISO C standard is unintentional. This  
35982 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

35983 The function call *putchar(c)* shall be equivalent to *putc(c,stdout)*.

35984 **RETURN VALUE**

35985 Refer to *fputc()*.

35986 **ERRORS**

35987 Refer to *fputc()*.

35988 **EXAMPLES**

35989 None.

35990 **APPLICATION USAGE**

35991 None.

35992 **RATIONALE**

35993 None.

35994 **FUTURE DIRECTIONS**

35995 None.

35996 **SEE ALSO**

35997 *putc()*, the Base Definitions volume of IEEE Std 1003.1-200x, <stdio.h>

35998 **CHANGE HISTORY**

35999 First released in Issue 1. Derived from Issue 1 of the SVID.

36000 **NAME**

36001 putchar\_unlocked — stdio with explicit client locking

36002 **SYNOPSIS**36003 TSF `#include <stdio.h>`36004 `int putchar_unlocked(int c);`

36005

36006 **DESCRIPTION**36007 Refer to *getc\_unlocked()*.

36008 **NAME**

36009 putenv — change or add a value to environment

36010 **SYNOPSIS**

36011 xSI #include &lt;stdlib.h&gt;

36012 int putenv(char \*string);

36013

36014 **DESCRIPTION**

36015 The *putenv()* function shall use the *string* argument to set environment variable values. The |  
 36016 *string* argument should point to a string of the form "*name=value*". The *putenv()* function shall |  
 36017 make the value of the environment variable *name* equal to *value* by altering an existing variable |  
 36018 or creating a new one. In either case, the string pointed to by *string* shall become part of the |  
 36019 environment, so altering the string shall change the environment. The space used by *string* is no |  
 36020 longer used once a new string-defining *name* is passed to *putenv()*.

36021 The *putenv()* function need not be reentrant. A function that is not required to be reentrant is not  
 36022 required to be thread-safe.

36023 **RETURN VALUE**

36024 Upon successful completion, *putenv()* shall return 0; otherwise, it shall return a non-zero value  
 36025 and set *errno* to indicate the error.

36026 **ERRORS**36027 The *putenv()* function may fail if:

36028 [ENOMEM] Insufficient memory was available.

36029 **EXAMPLES**36030 **Changing the Value of an Environment Variable**

36031 The following example changes the value of the *HOME* environment variable to the value  
 36032 */usr/home*.

36033 #include &lt;stdlib.h&gt;

36034 ...

36035 static char \*var = "HOME=/usr/home";

36036 int ret;

36037 ret = putenv(var);

36038 **APPLICATION USAGE**

36039 The *putenv()* function manipulates the environment pointed to by *environ*, and can be used in  
 36040 conjunction with *getenv()*.

36041 This routine may use *malloc()* to enlarge the environment.

36042 A potential error is to call *putenv()* with an automatic variable as the argument, then return from  
 36043 the calling function while *string* is still part of the environment.

36044 The *setenv()* function is preferred over this function.36045 **RATIONALE**

36046 The standard developers noted that *putenv()* is the only function available to add to the |  
 36047 environment without permitting memory leaks. |

36048 **FUTURE DIRECTIONS**

36049 None.

36050 **SEE ALSO**36051 *exec*, *getenv()*, *malloc()*, *setenv()*, the Base Definitions volume of IEEE Std 1003.1-200x, <**stdlib.h**>36052 **CHANGE HISTORY**

36053 First released in Issue 1. Derived from Issue 1 of the SVID.

36054 **Issue 5**36055 The type of the argument to this function is changed from **const char \*** to **char \***. This was indicated as a FUTURE DIRECTION in previous issues.

36057 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

## 36058 NAME

36059 putmsg, putpmsg — send a message on a STREAM (STREAMS)

## 36060 SYNOPSIS

36061 XSR #include &lt;stropts.h&gt;

```

36062 int putmsg(int fildes, const struct strbuf *ctlptr,
36063           const struct strbuf *dataptr, int flags);
36064 int putpmsg(int fildes, const struct strbuf *ctlptr,
36065            const struct strbuf *dataptr, int band, int flags);
36066

```

## 36067 DESCRIPTION

36068 The *putmsg()* function shall create a message from a process buffer(s) and send the message to a  
 36069 STREAMS file. The message may contain either a data part, a control part, or both. The data and  
 36070 control parts are distinguished by placement in separate buffers, as described below. The  
 36071 semantics of each part are defined by the STREAMS module that receives the message.

36072 The *putpmsg()* function is equivalent to *putmsg()*, except that the process can send messages in  
 36073 different priority bands. Except where noted, all requirements on *putmsg()* also pertain to  
 36074 *putpmsg()*.

36075 The *fildes* argument specifies a file descriptor referencing an open STREAM. The *ctlptr* and  
 36076 *dataptr* arguments each point to a **strbuf** structure.

36077 The *ctlptr* argument points to the structure describing the control part, if any, to be included in  
 36078 the message. The *buf* member in the **strbuf** structure points to the buffer where the control  
 36079 information resides, and the *len* member indicates the number of bytes to be sent. The *maxlen*  
 36080 member is not used by *putmsg()*. In a similar manner, the argument *dataptr* specifies the data, if  
 36081 any, to be included in the message. The *flags* argument indicates what type of message should be  
 36082 sent and is described further below.

36083 To send the data part of a message, the application shall ensure that *dataptr* is not a null pointer  
 36084 and the *len* member of *dataptr* is 0 or greater. To send the control part of a message, the  
 36085 application shall ensure that the corresponding values are set for *ctlptr*. No data (control) part  
 36086 shall be sent if either *dataptr(ctlptr)* is a null pointer or the *len* member of *dataptr(ctlptr)* is set to  
 36087 -1.

36088 For *putmsg()*, if a control part is specified and *flags* is set to RS\_HIPRI, a high priority message  
 36089 shall be sent. If no control part is specified, and *flags* is set to RS\_HIPRI, *putmsg()* shall fail and  
 36090 set *errno* to [EINVAL]. If *flags* is set to 0, a normal message (priority band equal to 0) shall be  
 36091 sent. If a control part and data part are not specified and *flags* is set to 0, no message shall be  
 36092 sent and 0 shall be returned.

36093 For *putpmsg()*, the flags are different. The *flags* argument is a bitmask with the following  
 36094 mutually-exclusive flags defined: MSG\_HIPRI and MSG\_BAND. If *flags* is set to 0, *putpmsg()*  
 36095 shall fail and set *errno* to [EINVAL]. If a control part is specified and *flags* is set to MSG\_HIPRI  
 36096 and *band* is set to 0, a high-priority message shall be sent. If *flags* is set to MSG\_HIPRI and either  
 36097 no control part is specified or *band* is set to a non-zero value, *putpmsg()* shall fail and set *errno* to  
 36098 [EINVAL]. If *flags* is set to MSG\_BAND, then a message shall be sent in the priority band  
 36099 specified by *band*. If a control part and data part are not specified and *flags* is set to MSG\_BAND,  
 36100 no message shall be sent and 0 shall be returned.

36101 The *putmsg()* function shall block if the STREAM write queue is full due to internal flow control  
 36102 conditions, with the following exceptions:

- 36103 • For high-priority messages, *putmsg()* shall not block on this condition and continues  
 36104 processing the message.

36105           • For other messages, *putmsg()* shall not block but shall fail when the write queue is full and  
36106            O\_NONBLOCK is set. |

36107           The *putmsg()* function shall also block, unless prevented by lack of internal resources, while |  
36108            waiting for the availability of message blocks in the STREAM, regardless of priority or whether |  
36109            O\_NONBLOCK has been specified. No partial message shall be sent. |

#### 36110 RETURN VALUE

36111           Upon successful completion, *putmsg()* and *putpmsg()* shall return 0; otherwise, they shall return  
36112            -1 and set *errno* to indicate the error.

#### 36113 ERRORS

36114           The *putmsg()* and *putpmsg()* functions shall fail if:

36115           [EAGAIN]        A non-priority message was specified, the O\_NONBLOCK flag is set, and the  
36116            STREAM write queue is full due to internal flow control conditions; or buffers  
36117            could not be allocated for the message that was to be created.

36118           [EBADF]        *fildev* is not a valid file descriptor open for writing.

36119           [EINTR]        A signal was caught during *putmsg()*.

36120           [EINVAL]       An undefined value is specified in *flags*, or *flags* is set to RS\_HIPRI or  
36121            MSG\_HIPRI and no control part is supplied, or the STREAM or multiplexer  
36122            referenced by *fildev* is linked (directly or indirectly) downstream from a  
36123            multiplexer, or *flags* is set to MSG\_HIPRI and *band* is non-zero (for *putpmsg()*  
36124            only).

36125           [ENOSR]        Buffers could not be allocated for the message that was to be created due to  
36126            insufficient STREAMS memory resources.

36127           [ENOSTR]       A STREAM is not associated with *fildev*.

36128           [ENXIO]        A hangup condition was generated downstream for the specified STREAM.

36129           [EPIPE] or [EIO] The *fildev* argument refers to a STREAMS-based pipe and the other end of the  
36130            pipe is closed. A SIGPIPE signal is generated for the calling thread.

36131           [ERANGE]        The size of the data part of the message does not fall within the range  
36132            specified by the maximum and minimum packet sizes of the topmost  
36133            STREAM module. This value is also returned if the control part of the message  
36134            is larger than the maximum configured size of the control part of a message,  
36135            or if the data part of a message is larger than the maximum configured size of  
36136            the data part of a message.

36137           In addition, *putmsg()* and *putpmsg()* shall fail if the STREAM head had processed an  
36138            asynchronous error before the call. In this case, the value of *errno* does not reflect the result of  
36139            *putmsg()* or *putpmsg()*, but reflects the prior error. |

## 36140 EXAMPLES

36141 **Sending a High-Priority Message**

36142 The value of *fd* is assumed to refer to an open STREAMS file. This call to *putmsg()* does the  
36143 following:

- 36144 1. Creates a high-priority message with a control part and a data part, using the buffers  
36145 pointed to by *ctrlbuf* and *databuf*, respectively.
- 36146 2. Sends the message to the STREAMS file identified by *fd*.

```
36147 #include <stropts.h>
36148 #include <string.h>
36149 ...
36150 int fd;
36151 char *ctrlbuf = "This is the control part";
36152 char *databuf = "This is the data part";
36153 struct strbuf ctrl;
36154 struct strbuf data;
36155 int ret;

36156 ctrl.buf = ctrlbuf;
36157 ctrl.len = strlen(ctrlbuf);

36158 data.buf = databuf;
36159 data.len = strlen(databuf);

36160 ret = putmsg(fd, &ctrl, &data, MSG_HIPRI);
```

36161 **Using putpmsg()**

36162 This example has the same effect as the previous example. In this example, however, the  
36163 *putpmsg()* function creates and sends the message to the STREAMS file.

```
36164 #include <stropts.h>
36165 #include <string.h>
36166 ...
36167 int fd;
36168 char *ctrlbuf = "This is the control part";
36169 char *databuf = "This is the data part";
36170 struct strbuf ctrl;
36171 struct strbuf data;
36172 int ret;

36173 ctrl.buf = ctrlbuf;
36174 ctrl.len = strlen(ctrlbuf);

36175 data.buf = databuf;
36176 data.len = strlen(databuf);

36177 ret = putpmsg(fd, &ctrl, &data, 0, MSG_HIPRI);
```

36178 **APPLICATION USAGE**

36179 None.

36180 **RATIONALE**

36181 None.

36182 **FUTURE DIRECTIONS**

36183 None.

36184 **SEE ALSO**36185 *getmsg()*, *poll()*, *read()*, *write()*, the Base Definitions volume of IEEE Std 1003.1-200x, <**stropts.h**>,  
36186 Section 2.6 (on page 488)36187 **CHANGE HISTORY**

36188 First released in Issue 4, Version 2.

36189 **Issue 5**

36190 Moved from X/OPEN UNIX extension to BASE.

36191 The following text is removed from the DESCRIPTION: “The STREAM head guarantees that the  
36192 control part of a message generated by *putmsg()* is at least 64 bytes in length”.36193 **Issue 6**

36194 This function is marked as part of the XSI STREAMS Option Group.

36195 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

36196 **NAME**

36197       putpmsg — send a message on a STREAM (**STREAMS**)

36198 **SYNOPSIS**

36199 xSR     #include <stropts.h>

```
36200     int putpmsg(int fildev, const struct strbuf *ctlptr,  
36201                const struct strbuf *dataptr, int band, int flags);
```

36202

36203 **DESCRIPTION**

36204       Refer to *putmsg()*.

36205 **NAME**

36206 puts — put a string on standard output

36207 **SYNOPSIS**

36208 #include &lt;stdio.h&gt;

36209 int puts(const char \*s);

36210 **DESCRIPTION**

36211 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
 36212 conflict between the requirements described here and the ISO C standard is unintentional. This  
 36213 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

36214 The *puts()* function shall write the string pointed to by *s*, followed by a <newline>, to the  
 36215 standard output stream *stdout*. The terminating null byte shall not be written.

36216 cx The *st\_ctime* and *st\_mtime* fields of the file shall be marked for update between the successful  
 36217 execution of *puts()* and the next successful completion of a call to *fflush()* or *fclose()* on the same  
 36218 stream or a call to *exit()* or *abort()*.

36219 **RETURN VALUE**

36220 Upon successful completion, *puts()* shall return a non-negative number. Otherwise, it shall  
 36221 cx return EOF, shall set an error indicator for the stream, and *errno* shall be set to indicate the error.

36222 **ERRORS**36223 Refer to *fputc()*.36224 **EXAMPLES**36225 **Printing to Standard Output**

36226 The following example gets the current time, converts it to a string using *localtime()* and  
 36227 *asctime()*, and prints it to standard output using *puts()*. It then prints the number of minutes to  
 36228 an event for which it is waiting.

```

36229 #include <time.h>
36230 #include <stdio.h>
36231 ...
36232 time_t now;
36233 int minutes_to_event;
36234 ...
36235 time(&now);
36236 printf("The time is ");
36237 puts(asctime(localtime(&now)));
36238 printf("There are %d minutes to the event.\n",
36239     minutes_to_event);
36240 ...

```

36241 **APPLICATION USAGE**36242 The *puts()* function appends a <newline>, while *fputs()* does not.36243 **RATIONALE**

36244 None.

36245 **FUTURE DIRECTIONS**

36246 None.

36247 **SEE ALSO**

36248 *fopen()*, *fputs()*, *putc()*, the Base Definitions volume of IEEE Std 1003.1-200x, <**stdio.h**>

36249 **CHANGE HISTORY**

36250 First released in Issue 1. Derived from Issue 1 of the SVID.

36251 **Issue 6**

36252 Extensions beyond the ISO C standard are now marked.

36253 **NAME**

36254 pututxline — put an entry into user accounting database

36255 **SYNOPSIS**36256 XSI `#include <utmpx.h>`36257 `struct utmpx *pututxline(const struct utmpx *utmpx);`

36258

36259 **DESCRIPTION**36260 Refer to *endutxent()*.

36261 **NAME**

36262 putwc — put a wide character on a stream

36263 **SYNOPSIS**

36264 #include &lt;stdio.h&gt;

36265 #include &lt;wchar.h&gt;

36266 wint\_t putwc(wchar\_t *wc*, FILE \**stream*);36267 **DESCRIPTION**

36268 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
36269 conflict between the requirements described here and the ISO C standard is unintentional. This  
36270 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

36271 The *putwc()* function shall be equivalent to *fputwc()*, except that if it is implemented as a macro  
36272 it may evaluate *stream* more than once, so the argument should never be an expression with side  
36273 effects.

36274 **RETURN VALUE**36275 Refer to *fputwc()*.36276 **ERRORS**36277 Refer to *fputwc()*.36278 **EXAMPLES**

36279 None.

36280 **APPLICATION USAGE**

36281 Since it may be implemented as a macro, *putwc()* may treat a *stream* argument with side effects |  
36282 incorrectly. In particular, *putwc(wc,\*f++)* need not work correctly. Therefore, use of this function  
36283 is not recommended; *fputwc()* should be used instead.

36284 **RATIONALE**

36285 None.

36286 **FUTURE DIRECTIONS**

36287 None.

36288 **SEE ALSO**36289 *fputwc()*, the Base Definitions volume of IEEE Std 1003.1-200x, <stdio.h>, <wchar.h>36290 **CHANGE HISTORY**

36291 First released as a World-wide Portability Interface in Issue 4.

36292 **Issue 5**

36293 Aligned with ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, the type of argument *wc*  
36294 is changed from **wint\_t** to **wchar\_t**.

36295 The Optional Header (OH) marking is removed from &lt;stdio.h&gt;.

36296 **NAME**

36297 putwchar — put a wide character on stdout stream

36298 **SYNOPSIS**

36299 #include <wchar.h>

36300 wint\_t putwchar(wchar\_t wc);

36301 **DESCRIPTION**

36302 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
36303 conflict between the requirements described here and the ISO C standard is unintentional. This  
36304 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

36305 The function call *putwchar(wc)* shall be equivalent to *putwc(wc,stdout)*.

36306 **RETURN VALUE**

36307 Refer to *fputwc()*.

36308 **ERRORS**

36309 Refer to *fputwc()*.

36310 **EXAMPLES**

36311 None.

36312 **APPLICATION USAGE**

36313 None.

36314 **RATIONALE**

36315 None.

36316 **FUTURE DIRECTIONS**

36317 None.

36318 **SEE ALSO**

36319 *fputwc()*, *putwc()*, the Base Definitions volume of IEEE Std 1003.1-200x, <wchar.h>

36320 **CHANGE HISTORY**

36321 First released in Issue 4.

36322 **Issue 5**

36323 Aligned with ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, the type of argument *wc*  
36324 is changed from **wint\_t** to **wchar\_t**.

36325 **NAME**

36326           pwrite — write on a file

36327 **SYNOPSIS**

36328           #include <unistd.h>

```
36329 xSI       ssize_t pwrite(int fildev, const void *buf, size_t nbyte,  
36330               off_t offset);
```

36331

36332 **DESCRIPTION**

36333           Refer to *write()*.

36334 **NAME**

36335 qsort — sort a table of data

36336 **SYNOPSIS**

36337 #include &lt;stdlib.h&gt;

36338 void qsort(void \*base, size\_t nel, size\_t width,  
36339 int (\*compar)(const void \*, const void \*));36340 **DESCRIPTION**36341 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
36342 conflict between the requirements described here and the ISO C standard is unintentional. This  
36343 volume of IEEE Std 1003.1-200x defers to the ISO C standard.36344 The *qsort()* function shall sort an array of *nel* objects, the initial element of which is pointed to by  
36345 *base*. The size of each object, in bytes, is specified by the *width* argument.36346 The contents of the array shall be sorted in ascending order according to a comparison function.  
36347 The *compar* argument is a pointer to the comparison function, which is called with two  
36348 arguments that point to the elements being compared. The application shall ensure that the  
36349 function returns an integer less than, equal to, or greater than 0, if the first argument is  
36350 considered respectively less than, equal to, or greater than the second. If two members compare  
36351 as equal, their order in the sorted array is unspecified.36352 **RETURN VALUE**36353 The *qsort()* function shall not return a value.36354 **ERRORS**

36355 No errors are defined.

36356 **EXAMPLES**

36357 None.

36358 **APPLICATION USAGE**36359 The comparison function need not compare every byte, so arbitrary data may be contained in  
36360 the elements in addition to the values being compared.36361 **RATIONALE**

36362 None.

36363 **FUTURE DIRECTIONS**

36364 None.

36365 **SEE ALSO**

36366 The Base Definitions volume of IEEE Std 1003.1-200x, &lt;stdlib.h&gt;

36367 **CHANGE HISTORY**

36368 First released in Issue 1. Derived from Issue 1 of the SVID.

36369 **Issue 6**

36370 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

36371 **NAME**

36372 raise — send a signal to the executing process

36373 **SYNOPSIS**

36374 #include <signal.h>

36375 int raise(int sig);

36376 **DESCRIPTION**

36377 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 36378 conflict between the requirements described here and the ISO C standard is unintentional. This  
 36379 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

36380 CX The *raise()* function shall send the signal *sig* to the executing thread or process. If a signal  
 36381 handler is called, the *raise()* function shall not return until after the signal handler does.

36382 THR If the implementation supports the Threads option, the effect of the *raise()* function shall be  
 36383 equivalent to calling:

36384 pthread\_kill(pthread\_self(), sig);

36385

36386 CX Otherwise, the effect of the *raise()* function shall be equivalent to calling:

36387 kill(getpid(), sig);

36388

36389 **RETURN VALUE**

36390 CX Upon successful completion, 0 shall be returned. Otherwise, a non-zero value shall be returned  
 36391 and *errno* shall be set to indicate the error.

36392 **ERRORS**

36393 The *raise()* function shall fail if:

36394 CX [EINVAL] The value of the *sig* argument is an invalid signal number.

36395 **EXAMPLES**

36396 None.

36397 **APPLICATION USAGE**

36398 None.

36399 **RATIONALE**

36400 The term “thread” is an extension to the ISO C standard.

36401 **FUTURE DIRECTIONS**

36402 None.

36403 **SEE ALSO**

36404 *kill()*, *sigaction()*, the Base Definitions volume of IEEE Std 1003.1-200x, <signal.h>,  
 36405 <sys/types.h>

36406 **CHANGE HISTORY**

36407 First released in Issue 4. Derived from the ANSI C standard.

36408 **Issue 5**

36409 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

36410 **Issue 6**

36411 Extensions beyond the ISO C standard are now marked.

36412 The following new requirements on POSIX implementations derive from alignment with the  
36413 Single UNIX Specification:

- 36414 • In the RETURN VALUE section, the requirement to set *errno* on error is added.
- 36415 • The [EINVAL] error condition is added.

36416 **NAME**

36417 rand, rand\_r, srand — pseudo-random number generator

36418 **SYNOPSIS**

36419 #include &lt;stdlib.h&gt;

36420 int rand(void);

36421 TSF int rand\_r(unsigned \*seed);

36422 void srand(unsigned seed);

36423 **DESCRIPTION**

36424 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 36425 conflict between the requirements described here and the ISO C standard is unintentional. This  
 36426 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

36427 The *rand()* function shall compute a sequence of pseudo-random integers in the range 0 to  
 36428 XSI {RAND\_MAX} with a period of at least  $2^{32}$ .

36429 CX The *rand()* function need not be reentrant. A function that is not required to be reentrant is not  
 36430 required to be thread-safe.

36431 TSF The *rand\_r()* function shall compute a sequence of pseudo-random integers in the range 0 to  
 36432 {RAND\_MAX}. (The value of the {RAND\_MAX} macro shall be at least 32 767.)

36433 If *rand\_r()* is called with the same initial value for the object pointed to by *seed* and that object is  
 36434 not modified between successive returns and calls to *rand\_r()*, the same sequence shall be  
 36435 generated.

36436 The *srand()* function uses the argument as a seed for a new sequence of pseudo-random  
 36437 numbers to be returned by subsequent calls to *rand()*. If *srand()* is then called with the same  
 36438 seed value, the sequence of pseudo-random numbers shall be repeated. If *rand()* is called before  
 36439 any calls to *srand()* are made, the same sequence shall be generated as when *srand()* is first  
 36440 called with a seed value of 1.

36441 The implementation shall behave as if no function defined in this volume of  
 36442 IEEE Std 1003.1-200x calls *rand()* or *srand()*.

36443 **RETURN VALUE**36444 The *rand()* function shall return the next pseudo-random number in the sequence.36445 TSF The *rand\_r()* function shall return a pseudo-random integer.36446 The *srand()* function shall not return a value.36447 **ERRORS**

36448 No errors are defined.

36449 **EXAMPLES**36450 **Generating a Pseudo-Random Number Sequence**

36451 The following example demonstrates how to generate a sequence of pseudo-random numbers.

36452 #include &lt;stdio.h&gt;

36453 #include &lt;stdlib.h&gt;

36454 ...

36455 long count, i;

36456 char \*keyst;

36457 int elementlen, len;

36458 char c;

```

36459     ...
36460     /* Initial random number generator. */
36461     srand(1);

36462     /* Create keys using only lower case characters */
36463     len = 0;
36464     for (i=0; i<count; i++) {
36465         while (len < elementlen) {
36466             c = (char) (rand() % 128);
36467             if (islower(c))
36468                 keystr[len++] = c;
36469         }

36470         keystr[len] = '\0';
36471         printf("%s Element%0*ld\n", keystr, elementlen, i);
36472         len = 0;
36473     }

```

### 36474 **Generating the Same Sequence on Different Machines**

36475 The following code defines a pair of functions that could be incorporated into applications  
 36476 wishing to ensure that the same sequence of numbers is generated across different machines.

```

36477     static unsigned long next = 1;
36478     int myrand(void) /* RAND_MAX assumed to be 32767. */
36479     {
36480         next = next * 1103515245 + 12345;
36481         return((unsigned)(next/65536) % 32768);
36482     }

36483     void mysrand(unsigned seed)
36484     {
36485         next = seed;
36486     }

```

### 36487 **APPLICATION USAGE**

36488 The *drand48()* function provides a much more elaborate random number generator.

### 36489 **RATIONALE**

36490 The ISO C standard *rand()* and *srand()* functions allow per-process pseudo-random streams  
 36491 shared by all threads. Those two functions need not change, but there has to be mutual-  
 36492 exclusion that prevents interference between two threads concurrently accessing the random  
 36493 number generator.

36494 With regard to *rand()*, there are two different behaviors that may be wanted in a multi-threaded  
 36495 program:

- 36496 1. A single per-process sequence of pseudo-random numbers that is shared by all threads  
 36497 that call *rand()*
- 36498 2. A different sequence of pseudo-random numbers for each thread that calls *rand()*

36499 This is provided by the modified thread-safe function based on whether the seed value is global  
 36500 to the entire process or local to each thread.

36501 This does not address the known deficiencies of the *rand()* function implementations, which  
 36502 have been approached by maintaining more state. In effect, this specifies new thread-safe forms  
 36503 of a deficient function.

36504 **FUTURE DIRECTIONS**

36505           None.

36506 **SEE ALSO**36507           *drand48()*, the Base Definitions volume of IEEE Std 1003.1-200x, <**stdlib.h**>36508 **CHANGE HISTORY**

36509           First released in Issue 1. Derived from Issue 1 of the SVID.

36510 **Issue 5**36511           The *rand\_r()* function is included for alignment with the POSIX Threads Extension.36512           A note indicating that the *rand()* function need not be reentrant is added to the DESCRIPTION.36513 **Issue 6**

36514           Extensions beyond the ISO C standard are now marked.

36515           The *rand\_r()* function is marked as part of the Thread-Safe Functions option.

36516 **NAME**

36517 random — generate pseudo-random number

36518 **SYNOPSIS**36519 xSI `#include <stdlib.h>`36520 `long random(void);`

36521

36522 **DESCRIPTION**36523 Refer to *initstate()*.

36524 **NAME**

36525 pread, read — read from a file

36526 **SYNOPSIS**

36527 #include &lt;unistd.h&gt;

36528 XSI `ssize_t pread(int fildev, void *buf, size_t nbyte, off_t offset);`36529 `ssize_t read(int fildev, void *buf, size_t nbyte);`36530 **DESCRIPTION**

36531 The `read()` function shall attempt to read *nbyte* bytes from the file associated with the open file  
 36532 descriptor, *fildev*, into the buffer pointed to by *buf*. The behavior of multiple concurrent reads on  
 36533 the same pipe, FIFO, or terminal device is unspecified.

36534 Before any action described below is taken, and if *nbyte* is zero, the `read()` function may detect  
 36535 and return errors as described below. In the absence of errors, or if error detection is not  
 36536 performed, the `read()` function shall return zero and have no other results.

36537 On files that support seeking (for example, a regular file), the `read()` shall start at a position in  
 36538 the file given by the file offset associated with *fildev*. The file offset shall be incremented by the  
 36539 number of bytes actually read.

36540 Files that do not support seeking—for example, terminals—always read from the current  
 36541 position. The value of a file offset associated with such a file is undefined.

36542 No data transfer shall occur past the current end-of-file. If the starting position is at or after the  
 36543 end-of-file, 0 shall be returned. If the file refers to a device special file, the result of subsequent  
 36544 `read()` requests is implementation-defined.

36545 If the value of *nbyte* is greater than {SSIZE\_MAX}, the result is implementation-defined.

36546 When attempting to read from an empty pipe or FIFO:

- 36547 • If no process has the pipe open for writing, `read()` shall return 0 to indicate end-of-file.
- 36548 • If some process has the pipe open for writing and O\_NONBLOCK is set, `read()` shall return  
 36549 -1 and set *errno* to [EAGAIN].
- 36550 • If some process has the pipe open for writing and O\_NONBLOCK is clear, `read()` shall block  
 36551 the calling thread until some data is written or the pipe is closed by all processes that had the  
 36552 pipe open for writing.

36553 When attempting to read a file (other than a pipe or FIFO) that supports non-blocking reads and  
 36554 has no data currently available:

- 36555 • If O\_NONBLOCK is set, `read()` shall return -1 and set *errno* to [EAGAIN].
- 36556 • If O\_NONBLOCK is clear, `read()` shall block the calling thread until some data becomes  
 36557 available.
- 36558 • The use of the O\_NONBLOCK flag has no effect if there is some data available.

36559 The `read()` function reads data previously written to a file. If any portion of a regular file prior to  
 36560 the end-of-file has not been written, `read()` shall return bytes with value 0. For example, `lseek()`  
 36561 allows the file offset to be set beyond the end of existing data in the file. If data is later written at  
 36562 this point, subsequent reads in the gap between the previous end of data and the newly written  
 36563 data shall return bytes with value 0 until data is written into the gap.

36564 Upon successful completion, where *nbyte* is greater than 0, `read()` shall mark for update the  
 36565 *st\_atime* field of the file, and shall return the number of bytes read. This number shall never be  
 36566 greater than *nbyte*. The value returned may be less than *nbyte* if the number of bytes left in the

36567 file is less than *nbyte*, if the *read()* request was interrupted by a signal, or if the file is a pipe or  
 36568 FIFO or special file and has fewer than *nbyte* bytes immediately available for reading. For  
 36569 example, a *read()* from a file associated with a terminal may return one typed line of data.

36570 If a *read()* is interrupted by a signal before it reads any data, it shall return  $-1$  with *errno* set to  
 36571 [EINTR].

36572 If a *read()* is interrupted by a signal after it has successfully read some data, it shall return the  
 36573 number of bytes read.

36574 For regular files, no data transfer shall occur past the offset maximum established in the open  
 36575 file description associated with *fildev*.

36576 If *fildev* refers to a socket, *read()* shall be equivalent to *recv()* with no flags set. |

36577 SIO If the O\_DSYNC and O\_RSYNC bits have been set, read I/O operations on the file descriptor |  
 36578 shall complete as defined by synchronized I/O data integrity completion. If the O\_SYNC and |  
 36579 O\_RSYNC bits have been set, read I/O operations on the file descriptor shall complete as |  
 36580 defined by synchronized I/O file integrity completion. |

36581 SHM If *fildev* refers to a shared memory object, the result of the *read()* function is unspecified. |

36582 TYM If *fildev* refers to a typed memory object, the result of the *read()* function is unspecified. |

36583 XSR A *read()* from a STREAMS file can read data in three different modes: *byte-stream* mode, |  
 36584 *message-nondiscard* mode, and *message-discard* mode. The default shall be byte-stream mode. This |  
 36585 can be changed using the I\_SRDOPT *ioctl()* request, and can be tested with the I\_GRDOPT |  
 36586 *ioctl()*. In byte-stream mode, *read()* shall retrieve data from the STREAM until as many bytes as |  
 36587 were requested are transferred, or until there is no more data to be retrieved. Byte-stream mode |  
 36588 ignores message boundaries.

36589 In STREAMS message-nondiscard mode, *read()* shall retrieve data until as many bytes as were  
 36590 requested are transferred, or until a message boundary is reached. If *read()* does not retrieve all  
 36591 the data in a message, the remaining data shall be left on the STREAM, and can be retrieved by  
 36592 the next *read()* call. Message-discard mode also retrieves data until as many bytes as were  
 36593 requested are transferred, or a message boundary is reached. However, unread data remaining  
 36594 in a message after the *read()* returns shall be discarded, and shall not be available for a  
 36595 subsequent *read()*, *getmsg()*, or *getpmsg()* call. |

36596 How *read()* handles zero-byte STREAMS messages is determined by the current read mode  
 36597 setting. In byte-stream mode, *read()* shall accept data until it has read *nbyte* bytes, or until there  
 36598 is no more data to read, or until a zero-byte message block is encountered. The *read()* function  
 36599 shall then return the number of bytes read, and place the zero-byte message back on the  
 36600 STREAM to be retrieved by the next *read()*, *getmsg()*, or *getpmsg()*. In message-nondiscard mode |  
 36601 or message-discard mode, a zero-byte message shall return 0 and the message shall be removed  
 36602 from the STREAM. When a zero-byte message is read as the first message on a STREAM, the  
 36603 message shall be removed from the STREAM and 0 shall be returned, regardless of the read  
 36604 mode.

36605 A *read()* from a STREAMS file shall return the data in the message at the front of the STREAM  
 36606 head read queue, regardless of the priority band of the message.

36607 By default, STREAMS are in control-normal mode, in which a *read()* from a STREAMS file can  
 36608 only process messages that contain a data part but do not contain a control part. The *read()* shall  
 36609 fail if a message containing a control part is encountered at the STREAM head. This default  
 36610 action can be changed by placing the STREAM in either control-data mode or control-discard  
 36611 mode with the I\_SRDOPT *ioctl()* command. In control-data mode, *read()* shall convert any  
 36612 control part to data and pass it to the application before passing any data part originally present

36613 in the same message. In control-discard mode, *read()* shall discard message control parts but  
 36614 return to the process any data part in the message.

36615 In addition, *read()* shall fail if the STREAM head had processed an asynchronous error before the  
 36616 call. In this case, the value of *errno* shall not reflect the result of *read()*, but reflects the prior error.  
 36617 If a hangup occurs on the STREAM being read, *read()* shall continue to operate normally until  
 36618 the STREAM head read queue is empty. Thereafter, it shall return 0.

36619 XSI The *pread()* function shall be equivalent to *read()*, except that it shall read from a given position  
 36620 in the file without changing the file pointer. The first three arguments to *pread()* are the same as  
 36621 *read()* with the addition of a fourth argument offset for the desired position inside the file. An  
 36622 attempt to perform a *pread()* on a file that is incapable of seeking shall result in an error.

### 36623 RETURN VALUE

36624 XSI Upon successful completion, *read()* and *pread()* shall return a non-negative integer indicating the  
 36625 number of bytes actually read. Otherwise, the functions shall return -1 and set *errno* to indicate  
 36626 the error.

### 36627 ERRORS

36628 XSI The *read()* and *pread()* functions shall fail if:

36629 [EAGAIN] The O\_NONBLOCK flag is set for the file descriptor and the process would be  
 36630 delayed.

36631 [EBADF] The *fdes* argument is not a valid file descriptor open for reading.

36632 XSR [EBADMSG] The file is a STREAM file that is set to control-normal mode and the message  
 36633 waiting to be read includes a control part.

36634 [EINTR] The read operation was terminated due to the receipt of a signal, and no data  
 36635 was transferred.

36636 XSR [EINVAL] The STREAM or multiplexer referenced by *fdes* is linked (directly or  
 36637 indirectly) downstream from a multiplexer.

36638 [EIO] The process is a member of a background process attempting to read from its  
 36639 controlling terminal, the process is ignoring or blocking the SIGTTIN signal,  
 36640 or the process group is orphaned. This error may also be generated for  
 36641 implementation-defined reasons.

36642 XSI [EISDIR] The *fdes* argument refers to a directory and the implementation does not  
 36643 allow the directory to be read using *read()* or *pread()*. The *readdir()* function  
 36644 should be used instead.

36645 [EOVERFLOW] The file is a regular file, *nbyte* is greater than 0, the starting position is before  
 36646 the end-of-file, and the starting position is greater than or equal to the offset  
 36647 maximum established in the open file description associated with *fdes*.

36648 The *read()* function shall fail if:

36649 [EAGAIN] or [EWOULDBLOCK]

36650 The file descriptor is for a socket, is marked O\_NONBLOCK, and no data is  
 36651 waiting to be received.

36652 [ECONNRESET] A read was attempted on a socket and the connection was forcibly closed by  
 36653 its peer.

36654 [ENOTCONN] A read was attempted on a socket that is not connected.

36655 [ETIMEDOUT] A read was attempted on a socket and a transmission timeout occurred.

36656 XSI	The <code>read()</code> and <code>pread()</code> functions may fail if:	
36657	[EIO] A physical I/O error has occurred.	
36658	[ENOBUFS] Insufficient resources were available in the system to perform the operation.	
36659	[ENOMEM] Insufficient memory was available to fulfill the request.	
36660	[ENXIO] A request was made of a nonexistent device, or the request was outside the	
36661	capabilities of the device.	
36662	The <code>pread()</code> function shall fail, and the file pointer shall remain unchanged, if:	
36663 XSI	[EINVAL] The <code>offset</code> argument is invalid. The value is negative.	
36664 XSI	[EOVERFLOW] The file is a regular file and an attempt was made to read at or beyond the	
36665	offset maximum associated with the file.	
36666 XSI	[ENXIO] A request was outside the capabilities of the device.	
36667 XSI	[ESPIPE] <code>fdes</code> is associated with a pipe or FIFO.	

### 36668 EXAMPLES

#### 36669 Reading Data into a Buffer

36670 The following example reads data from the file associated with the file descriptor `fd` into the  
36671 buffer pointed to by `buf`.

```
36672 #include <sys/types.h>
36673 #include <unistd.h>
36674 ...
36675 char buf[20];
36676 size_t nbytes;
36677 ssize_t bytes_read;
36678 int fd;
36679 ...
36680 nbytes = sizeof(buf);
36681 bytes_read = read(fd, buf, nbytes);
36682 ...
```

### 36683 APPLICATION USAGE

36684 None.

### 36685 RATIONALE

36686 This volume of IEEE Std 1003.1-200x does not specify the value of the file offset after an error is  
36687 returned; there are too many cases. For programming errors, such as [EBADF], the concept is  
36688 meaningless since no file is involved. For errors that are detected immediately, such as  
36689 [EAGAIN], clearly the pointer should not change. After an interrupt or hardware error, however,  
36690 an updated value would be very useful and is the behavior of many implementations.

36691 Note that a `read()` of zero bytes does not modify `st_atime`. A `read()` that requests more than zero  
36692 bytes, but returns zero, shall modify `st_atime`.

36693 Implementations are allowed, but not required, to perform error checking for `read()` requests of  
36694 zero bytes.

36695 **Input and Output**

36696 The use of I/O with large byte counts has always presented problems. Ideas such as *lread()* and  
36697 *lwrite()* (using and returning **longs**) were considered at one time. The current solution is to use  
36698 abstract types on the ISO C standard function to *read()* and *write()*. The abstract types can be  
36699 declared so that existing functions work, but can also be declared so that larger types can be  
36700 represented in future implementations. It is presumed that whatever constraints limit the  
36701 maximum range of **size\_t** also limit portable I/O requests to the same range. This volume of  
36702 IEEE Std 1003.1-200x also limits the range further by requiring that the byte count be limited so  
36703 that a signed return value remains meaningful. Since the return type is also a (signed) abstract  
36704 type, the byte count can be defined by the implementation to be larger than an **int** can hold.

36705 The standard developers considered adding atomicity requirements to a pipe or FIFO, but  
36706 recognized that due to the nature of pipes and FIFOs there could be no guarantee of atomicity of  
36707 reads of {PIPE\_BUF} or any other size that would be an aid to applications portability.

36708 This volume of IEEE Std 1003.1-200x requires that no action be taken for *read()* or *write()* when  
36709 *nbyte* is zero. This is not intended to take precedence over detection of errors (such as invalid  
36710 buffer pointers or file descriptors). This is consistent with the rest of this volume of  
36711 IEEE Std 1003.1-200x, but the phrasing here could be misread to require detection of the zero  
36712 case before any other errors. A value of zero is to be considered a correct value, for which the  
36713 semantics are a no-op.

36714 I/O is intended to be atomic to ordinary files and pipes and FIFOs. Atomic means that all the  
36715 bytes from a single operation that started out together end up together, without interleaving  
36716 from other I/O operations. It is a known attribute of terminals that this is not honored, and  
36717 terminals are explicitly (and implicitly permanently) excepted, making the behavior unspecified.  
36718 The behavior for other device types is also left unspecified, but the wording is intended to imply  
36719 that future standards might choose to specify atomicity (or not).

36720 There were recommendations to add format parameters to *read()* and *write()* in order to handle  
36721 networked transfers among heterogeneous file system and base hardware types. Such a facility  
36722 may be required for support by the OSI presentation of layer services. However, it was  
36723 determined that this should correspond with similar C-language facilities, and that is beyond the  
36724 scope of this volume of IEEE Std 1003.1-200x. The concept was suggested to the developers of  
36725 the ISO C standard for their consideration as a possible area for future work.

36726 In 4.3 BSD, a *read()* or *write()* that is interrupted by a signal before transferring any data does not  
36727 by default return an [EINTR] error, but is restarted. In 4.2 BSD, 4.3 BSD, and the Eighth Edition,  
36728 there is an additional function, *select()*, whose purpose is to pause until specified activity (data  
36729 to read, space to write, and so on) is detected on specified file descriptors. It is common in  
36730 applications written for those systems for *select()* to be used before *read()* in situations (such as  
36731 keyboard input) where interruption of I/O due to a signal is desired.

36732 The issue of which files or file types are interruptible is considered an implementation design  
36733 issue. This is often affected primarily by hardware and reliability issues.

36734 There are no references to actions taken following an “unrecoverable error”. It is considered  
36735 beyond the scope of this volume of IEEE Std 1003.1-200x to describe what happens in the case of  
36736 hardware errors.

36737 Previous versions of IEEE Std 1003.1-200x allowed two very different behaviors with regard to  
36738 the handling of interrupts. In order to minimize the resulting confusion, it was decided that  
36739 IEEE Std 1003.1-200x should support only one of these behaviors. Historical practice on AT&T-  
36740 derived systems was to have *read()* and *write()* return **-1** and set *errno* to [EINTR] when  
36741 interrupted after some, but not all, of the data requested had been transferred. However, the U.S.  
36742 Department of Commerce FIPS 151-1 and FIPS 151-2 require the historical BSD behavior, in

36743 which *read()* and *write()* return the number of bytes actually transferred before the interrupt. If  
 36744  $-1$  is returned when any data is transferred, it is difficult to recover from the error on a seekable  
 36745 device and impossible on a non-seekable device. Most new implementations support this  
 36746 behavior. The behavior required by IEEE Std 1003.1-200x is to return the number of bytes  
 36747 transferred.

36748 IEEE Std 1003.1-200x does not specify when an implementation that buffers *read()*s actually  
 36749 moves the data into the user-supplied buffer, so an implementation may chose to do this at the  
 36750 latest possible moment. Therefore, an interrupt arriving earlier may not cause *read()* to return a  
 36751 partial byte count, but rather to return  $-1$  and set *errno* to [EINTR].

36752 Consideration was also given to combining the two previous options, and setting *errno* to  
 36753 [EINTR] while returning a short count. However, not only is there no existing practice that  
 36754 implements this, it is also contradictory to the idea that when *errno* is set, the function  
 36755 responsible shall return  $-1$ .

#### 36756 FUTURE DIRECTIONS

36757 None.

#### 36758 SEE ALSO

36759 *fcntl()*, *ioctl()*, *lseek()*, *open()*, *pipe()*, *readv()*, the Base Definitions volume of |  
 36760 IEEE Std 1003.1-200x, <stropts.h>, <sys/uio.h>, <unistd.h>, the Base Definitions volume of  
 36761 IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface

#### 36762 CHANGE HISTORY

36763 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 36764 Issue 5

36765 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX  
 36766 Threads Extension.

36767 Large File Summit extensions are added.

36768 The *pread()* function is added.

#### 36769 Issue 6

36770 The DESCRIPTION and ERRORS sections are updated so that references to STREAMS are  
 36771 marked as part of the XSI STREAMS Option Group.

36772 The following new requirements on POSIX implementations derive from alignment with the  
 36773 Single UNIX Specification:

36774 • The DESCRIPTION now states that if *read()* is interrupted by a signal after it has successfully  
 36775 read some data, it returns the number of bytes read. In Issue 3, it was optional whether *read()*  
 36776 returned the number of bytes read, or whether it returned  $-1$  with *errno* set to [EINTR]. This  
 36777 is a FIPS requirement.

36778 • In the DESCRIPTION, text is added to indicate that for regular files, no data transfer occurs  
 36779 past the offset maximum established in the open file description associated with *files*. This  
 36780 change is to support large files.

36781 • The [EOVERFLOW] mandatory error condition is added.

36782 • The [ENXIO] optional error condition is added.

36783 Text referring to sockets is added to the DESCRIPTION.

36784 The following changes were made to align with the IEEE P1003.1a draft standard:

36785 • The effect of reading zero bytes is clarified.

36786 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that  
36787 *read()* results are unspecified for typed memory objects.

36788 New RATIONALE is added to explain the atomicity requirements for input and output  
36789 operations.

36790 The following error conditions are added for operations on sockets: [EAGAIN],  
36791 [ECONNRESET], [ENOTCONN], and [ETIMEDOUT].

36792 The [EIO] error is changed to “may fail”.

36793 The following error conditions are added for operations on sockets: [ENOBUFFS] and  
36794 [ENOMEM]. |

36795 The *readv()* function is split out into a separate reference page. |

## 36796 NAME

36797 readdir, readdir\_r — read directory

## 36798 SYNOPSIS

36799 #include &lt;dirent.h&gt;

36800 struct dirent \*readdir(DIR \*dirp);

36801 TSF int readdir\_r(DIR \*restrict dirp, struct dirent \*restrict entry,

36802 struct dirent \*\*restrict result);

36803

## 36804 DESCRIPTION

36805 The type **DIR**, which is defined in the <**dirent.h**> header, represents a *directory stream*, which is  
 36806 an ordered sequence of all the directory entries in a particular directory. Directory entries  
 36807 represent files; files may be removed from a directory or added to a directory asynchronously to  
 36808 the operation of *readdir()*.

36809 The *readdir()* function shall return a pointer to a structure representing the directory entry at the  
 36810 current position in the directory stream specified by the argument *dirp*, and position the  
 36811 directory stream at the next entry. It shall return a null pointer upon reaching the end of the  
 36812 directory stream. The structure **dirent** defined in the <**dirent.h**> header describes a directory  
 36813 entry.

36814 The *readdir()* function shall not return directory entries containing empty names. If entries for  
 36815 dot or dot-dot exist, one entry shall be returned for dot and one entry shall be returned for dot-  
 36816 dot; otherwise, they shall not be returned.

36817 The pointer returned by *readdir()* points to data which may be overwritten by another call to  
 36818 *readdir()* on the same directory stream. This data is not overwritten by another call to *readdir()*  
 36819 on a different directory stream.

36820 If a file is removed from or added to the directory after the most recent call to *opendir()* or  
 36821 *rewinddir()*, whether a subsequent call to *readdir()* returns an entry for that file is unspecified.

36822 The *readdir()* function may buffer several directory entries per actual read operation; *readdir()*  
 36823 shall mark for update the *st\_atime* field of the directory each time the directory is actually read.

36824 After a call to *fork()*, either the parent or child (but not both) may continue processing the  
 36825 XSI directory stream using *readdir()*, *rewinddir()*, or *seekdir()*. If both the parent and child processes  
 36826 use these functions, the result is undefined.

36827 If the entry names a symbolic link, the value of the *d\_ino* member is unspecified.

36828 The *readdir()* function need not be reentrant. A function that is not required to be reentrant is not  
 36829 required to be thread-safe.

36830 TSF The *readdir\_r()* function shall initialize the **dirent** structure referenced by *entry* to represent the  
 36831 directory entry at the current position in the directory stream referred to by *dirp*, store a pointer  
 36832 to this structure at the location referenced by *result*, and position the directory stream at the next  
 36833 entry.

36834 The storage pointed to by *entry* shall be large enough for a **dirent** with an array of **char** *d\_name*  
 36835 members containing at least {NAME\_MAX} plus one elements.

36836 Upon successful return, the pointer returned at *\*result* shall have the same value as the argument  
 36837 *entry*. Upon reaching the end of the directory stream, this pointer shall have the value NULL.

36838 The *readdir\_r()* function shall not return directory entries containing empty names.

36839 If a file is removed from or added to the directory after the most recent call to *opendir()* or  
 36840 *rewinddir()*, whether a subsequent call to *readdir\_r()* returns an entry for that file is unspecified.

36841 The *readdir\_r()* function may buffer several directory entries per actual read operation; the  
 36842 *readdir\_r()* function shall mark for update the *st\_atime* field of the directory each time the  
 36843 directory is actually read.

36844 Applications wishing to check for error situations should set *errno* to 0 before calling *readdir()*. If  
 36845 *errno* is set to non-zero on return, an error occurred.

#### 36846 RETURN VALUE

36847 Upon successful completion, *readdir()* shall return a pointer to an object of type **struct dirent**.  
 36848 When an error is encountered, a null pointer shall be returned and *errno* shall be set to indicate  
 36849 the error. When the end of the directory is encountered, a null pointer shall be returned and *errno*  
 36850 is not changed.

36851 TSF If successful, the *readdir\_r()* function shall return zero; otherwise, an error number shall be  
 36852 returned to indicate the error.

#### 36853 ERRORS

36854 The *readdir()* function shall fail if:

36855 [EOVERFLOW] One of the values in the structure to be returned cannot be represented  
 36856 correctly.

36857 The *readdir()* function may fail if:

36858 [EBADF] The *dirp* argument does not refer to an open directory stream.

36859 [ENOENT] The current position of the directory stream is invalid.

36860 The *readdir\_r()* function may fail if:

36861 [EBADF] The *dirp* argument does not refer to an open directory stream.

#### 36862 EXAMPLES

36863 The following sample code searches the current directory for the entry *name*:

```
36864 dirp = opendir(".");
36865 while (dirp) {
36866     errno = 0;
36867     if ((dp = readdir(dirp)) != NULL) {
36868         if (strcmp(dp->d_name, name) == 0) {
36869             closedir(dirp);
36870             return FOUND;
36871         }
36872     } else {
36873         if (errno == 0) {
36874             closedir(dirp);
36875             return NOT_FOUND;
36876         }
36877         closedir(dirp);
36878         return READ_ERROR;
36879     }
36880 }
36881 return OPEN_ERROR;
```

36882 **APPLICATION USAGE**

36883 The *readdir()* function should be used in conjunction with *opendir()*, *closedir()*, and *rewinddir()* to  
 36884 examine the contents of the directory.

36885 The *readdir\_r()* function is thread-safe and shall return values in a user-supplied buffer instead  
 36886 of possibly using a static data area that may be overwritten by each call.

36887 **RATIONALE**

36888 The returned value of *readdir()* merely *represents* a directory entry. No equivalence should be  
 36889 inferred.

36890 Historical implementations of *readdir()* obtain multiple directory entries on a single read  
 36891 operation, which permits subsequent *readdir()* operations to operate from the buffered  
 36892 information. Any wording that required each successful *readdir()* operation to mark the  
 36893 directory *st\_atime* field for update would militate against the historical performance-oriented  
 36894 implementations.

36895 Since *readdir()* returns NULL when it detects an error and when the end of the directory is  
 36896 encountered, an application that needs to tell the difference must set *errno* to zero before the call  
 36897 and check it if NULL is returned. Since the function must not change *errno* in the second case  
 36898 and must set it to a non-zero value in the first case, a zero *errno* after a call returning NULL  
 36899 indicates end of directory; otherwise, an error.

36900 Routines to deal with this problem more directly were proposed:

```
36901 int derror (dirp)
```

```
36902 DIR *dirp;
```

```
36903 void clearerr (dirp)
```

```
36904 DIR *dirp;
```

36905 The first would indicate whether an error had occurred, and the second would clear the error  
 36906 indication. The simpler method involving *errno* was adopted instead by requiring that *readdir()*  
 36907 not change *errno* when end-of-directory is encountered.

36908 An error or signal indicating that a directory has changed while open was considered but  
 36909 rejected.

36910 The thread-safe version of the directory reading function returns values in a user-supplied buffer  
 36911 instead of possibly using a static data area that may be overwritten by each call. Either the  
 36912 {NAME\_MAX} compile-time constant or the corresponding *pathconf()* option can be used to  
 36913 determine the maximum sizes of returned pathnames.

36914 **FUTURE DIRECTIONS**

36915 None.

36916 **SEE ALSO**

36917 *closedir()*, *lstat()*, *opendir()*, *rewinddir()*, *symlink()*, the Base Definitions volume of  
 36918 IEEE Std 1003.1-200x, <*dirent.h*>, <*sys/types.h*>

36919 **CHANGE HISTORY**

36920 First released in Issue 2.

36921 **Issue 5**

36922 Large File Summit extensions are added.

36923 The *readdir\_r()* function is included for alignment with the POSIX Threads Extension.

36924 A note indicating that the *readdir()* function need not be reentrant is added to the  
 36925 DESCRIPTION.

36926 **Issue 6**

- 36927 The `readdir_r()` function is marked as part of the Thread-Safe Functions option.
- 36928 The Open Group Corrigendum U026/7 is applied, correcting the prototype for `readdir_r()`.
- 36929 The Open Group Corrigendum U026/8 is applied, clarifying the wording of the successful  
36930 return for the `readdir_r()` function.
- 36931 The following new requirements on POSIX implementations derive from alignment with the  
36932 Single UNIX Specification:
- 36933 • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was  
36934 required for conforming implementations of previous POSIX specifications, it was not  
36935 required for UNIX applications.
  - 36936 • A statement is added to the DESCRIPTION indicating the disposition of certain fields in  
36937 **struct dirent** when an entry refers to a symbolic link.
  - 36938 • The [EOVERFLOW] mandatory error condition is added. This change is to support large  
36939 files.
  - 36940 • The [ENOENT] optional error condition is added.
- 36941 The APPLICATION USAGE section is updated to include a note on the thread-safe function and  
36942 its avoidance of possibly using a static data area.
- 36943 The **restrict** keyword is added to the `readdir_r()` prototype for alignment with the  
36944 ISO/IEC 9899:1999 standard.

36945 **NAME**

36946 readlink — read the contents of a symbolic link

36947 **SYNOPSIS**

36948 #include &lt;unistd.h&gt;

36949 ssize\_t readlink(const char \*restrict path, char \*restrict buf,  
36950 size\_t bufsize);36951 **DESCRIPTION**36952 The *readlink()* function shall place the contents of the symbolic link referred to by *path* in the  
36953 buffer *buf* which has size *bufsize*. If the number of bytes in the symbolic link is less than *bufsize*,  
36954 the contents of the remainder of *buf* are unspecified. If the *buf* argument is not large enough to  
36955 contain the link content, the first *bufsize* bytes shall be placed in *buf*.36956 If the value of *bufsize* is greater than {SSIZE\_MAX}, the result is implementation-defined.36957 **RETURN VALUE**36958 Upon successful completion, *readlink()* shall return the count of bytes placed in the buffer.  
36959 Otherwise, it shall return a value of -1, leave the buffer unchanged, and set *errno* to indicate the  
36960 error.36961 **ERRORS**36962 The *readlink()* function shall fail if:36963 [EACCES] Search permission is denied for a component of the path prefix of *path*.36964 [EINVAL] The *path* argument names a file that is not a symbolic link.

36965 [EIO] An I/O error occurred while reading from the file system.

36966 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*  
36967 argument.

36968 [ENAMETOOLONG]

36969 The length of the *path* argument exceeds {PATH\_MAX} or a pathname |  
36970 component is longer than {NAME\_MAX}. |36971 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

36972 [ENOTDIR] A component of the path prefix is not a directory.

36973 The *readlink()* function may fail if:

36974 [EACCES] Read permission is denied for the directory.

36975 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
36976 resolution of the *path* argument.

36977 [ENAMETOOLONG]

36978 As a result of encountering a symbolic link in resolution of the *path* argument, |  
36979 the length of the substituted pathname string exceeded {PATH\_MAX}. |

36980 **EXAMPLES**36981 **Reading the Name of a Symbolic Link**

36982 The following example shows how to read the name of a symbolic link named `/modules/pass1`.

```
36983 #include <unistd.h>
36984 char buf[1024];
36985 int len;
36986 ...
36987 if ((len = readlink("/modules/pass1", buf, sizeof(buf)-1)) != -1);
36988 buf[len] = '\0';
```

36989 **APPLICATION USAGE**

36990 Conforming applications should not assume that the returned contents of the symbolic link are null-terminated.

36992 **RATIONALE**

36993 Since IEEE Std 1003.1-200x does not require any association of file times with symbolic links, there is no requirement that file times be updated by `readlink()`. The type associated with `bufsiz` is a `size_t` in order to be consistent with both the ISO C standard and the definition of `read()`. The behavior specified for `readlink()` when `bufsiz` is zero represents historical practice. For this case, the standard developers considered a change whereby `readlink()` would return the number of non-null bytes contained in the symbolic link with the buffer `buf` remaining unchanged; however, since the `stat` structure member `st_size` value can be used to determine the size of buffer necessary to contain the contents of the symbolic link as returned by `readlink()`, this proposal was rejected, and the historical practice retained.

37002 **FUTURE DIRECTIONS**

37003 None.

37004 **SEE ALSO**

37005 `lstat()`, `stat()`, `symlink()`, the Base Definitions volume of IEEE Std 1003.1-200x, `<unistd.h>`

37006 **CHANGE HISTORY**

37007 First released in Issue 4, Version 2.

37008 **Issue 5**

37009 Moved from X/OPEN UNIX extension to BASE.

37010 **Issue 6**

37011 The return type is changed to `ssize_t`, to align with the IEEE P1003.1a draft standard.

37012 The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- 37014 • This function is made mandatory.
- 37015 • In this function it is possible for the return value to exceed the range of the type `ssize_t` (since `size_t` has a larger range of positive values than `ssize_t`). A sentence restricting the size of the `size_t` object is added to the description to resolve this conflict.

37018 The following changes are made for alignment with the ISO POSIX-1:1996 standard:

- 37019 • The FUTURE DIRECTIONS section is changed to None.

37020 The following changes were made to align with the IEEE P1003.1a draft standard:

- 37021 • The [ELOOP] optional error condition is added.

37022  
37023

The **restrict** keyword is added to the *readlink()* prototype for alignment with the ISO/IEC 9899:1999 standard.

37024 **NAME**

37025 readv — read a vector

37026 **SYNOPSIS**37027 XSI `#include <sys/uio.h>`37028 `ssize_t readv(int fildes, const struct iovec *iovcnt, int iovcnt);`

37029

37030 **DESCRIPTION**

37031 The `readv()` function shall be equivalent to `read()`, except as described below. The `readv()`  
 37032 function shall place the input data into the `iovcnt` buffers specified by the members of the `iovcnt`  
 37033 array: `iovcnt[0]`, `iovcnt[1]`, ..., `iovcnt[iovcnt-1]`. The `iovcnt` argument is valid if greater than 0 and less than  
 37034 or equal to `{IOV_MAX}`.

37035 Each `iovcnt` entry specifies the base address and length of an area in memory where data should  
 37036 be placed. The `readv()` function shall always fill an area completely before proceeding to the  
 37037 next.

37038 Upon successful completion, `readv()` shall mark for update the `st_atime` field of the file.

37039 **RETURN VALUE**37040 Refer to `read()`.37041 **ERRORS**37042 Refer to `read()`.37043 In addition, the `readv()` function shall fail if:37044 `[EINVAL]` The sum of the `iovcnt[i].iovcnt` values in the `iovcnt` array overflowed an `ssize_t`.37045 The `readv()` function may fail if:37046 `[EINVAL]` The `iovcnt` argument was less than or equal to 0, or greater than `{IOV_MAX}`.37047 **EXAMPLES**37048 **Reading Data into an Array**

37049 The following example reads data from the file associated with the file descriptor `fd` into the  
 37050 buffers specified by members of the `iovcnt` array.

```

37051 #include <sys/types.h>
37052 #include <sys/uio.h>
37053 #include <unistd.h>
37054 ...
37055 ssize_t bytes_read;
37056 int fd;
37057 char buf0[20];
37058 char buf1[30];
37059 char buf2[40];
37060 int iovcnt;
37061 struct iovec iov[3];

37062 iov[0].iovcnt_base = buf0;
37063 iov[0].iovcnt_len = sizeof(buf0);
37064 iov[1].iovcnt_base = buf1;
37065 iov[1].iovcnt_len = sizeof(buf1);
37066 iov[2].iovcnt_base = buf2;
37067 iov[2].iovcnt_len = sizeof(buf2);

```

```
37068     ...
37069     iovcnt = sizeof(iov) / sizeof(struct iovec);
37070     bytes_read = readv(fd, iov, iovcnt);
37071     ...

37072 APPLICATION USAGE
37073     None.

37074 RATIONALE
37075     Refer to read().

37076 FUTURE DIRECTIONS
37077     None.

37078 SEE ALSO
37079     read(), writenv(), the Base Definitions volume of IEEE Std 1003.1-200x, <sys/uio.h>

37080 CHANGE HISTORY
37081     First released in Issue 4, Version 2.

37082 Issue 6
37083     Split out from the read() reference page.
```

37084 **NAME**

37085           realloc — memory reallocator

37086 **SYNOPSIS**

37087           #include &lt;stdlib.h&gt;

37088           void \*realloc(void \*ptr, size\_t size);

37089 **DESCRIPTION**

37090 cx       The functionality described on this reference page is aligned with the ISO C standard. Any  
37091       conflict between the requirements described here and the ISO C standard is unintentional. This  
37092       volume of IEEE Std 1003.1-200x defers to the ISO C standard.

37093       The *realloc()* function shall change the size of the memory object pointed to by *ptr* to the size  
37094       specified by *size*. The contents of the object shall remain unchanged up to the lesser of the new  
37095       and old sizes. If the new size of the memory object would require movement of the object, the  
37096       space for the previous instantiation of the object is freed. If the new size is larger, the contents of  
37097       the newly allocated portion of the object are unspecified. If *size* is 0 and *ptr* is not a null pointer,  
37098       the object pointed to is freed. If the space cannot be allocated, the object shall remain unchanged.

37099       If *ptr* is a null pointer, *realloc()* shall be equivalent to *malloc()* for the specified size.

37100       If *ptr* does not match a pointer returned earlier by *calloc()*, *malloc()*, or *realloc()* or if the space has  
37101       previously been deallocated by a call to *free()* or *realloc()*, the behavior is undefined.

37102       The order and contiguity of storage allocated by successive calls to *realloc()* is unspecified. The  
37103       pointer returned if the allocation succeeds shall be suitably aligned so that it may be assigned to  
37104       a pointer to any type of object and then used to access such an object in the space allocated (until  
37105       the space is explicitly freed or reallocated). Each such allocation shall yield a pointer to an object  
37106       disjoint from any other object. The pointer returned shall point to the start (lowest byte address)  
37107       of the allocated space. If the space cannot be allocated, a null pointer shall be returned.

37108 **RETURN VALUE**

37109       Upon successful completion with a *size* not equal to 0, *realloc()* shall return a pointer to the  
37110       (possibly moved) allocated space. If *size* is 0, either a null pointer or a unique pointer that can be  
37111       successfully passed to *free()* shall be returned. If there is not enough available memory, *realloc()*  
37112 cx       shall return a null pointer and set *errno* to [ENOMEM].

37113 **ERRORS**37114       The *realloc()* function shall fail if:

37115 cx       [ENOMEM]       Insufficient memory is available.

37116 **EXAMPLES**

37117       None.

37118 **APPLICATION USAGE**

37119       None.

37120 **RATIONALE**

37121       None.

37122 **FUTURE DIRECTIONS**

37123       None.

37124 **SEE ALSO**37125       *calloc()*, *free()*, *malloc()*, the Base Definitions volume of IEEE Std 1003.1-200x, <stdlib.h>

37126 **CHANGE HISTORY**

37127 First released in Issue 1. Derived from Issue 1 of the SVID.

37128 **Issue 6**

37129 Extensions beyond the ISO C standard are now marked.

37130 The following new requirements on POSIX implementations derive from alignment with the  
37131 Single UNIX Specification:

- 37132 • In the RETURN VALUE section, if there is not enough available memory, the setting of *errno*  
37133 to [ENOMEM] is added.
- 37134 • The [ENOMEM] error condition is added.

## 37135 NAME

37136        realpath — resolve a pathname |

## 37137 SYNOPSIS

37138 xSI        #include &lt;stdlib.h&gt;

37139        char \*realpath(const char \*restrict *file\_name*,  
37140                      char \*restrict *resolved\_name*);

37141

## 37142 DESCRIPTION

37143        The *realpath()* function shall derive, from the pathname pointed to by *file\_name*, an absolute |  
 37144        pathname that names the same file, whose resolution does not involve '.', '..', or symbolic |  
 37145        links. The generated pathname shall be stored as a null-terminated string, up to a maximum of |  
 37146        {PATH\_MAX} bytes, in the buffer pointed to by *resolved\_name*.

## 37147 RETURN VALUE

37148        Upon successful completion, *realpath()* shall return a pointer to the resolved name. Otherwise,  
 37149        *realpath()* shall return a null pointer and set *errno* to indicate the error, and the contents of the  
 37150        buffer pointed to by *resolved\_name* are undefined.

## 37151 ERRORS

37152        The *realpath()* function shall fail if:37153        [EACCES]        Read or search permission was denied for a component of *file\_name*.37154        [EINVAL]        Either the *file\_name* or *resolved\_name* argument is a null pointer.

37155        [EIO]            An error occurred while reading from the file system.

37156        [ELOOP]        A loop exists in symbolic links encountered during resolution of the *path*  
37157        argument.

37158        [ENAMETOOLONG]

37159                      The length of the *file\_name* argument exceeds {PATH\_MAX} or a pathname |  
37160                      component is longer than {NAME\_MAX}. |37161        [ENOENT]        A component of *file\_name* does not name an existing file or *file\_name* points to  
37162        an empty string.

37163        [ENOTDIR]       A component of the path prefix is not a directory.

37164        The *realpath()* function may fail if:37165        [ELOOP]        More than {SYMLOOP\_MAX} symbolic links were encountered during  
37166        resolution of the *path* argument.

37167        [ENAMETOOLONG]

37168                      Pathname resolution of a symbolic link produced an intermediate result |  
37169                      whose length exceeds {PATH\_MAX}. |

37170        [ENOMEM]        Insufficient storage space is available.

37171 **EXAMPLES**37172 **Generating an Absolute Pathname** |

37173 The following example generates an absolute pathname for the file identified by the *symlinkpath* |  
37174 argument. The generated pathname is stored in the *actualpath* array. |

```
37175 #include <stdlib.h>
37176 ...
37177 char *symlinkpath = "/tmp/symlink/file";
37178 char actualpath [PATH_MAX+1];
37179 char *ptr;

37180 ptr = realpath(symlinkpath, actualpath);
```

37181 **APPLICATION USAGE**

37182 None.

37183 **RATIONALE**

37184 None.

37185 **FUTURE DIRECTIONS**

37186 None.

37187 **SEE ALSO**

37188 *getcwd()*, *sysconf()*, the Base Definitions volume of IEEE Std 1003.1-200x, <**stdlib.h**>

37189 **CHANGE HISTORY**

37190 First released in Issue 4, Version 2.

37191 **Issue 5**

37192 Moved from X/OPEN UNIX extension to BASE.

37193 **Issue 6**

37194 The **restrict** keyword is added to the *realpath()* prototype for alignment with the |  
37195 ISO/IEC 9899:1999 standard.

37196 The wording of the mandatory [ELOOP] error condition is updated, and a second optional  
37197 [ELOOP] error condition is added.

37198 **NAME**

37199       recv — receive a message from a connected socket

37200 **SYNOPSIS**

37201       #include &lt;sys/socket.h&gt;

37202       ssize\_t recv(int *socket*, void \**buffer*, size\_t *length*, int *flags*);37203 **DESCRIPTION**

37204       The *recv()* function shall receive a message from a connection-mode or connectionless-mode  
 37205       socket. It is normally used with connected sockets because it does not permit the application to  
 37206       retrieve the source address of received data.

37207       The *recv()* function takes the following arguments:37208       *socket*       Specifies the socket file descriptor.37209       *buffer*       Points to a buffer where the message should be stored.37210       *length*       Specifies the length in bytes of the buffer pointed to by the *buffer* argument.

37211       *flags*       Specifies the type of message reception. Values of this argument are formed by  
 37212       logically OR'ing zero or more of the following values:

37213               MSG\_PEEK       Peeks at an incoming message. The data is treated as unread and  
 37214               the next *recv()* or similar function shall still return this data.

37215               MSG\_OOB       Requests out-of-band data. The significance and semantics of  
 37216               out-of-band data are protocol-specific.

37217               MSG\_WAITALL   On SOCK\_STREAM sockets this requests that the function block |  
 37218               until the full amount of data can be returned. The function may |  
 37219               return the smaller amount of data if the socket is a message- |  
 37220               based socket, if a signal is caught, if the connection is |  
 37221               terminated, if MSG\_PEEK was specified, or if an error is pending |  
 37222               for the socket. |

37223       The *recv()* function shall return the length of the message written to the buffer pointed to by the  
 37224       *buffer* argument. For message-based sockets, such as SOCK\_DGRAM and SOCK\_SEQPACKET,  
 37225       the entire message shall be read in a single operation. If a message is too long to fit in the  
 37226       supplied buffer, and MSG\_PEEK is not set in the *flags* argument, the excess bytes shall be  
 37227       discarded. For stream-based sockets, such as SOCK\_STREAM, message boundaries shall be  
 37228       ignored. In this case, data shall be returned to the user as soon as it becomes available, and no  
 37229       data shall be discarded. |

37230       If the MSG\_WAITALL flag is not set, data shall be returned only up to the end of the first  
 37231       message.

37232       If no messages are available at the socket and O\_NONBLOCK is not set on the socket's file  
 37233       descriptor, *recv()* shall block until a message arrives. If no messages are available at the socket  
 37234       and O\_NONBLOCK is set on the socket's file descriptor, *recv()* shall fail and set *errno* to  
 37235       [EAGAIN] or [EWOULDBLOCK].

37236 **RETURN VALUE**

37237       Upon successful completion, *recv()* shall return the length of the message in bytes. If no  
 37238       messages are available to be received and the peer has performed an orderly shutdown, *recv()*  
 37239       shall return 0. Otherwise, -1 shall be returned and *errno* set to indicate the error.

37240 **ERRORS**

- 37241 The *recv()* function shall fail if:
- 37242 [EAGAIN] or [EWOULDBLOCK]
- 37243 The socket's file descriptor is marked O\_NONBLOCK and no data is waiting  
37244 to be received; or MSG\_OOB is set and no out-of-band data is available and  
37245 either the socket's file descriptor is marked O\_NONBLOCK or the socket does  
37246 not support blocking to await out-of-band data.
- 37247 [EBADF] The *socket* argument is not a valid file descriptor.
- 37248 [ECONNRESET] A connection was forcibly closed by a peer.
- 37249 [EINTR] The *recv()* function was interrupted by a signal that was caught, before any  
37250 data was available.
- 37251 [EINVAL] The MSG\_OOB flag is set and no out-of-band data is available.
- 37252 [ENOTCONN] A receive is attempted on a connection-mode socket that is not connected.
- 37253 [ENOTSOCK] The *socket* argument does not refer to a socket.
- 37254 [EOPNOTSUPP] The specified flags are not supported for this socket type or protocol.
- 37255 [ETIMEDOUT] The connection timed out during connection establishment, or due to a  
37256 transmission timeout on active connection.
- 37257 The *recv()* function may fail if:
- 37258 [EIO] An I/O error occurred while reading from or writing to the file system.
- 37259 [ENOBUFS] Insufficient resources were available in the system to perform the operation.
- 37260 [ENOMEM] Insufficient memory was available to fulfill the request.

37261 **EXAMPLES**

37262 None.

37263 **APPLICATION USAGE**

37264 The *recv()* function is equivalent to *recvfrom()* with a zero *address\_len* argument, and to *read()* if  
37265 no flags are used.

37266 The *select()* and *poll()* functions can be used to determine when data is available to be received.

37267 **RATIONALE**

37268 None.

37269 **FUTURE DIRECTIONS**

37270 None.

37271 **SEE ALSO**

37272 *poll()*, *read()*, *recvmsg()*, *recvfrom()*, *select()*, *send()*, *sendmsg()*, *sendto()*, *shutdown()*, *socket()*,  
37273 *write()*, the Base Definitions volume of IEEE Std 1003.1-200x, <sys/socket.h>

37274 **CHANGE HISTORY**

37275 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

## 37276 NAME

37277 recvfrom — receive a message from a socket

## 37278 SYNOPSIS

37279 #include &lt;sys/socket.h&gt;

37280 ssize\_t recvfrom(int socket, void \*restrict buffer, size\_t length,

37281 int flags, struct sockaddr \*restrict address,

37282 socklen\_t \*restrict address\_len);

## 37283 DESCRIPTION

37284 The *recvfrom()* function shall receive a message from a connection-mode or connectionless-mode  
 37285 socket. It is normally used with connectionless-mode sockets because it permits the application  
 37286 to retrieve the source address of received data.

37287 The *recvfrom()* function takes the following arguments:

37288	<i>socket</i>	Specifies the socket file descriptor.
37289	<i>buffer</i>	Points to the buffer where the message should be stored.
37290	<i>length</i>	Specifies the length in bytes of the buffer pointed to by the <i>buffer</i> argument.
37291	<i>flags</i>	Specifies the type of message reception. Values of this argument are formed
37292		by logically OR'ing zero or more of the following values:
37293	MSG_PEEK	Peeks at an incoming message. The data is treated as unread
37294		and the next <i>recvfrom()</i> or similar function shall still return
37295		this data.
37296	MSG_OOB	Requests out-of-band data. The significance and semantics
37297		of out-of-band data are protocol-specific.
37298	MSG_WAITALL	On SOCK_STREAM sockets this requests that the function
37299		block until the full amount of data can be returned. The
37300		function may return the smaller amount of data if the socket
37301		is a message-based socket, if a signal is caught, if the
37302		connection is terminated, if MSG_PEEK was specified, or if
37303		an error is pending for the socket.
37304	<i>address</i>	A null pointer, or points to a <b>sockaddr</b> structure in which the sending address
37305		is to be stored. The length and format of the address depend on the address
37306		family of the socket.
37307	<i>address_len</i>	Specifies the length of the <b>sockaddr</b> structure pointed to by the <i>address</i>
37308		argument.

37309 The *recvfrom()* function shall return the length of the message written to the buffer pointed to by  
 37310 RS the *buffer* argument. For message-based sockets, such as SOCK\_RAW, SOCK\_DGRAM, and  
 37311 SOCK\_SEQPACKET, the entire message shall be read in a single operation. If a message is too  
 37312 long to fit in the supplied buffer, and MSG\_PEEK is not set in the *flags* argument, the excess  
 37313 bytes shall be discarded. For stream-based sockets, such as SOCK\_STREAM, message  
 37314 boundaries shall be ignored. In this case, data shall be returned to the user as soon as it becomes  
 37315 available, and no data shall be discarded.

37316 If the MSG\_WAITALL flag is not set, data shall be returned only up to the end of the first  
 37317 message.

37318 Not all protocols provide the source address for messages. If the *address* argument is not a null  
 37319 pointer and the protocol provides the source address of messages, the source address of the

37320 received message shall be stored in the **sockaddr** structure pointed to by the *address* argument, |  
 37321 and the length of this address shall be stored in the object pointed to by the *address\_len* |  
 37322 argument.

37323 If the actual length of the address is greater than the length of the supplied **sockaddr** structure, |  
 37324 the stored address shall be truncated.

37325 If the *address* argument is not a null pointer and the protocol does not provide the source address |  
 37326 of messages, the value stored in the object pointed to by *address* is unspecified.

37327 If no messages are available at the socket and O\_NONBLOCK is not set on the socket's file |  
 37328 descriptor, *recvfrom()* shall block until a message arrives. If no messages are available at the |  
 37329 socket and O\_NONBLOCK is set on the socket's file descriptor, *recvfrom()* shall fail and set *errno* |  
 37330 to [EAGAIN] or [EWOULDBLOCK].

### 37331 RETURN VALUE

37332 Upon successful completion, *recvfrom()* shall return the length of the message in bytes. If no |  
 37333 messages are available to be received and the peer has performed an orderly shutdown, |  
 37334 *recvfrom()* shall return 0. Otherwise, the function shall return -1 and set *errno* to indicate the |  
 37335 error.

### 37336 ERRORS

37337 The *recvfrom()* function shall fail if:

37338 [EAGAIN] or [EWOULDBLOCK]

37339 The socket's file descriptor is marked O\_NONBLOCK and no data is waiting |  
 37340 to be received; or MSG\_OOB is set and no out-of-band data is available and |  
 37341 either the socket's file descriptor is marked O\_NONBLOCK or the socket does |  
 37342 not support blocking to await out-of-band data.

37343 [EBADF] The *socket* argument is not a valid file descriptor.

37344 [ECONNRESET] A connection was forcibly closed by a peer.

37345 [EINTR] A signal interrupted *recvfrom()* before any data was available.

37346 [EINVAL] The MSG\_OOB flag is set and no out-of-band data is available.

37347 [ENOTCONN] A receive is attempted on a connection-mode socket that is not connected.

37348 [ENOTSOCK] The *socket* argument does not refer to a socket.

37349 [EOPNOTSUPP] The specified flags are not supported for this socket type.

37350 [ETIMEDOUT] The connection timed out during connection establishment, or due to a |  
 37351 transmission timeout on active connection.

37352 The *recvfrom()* function may fail if:

37353 [EIO] An I/O error occurred while reading from or writing to the file system.

37354 [ENOBUFS] Insufficient resources were available in the system to perform the operation.

37355 [ENOMEM] Insufficient memory was available to fulfill the request.

37356 **EXAMPLES**

37357 None.

37358 **APPLICATION USAGE**37359 The *select()* and *poll()* functions can be used to determine when data is available to be received.37360 **RATIONALE**

37361 None.

37362 **FUTURE DIRECTIONS**

37363 None.

37364 **SEE ALSO**37365 *poll()*, *read()*, *recv()*, *recvmsg()*, *select()* (on page 1742)1 *send()*, *sendmsg()*, *sendto()*, *shutdown()*,37366 *socket()*, *write()*, the Base Definitions volume of IEEE Std 1003.1-200x, <sys/socket.h>37367 **CHANGE HISTORY**

37368 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

37369 **NAME**

37370 recvmsg — receive a message from a socket

37371 **SYNOPSIS**

37372 #include &lt;sys/socket.h&gt;

37373 ssize\_t recvmsg(int socket, struct msghdr \*message, int flags);

37374 **DESCRIPTION**

37375 The *recvmsg()* function shall receive a message from a connection-mode or connectionless-mode  
 37376 socket. It is normally used with connectionless-mode sockets because it permits the application  
 37377 to retrieve the source address of received data.

37378 The *recvmsg()* function takes the following arguments:

37379	<i>socket</i>	Specifies the socket file descriptor.
37380	<i>message</i>	Points to a <b>msghdr</b> structure, containing both the buffer to store the source address and the buffers for the incoming message. The length and format of the address depend on the address family of the socket. The <i>msg_flags</i> member is ignored on input, but may contain meaningful values on output.
37381		
37382		
37383		
37384	<i>flags</i>	Specifies the type of message reception. Values of this argument are formed by logically OR'ing zero or more of the following values:
37385		
37386	MSG_OOB	Requests out-of-band data. The significance and semantics of out-of-band data are protocol-specific.
37387		
37388	MSG_PEEK	Peeks at the incoming message.
37389	MSG_WAITALL	On SOCK_STREAM sockets this requests that the function
37390		block until the full amount of data can be returned. The
37391		function may return the smaller amount of data if the socket
37392		is a message-based socket, if a signal is caught, if the
37393		connection is terminated, if MSG_PEEK was specified, or if
37394		an error is pending for the socket.

37395 The *recvmsg()* function shall receive messages from unconnected or connected sockets and shall  
 37396 return the length of the message.

37397 The *recvmsg()* function shall return the total length of the message. For message-based sockets,  
 37398 such as SOCK\_DGRAM and SOCK\_SEQPACKET, the entire message shall be read in a single  
 37399 operation. If a message is too long to fit in the supplied buffers, and MSG\_PEEK is not set in the  
 37400 *flags* argument, the excess bytes shall be discarded, and MSG\_TRUNC shall be set in the  
 37401 *msg\_flags* member of the **msghdr** structure. For stream-based sockets, such as SOCK\_STREAM,  
 37402 message boundaries shall be ignored. In this case, data shall be returned to the user as soon as it  
 37403 becomes available, and no data shall be discarded.

37404 If the MSG\_WAITALL flag is not set, data shall be returned only up to the end of the first  
 37405 message.

37406 If no messages are available at the socket and O\_NONBLOCK is not set on the socket's file  
 37407 descriptor, *recvmsg()* shall block until a message arrives. If no messages are available at the  
 37408 socket and O\_NONBLOCK is set on the socket's file descriptor, *recvmsg()* function shall fail and  
 37409 set *errno* to [EAGAIN] or [EWOULDBLOCK].

37410 In the **msghdr** structure, the *msg\_name* and *msg\_namelen* members specify the source address if  
 37411 the socket is unconnected. If the socket is connected, the *msg\_name* and *msg\_namelen* members  
 37412 shall be ignored. The *msg\_name* member may be a null pointer if no names are desired or  
 37413 required. The *msg\_iov* and *msg\_iovlen* fields are used to specify where the received data shall be

37414 stored. *msg\_iov* points to an array of **iovec** structures; *msg\_iovlen* shall be set to the dimension of  
 37415 this array. In each **iovec** structure, the *iov\_base* field specifies a storage area and the *iov\_len* field  
 37416 gives its size in bytes. Each storage area indicated by *msg\_iov* is filled with received data in turn  
 37417 until all of the received data is stored or all of the areas have been filled.

37418 Upon successful completion, the *msg\_flags* member of the message header shall be the bitwise- |  
 37419 inclusive OR of all of the following flags that indicate conditions detected for the received |  
 37420 message: |

37421 MSG\_EOR End of record was received (if supported by the protocol).

37422 MSG\_OOB Out-of-band data was received.

37423 MSG\_TRUNC Normal data was truncated.

37424 MSG\_CTRUNC Control data was truncated.

#### 37425 RETURN VALUE

37426 Upon successful completion, *recvmsg()* shall return the length of the message in bytes. If no  
 37427 messages are available to be received and the peer has performed an orderly shutdown,  
 37428 *recvmsg()* shall return 0. Otherwise,  $-1$  shall be returned and *errno* set to indicate the error.

#### 37429 ERRORS

37430 The *recvmsg()* function shall fail if:

37431 [EAGAIN] or [EWOULDBLOCK]

37432 The socket's file descriptor is marked O\_NONBLOCK and no data is waiting  
 37433 to be received; or MSG\_OOB is set and no out-of-band data is available and  
 37434 either the socket's file descriptor is marked O\_NONBLOCK or the socket does  
 37435 not support blocking to await out-of-band data.

37436 [EBADF] The *socket* argument is not a valid open file descriptor.

37437 [ECONNRESET] A connection was forcibly closed by a peer.

37438 [EINTR] This function was interrupted by a signal before any data was available.

37439 [EINVAL] The sum of the *iov\_len* values overflows a **ssize\_t**, or the MSG\_OOB flag is set  
 37440 and no out-of-band data is available.

37441 [EMSGSIZE] The *msg\_iovlen* member of the **msghdr** structure pointed to by *message* is less  
 37442 than or equal to 0, or is greater than {IOV\_MAX}.

37443 [ENOTCONN] A receive is attempted on a connection-mode socket that is not connected.

37444 [ENOTSOCK] The *socket* argument does not refer to a socket.

37445 [EOPNOTSUPP] The specified flags are not supported for this socket type.

37446 [ETIMEDOUT] The connection timed out during connection establishment, or due to a  
 37447 transmission timeout on active connection.

37448 The *recvmsg()* function may fail if:

37449 [EIO] An I/O error occurred while reading from or writing to the file system.

37450 [ENOBUFS] Insufficient resources were available in the system to perform the operation.

37451 [ENOMEM] Insufficient memory was available to fulfill the request.

37452 **EXAMPLES**

37453 None.

37454 **APPLICATION USAGE**37455 The *select()* and *poll()* functions can be used to determine when data is available to be received.37456 **RATIONALE**

37457 None.

37458 **FUTURE DIRECTIONS**

37459 None.

37460 **SEE ALSO**37461 *poll()*, *recv()*, *recvfrom()*, *select()*, *send()*, *sendmsg()*, *sendto()*, *shutdown()*, *socket()*, the Base

37462 Definitions volume of IEEE Std 1003.1-200x, &lt;sys/socket.h&gt;

37463 **CHANGE HISTORY**

37464 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

37465 **NAME**

37466 regcomp, regerror, regex, regfree — regular expression matching

37467 **SYNOPSIS**

```
37468 #include <regex.h>

37469 int regcomp(regex_t *restrict preg, const char *restrict pattern, int cflags);
37470 size_t regerror(int errcode, const regex_t *restrict preg,
37471 char *restrict errbuf, size_t errbuf_size);
37472 int regexexec(const regex_t *restrict preg, const char *restrict string,
37473 size_t rmatch, regmatch_t pmatch[restrict], int eflags);
37474 void regfree(regex_t *preg);
```

37475 **DESCRIPTION**

37476 These functions interpret *basic* and *extended* regular expressions as described in the Base  
37477 Definitions volume of IEEE Std 1003.1-200x, Chapter 9, Regular Expressions.

37478 The **regex\_t** structure is defined in **<regex.h>** and contains at least the following member:

37479  
37480  
37481

Member Type	Member Name	Description
size_t	re_nsub	Number of parenthesized subexpressions.

37482 The **regmatch\_t** structure is defined in **<regex.h>** and contains at least the following members:

37483  
37484  
37485  
37486  
37487

Member Type	Member Name	Description
regoff_t	rm_so	Byte offset from start of <i>string</i> to start of substring.
regoff_t	rm_eo	Byte offset from start of <i>string</i> of the first character after the end of substring.

37488 The *regcomp()* function shall compile the regular expression contained in the string pointed to by  
37489 the *pattern* argument and place the results in the structure pointed to by *preg*. The *cflags*  
37490 argument is the bitwise-inclusive OR of zero or more of the following flags, which are defined in  
37491 the **<regex.h>** header:

- 37492 REG\_EXTENDED Use Extended Regular Expressions.
- 37493 REG\_ICASE Ignore case in match. (See the Base Definitions volume of  
37494 IEEE Std 1003.1-200x, Chapter 9, Regular Expressions.)
- 37495 REG\_NOSUB Report only success/fail in *regexexec()*.
- 37496 REG\_NEWLINE Change the handling of <newline>s, as described in the text.

37497 The default regular expression type for *pattern* is a Basic Regular Expression. The application can  
37498 specify Extended Regular Expressions using the REG\_EXTENDED *cflags* flag.

37499 If the REG\_NOSUB flag was not set in *cflags*, then *regcomp()* shall set *re\_nsub* to the number of  
37500 parenthesized subexpressions (delimited by "\(\)" in basic regular expressions or "( )" in  
37501 extended regular expressions) found in *pattern*.

37502 The *regexexec()* function compares the null-terminated string specified by *string* with the compiled  
37503 regular expression *preg* initialized by a previous call to *regcomp()*. If it finds a match, *regexexec()*  
37504 shall return 0; otherwise, it shall return non-zero indicating either no match or an error. The  
37505 *eflags* argument is the bitwise-inclusive OR of zero or more of the following flags, which are  
37506 defined in the **<regex.h>** header:

37507 REG\_NOTBOL The first character of the string pointed to by *string* is not the beginning of the  
 37508 line. Therefore, the circumflex character ('^'), when taken as a special  
 37509 character, shall not match the beginning of *string*.

37510 REG\_NOTEOL The last character of the string pointed to by *string* is not the end of the line.  
 37511 Therefore, the dollar sign ('\$'), when taken as a special character, shall not  
 37512 match the end of *string*.

37513 If *nmatch* is 0 or REG\_NOSUB was set in the *cflags* argument to *regcomp()*, then *regexec()* shall  
 37514 ignore the *pmatch* argument. Otherwise, the application shall ensure that the *pmatch* argument  
 37515 points to an array with at least *nmatch* elements, and *regexec()* shall fill in the elements of that  
 37516 array with offsets of the substrings of *string* that correspond to the parenthesized subexpressions  
 37517 of *pattern*: *pmatch[i].rm\_so* shall be the byte offset of the beginning and *pmatch[i].rm\_eo* shall be  
 37518 one greater than the byte offset of the end of substring *i*. (Subexpression *i* begins at the *i*th  
 37519 matched open parenthesis, counting from 1.) Offsets in *pmatch[0]* identify the substring that  
 37520 corresponds to the entire regular expression. Unused elements of *pmatch* up to *pmatch[nmatch-1]*  
 37521 shall be filled with -1. If there are more than *nmatch* subexpressions in *pattern* (*pattern* itself  
 37522 counts as a subexpression), then *regexec()* shall still do the match, but shall record only the first  
 37523 *nmatch* substrings.

37524 When matching a basic or extended regular expression, any given parenthesized subexpression  
 37525 of *pattern* might participate in the match of several different substrings of *string*, or it might not  
 37526 match any substring even though the pattern as a whole did match. The following rules shall be  
 37527 used to determine which substrings to report in *pmatch* when matching regular expressions:

37528 1. If subexpression *i* in a regular expression is not contained within another subexpression,  
 37529 and it participated in the match several times, then the byte offsets in *pmatch[i]* shall  
 37530 delimit the last such match.

37531 2. If subexpression *i* is not contained within another subexpression, and it did not participate  
 37532 in an otherwise successful match, the byte offsets in *pmatch[i]* shall be -1. A subexpression  
 37533 does not participate in the match when:

37534 ' \* ' or "\{\}" appears immediately after the subexpression in a basic regular  
 37535 expression, or ' \* ', ' ? ', or "{ }" appears immediately after the subexpression in an  
 37536 extended regular expression, and the subexpression did not match (matched 0 times)

37537 or:

37538 ' | ' is used in an extended regular expression to select this subexpression or another,  
 37539 and the other subexpression matched.

37540 3. If subexpression *i* is contained within another subexpression *j*, and *i* is not contained  
 37541 within any other subexpression that is contained within *j*, and a match of subexpression *j*  
 37542 is reported in *pmatch[j]*, then the match or non-match of subexpression *i* reported in  
 37543 *pmatch[i]* shall be as described in 1. and 2. above, but within the substring reported in  
 37544 *pmatch[j]* rather than the whole string. The offsets in *pmatch[i]* are still relative to the start  
 37545 of string.

37546 4. If subexpression *i* is contained in subexpression *j*, and the byte offsets in *pmatch[j]* are -1,  
 37547 then the pointers in *pmatch[i]* shall also be -1.

37548 5. If subexpression *i* matched a zero-length string, then both byte offsets in *pmatch[i]* shall be  
 37549 the byte offset of the character or null terminator immediately following the zero-length  
 37550 string.

37551 If, when *regexec()* is called, the locale is different from when the regular expression was  
 37552 compiled, the result is undefined.

37553 If REG\_NEWLINE is not set in *cflags*, then a <newline> in *pattern* or *string* shall be treated as an  
 37554 ordinary character. If REG\_NEWLINE is set, then <newline> shall be treated as an ordinary  
 37555 character except as follows:

- 37556 1. A <newline> in *string* shall not be matched by a period outside a bracket expression or by  
 37557 any form of a non-matching list (see the Base Definitions volume of IEEE Std 1003.1-200x,  
 37558 Chapter 9, Regular Expressions).
- 37559 2. A circumflex ('^') in *pattern*, when used to specify expression anchoring (see the Base  
 37560 Definitions volume of IEEE Std 1003.1-200x, Section 9.3.8, BRE Expression Anchoring),  
 37561 shall match the zero-length string immediately after a <newline> in *string*, regardless of  
 37562 the setting of REG\_NOTBOL.
- 37563 3. A dollar sign ('\$') in *pattern*, when used to specify expression anchoring, shall match the  
 37564 zero-length string immediately before a <newline> in *string*, regardless of the setting of  
 37565 REG\_NOTEOL.

37566 The *regfree()* function frees any memory allocated by *regcomp()* associated with *preg*.

37567 The following constants are defined as error return values:

37568	REG_NOMATCH	<i>regexec()</i> failed to match.
37569	REG_BADPAT	Invalid regular expression.
37570	REG_ECOLLATE	Invalid collating element referenced.
37571	REG_ECTYPE	Invalid character class type referenced.
37572	REG_EESCAPE	Trailing '\\' in pattern.
37573	REG_ESUBREG	Number in "\digit" invalid or in error.
37574	REG_EBRACK	"[]" imbalance.
37575	REG_EPAREN	"\(\)" or "()" imbalance.
37576	REG_EBRACE	"\{\}" imbalance.
37577	REG_BADBR	Content of "\{\}" invalid: not a number, number too large, more than 37578 two numbers, first larger than second.
37579	REG_ERANGE	Invalid endpoint in range expression.
37580	REG_ESPACE	Out of memory.
37581	REG_BADRPT	'?', '*', or '+' not preceded by valid regular expression.

37582 The *regerror()* function provides a mapping from error codes returned by *regcomp()* and  
 37583 *regexec()* to unspecified printable strings. It generates a string corresponding to the value of the  
 37584 *errcode* argument, which the application shall ensure is the last non-zero value returned by  
 37585 *regcomp()* or *regexec()* with the given value of *preg*. If *errcode* is not such a value, the content of  
 37586 the generated string is unspecified.

37587 If *preg* is a null pointer, but *errcode* is a value returned by a previous call to *regexec()* or *regcomp()*,  
 37588 the *regerror()* still generates an error string corresponding to the value of *errcode*, but it might not  
 37589 be as detailed under some implementations.

37590 If the *errbuf\_size* argument is not 0, *regerror()* shall place the generated string into the buffer of  
 37591 size *errbuf\_size* bytes pointed to by *errbuf*. If the string (including the terminating null) cannot fit  
 37592 in the buffer, *regerror()* shall truncate the string and null-terminates the result.

37593 If *errbuf\_size* is 0, *regerror()* shall ignore the *errbuf* argument, and return the size of the buffer  
 37594 needed to hold the generated string.

37595 If the *preg* argument to *regexec()* or *regfree()* is not a compiled regular expression returned by  
 37596 *regcomp()*, the result is undefined. A *preg* is no longer treated as a compiled regular expression  
 37597 after it is given to *regfree()*.

#### 37598 RETURN VALUE

37599 Upon successful completion, the *regcomp()* function shall return 0. Otherwise, it shall return an  
 37600 integer value indicating an error as described in <**regex.h**>, and the content of *preg* is undefined.  
 37601 If a code is returned, the interpretation shall be as given in <**regex.h**>.

37602 If *regcomp()* detects an invalid RE, it may return REG\_BADPAT, or it may return one of the error  
 37603 codes that more precisely describes the error.

37604 Upon successful completion, the *regexec()* function shall return 0. Otherwise, it shall return  
 37605 REG\_NOMATCH to indicate no match.

37606 Upon successful completion, the *regerror()* function shall return the number of bytes needed to  
 37607 hold the entire generated string, including the null termination. If the return value is greater than  
 37608 *errbuf\_size*, the string returned in the buffer pointed to by *errbuf* has been truncated.

37609 The *regfree()* function shall not return a value.

#### 37610 ERRORS

37611 No errors are defined.

#### 37612 EXAMPLES

```

37613 #include <regex.h>
37614 /*
37615  * Match string against the extended regular expression in
37616  * pattern, treating errors as no match.
37617  *
37618  * Return 1 for match, 0 for no match.
37619  */
37620 int
37621 match(const char *string, char *pattern)
37622 {
37623     int    status;
37624     regex_t re;
37625     if (regcomp(&re, pattern, REG_EXTENDED|REG_NOSUB) != 0) {
37626         return(0); /* Report error. */
37627     }
37628     status = regexec(&re, string, (size_t) 0, NULL, 0);
37629     regfree(&re);
37630     if (status != 0) {
37631         return(0); /* Report error. */
37632     }
37633     return(1);
37634 }

```

37635 The following demonstrates how the REG\_NOTBOL flag could be used with *regexec()* to find all  
 37636 substrings in a line that match a pattern supplied by a user. (For simplicity of the example, very  
 37637 little error checking is done.)

```

37638     (void) regcomp (&re, pattern, 0);
37639     /* This call to regexec() finds the first match on the line. */
37640     error = regexec (&re, &buffer[0], 1, &pm, 0);
37641     while (error == 0) { /* While matches found. */
37642         /* Substring found between pm.rm_so and pm.rm_eo. */
37643         /* This call to regexec() finds the next match. */
37644         error = regexec (&re, buffer + pm.rm_eo, 1, &pm, REG_NOTBOL);
37645     }

```

#### 37646 APPLICATION USAGE

37647 An application could use:

```
37648     regerror(code, preg, (char *)NULL, (size_t)0)
```

37649 to find out how big a buffer is needed for the generated string, *malloc()* a buffer to hold the  
37650 string, and then call *regerror()* again to get the string. Alternatively, it could allocate a fixed,  
37651 static buffer that is big enough to hold most strings, and then use *malloc()* to allocate a larger  
37652 buffer if it finds that this is too small.

37653 To match a pattern as described in the Shell and Utilities volume of IEEE Std 1003.1-200x, Section  
37654 2.13, Pattern Matching Notation, use the *fnmatch()* function.

#### 37655 RATIONALE

37656 The *regmatch()* function must fill in all *nmatch* elements of *pmatch*, where *nmatch* and *pmatch* are  
37657 supplied by the application, even if some elements of *pmatch* do not correspond to  
37658 subexpressions in *pattern*. The application writer should note that there is probably no reason  
37659 for using a value of *nmatch* that is larger than *preg->re\_nsub+1*.

37660 The REG\_NEWLINE flag supports a use of RE matching that is needed in some applications like  
37661 text editors. In such applications, the user supplies an RE asking the application to find a line  
37662 that matches the given expression. An anchor in such an RE anchors at the beginning or end of  
37663 any line. Such an application can pass a sequence of <newline>-separated lines to *regexec()* as a  
37664 single long string and specify REG\_NEWLINE to *regcomp()* to get the desired behavior. The  
37665 application must ensure that there are no explicit <newline>s in *pattern* if it wants to ensure that  
37666 any match occurs entirely within a single line.

37667 The REG\_NEWLINE flag affects the behavior of *regexec()*, but it is in the *cflags* parameter to  
37668 *regcomp()* to allow flexibility of implementation. Some implementations will want to generate  
37669 the same compiled RE in *regcomp()* regardless of the setting of REG\_NEWLINE and have  
37670 *regexec()* handle anchors differently based on the setting of the flag. Other implementations will  
37671 generate different compiled REs based on the REG\_NEWLINE.

37672 The REG\_ICASE flag supports the operations taken by the *grep -i* option and the historical  
37673 implementations of *ex* and *vi*. Including this flag will make it easier for application code to be  
37674 written that does the same thing as these utilities.

37675 The substrings reported in *pmatch[]* are defined using offsets from the start of the string rather  
37676 than pointers. Since this is a new interface, there should be no impact on historical  
37677 implementations or applications, and offsets should be just as easy to use as pointers. The  
37678 change to offsets was made to facilitate future extensions in which the string to be searched is  
37679 presented to *regexec()* in blocks, allowing a string to be searched that is not all in memory at  
37680 once.

37681 A new type **regoff\_t** is used for the elements of *pmatch[]* to ensure that the application can  
37682 represent either the largest possible array in memory (important for an application conforming  
37683 to the Shell and Utilities volume of IEEE Std 1003.1-200x) or the largest possible file (important  
37684 for an application using the extension where a file is searched in chunks).

37685 The standard developers rejected the inclusion of a *regsub()* function that would be used to do  
37686 substitutions for a matched RE. While such a routine would be useful to some applications, its  
37687 utility would be much more limited than the matching function described here. Both RE parsing  
37688 and substitution are possible to implement without support other than that required by the  
37689 ISO C standard, but matching is much more complex than substituting. The only difficult part of  
37690 substitution, given the information supplied by *regexec()*, is finding the next character in a string  
37691 when there can be multi-byte characters. That is a much larger issue, and one that needs a more  
37692 general solution.

37693 The *errno* variable has not been used for error returns to avoid filling the *errno* name space for  
37694 this feature.

37695 The interface is defined so that the matched substrings *rm\_sp* and *rm\_ep* are in a separate  
37696 **regmatch\_t** structure instead of in **regex\_t**. This allows a single compiled RE to be used  
37697 simultaneously in several contexts; in *main()* and a signal handler, perhaps, or in multiple  
37698 threads of lightweight processes. (The *preg* argument to *regexec()* is declared with type **const**, so  
37699 the implementation is not permitted to use the structure to store intermediate results.) It also  
37700 allows an application to request an arbitrary number of substrings from an RE. The number of  
37701 subexpressions in the RE is reported in *re\_nsub* in *preg*. With this change to *regexec()*,  
37702 consideration was given to dropping the REG\_NOSUB flag since the user can now specify this  
37703 with a zero *nmatch* argument to *regexec()*. However, keeping REG\_NOSUB allows an  
37704 implementation to use a different (perhaps more efficient) algorithm if it knows in *regcomp()*  
37705 that no subexpressions need be reported. The implementation is only required to fill in *pmatch* if  
37706 *nmatch* is not zero and if REG\_NOSUB is not specified. Note that the **size\_t** type, as defined in  
37707 the ISO C standard, is unsigned, so the description of *regexec()* does not need to address  
37708 negative values of *nmatch*.

37709 REG\_NOTBOL was added to allow an application to do repeated searches for the same pattern  
37710 in a line. If the pattern contains a circumflex character that should match the beginning of a line,  
37711 then the pattern should only match when matched against the beginning of the line. Without  
37712 the REG\_NOTBOL flag, the application could rewrite the expression for subsequent matches,  
37713 but in the general case this would require parsing the expression. The need for REG\_NOTEOL is  
37714 not as clear; it was added for symmetry.

37715 The addition of the *regerror()* function addresses the historical need for conforming application  
37716 programs to have access to error information more than “Function failed to compile/match your  
37717 RE for unknown reasons”.

37718 This interface provides for two different methods of dealing with error conditions. The specific  
37719 error codes (REG\_EBRACE, for example), defined in **<regex.h>**, allow an application to recover  
37720 from an error if it is so able. Many applications, especially those that use patterns supplied by a  
37721 user, will not try to deal with specific error cases, but will just use *regerror()* to obtain a human-  
37722 readable error message to present to the user.

37723 The *regerror()* function uses a scheme similar to *confstr()* to deal with the problem of allocating  
37724 memory to hold the generated string. The scheme used by *strerror()* in the ISO C standard was  
37725 considered unacceptable since it creates difficulties for multi-threaded applications.

37726 The *preg* argument is provided to *regerror()* to allow an implementation to generate a more  
37727 descriptive message than would be possible with *errcode* alone. An implementation might, for  
37728 example, save the character offset of the offending character of the pattern in a field of *preg*, and  
37729 then include that in the generated message string. The implementation may also ignore *preg*.

37730 A REG\_FILENAME flag was considered, but omitted. This flag caused *regexec()* to match  
37731 patterns as described in the Shell and Utilities volume of IEEE Std 1003.1-200x, Section 2.13,  
37732 Pattern Matching Notation instead of REs. This service is now provided by the *fnmatch()*

- 37733 function.
- 37734 Notice that there is a difference in philosophy between the ISO POSIX-2:1993 standard and  
37735 IEEE Std 1003.1-200x in how to handle a bad regular expression. The ISO POSIX-2:1993 standard  
37736 says that many bad constructs produce undefined results, or that the interpretation is undefined.  
37737 IEEE Std 1003.1-200x, however, says that the interpretation of such REs is unspecified. The term  
37738 “undefined” means that the action by the application is an error, of similar severity to passing a  
37739 bad pointer to a function.
- 37740 The *regcomp()* and *regexec()* functions are required to accept any null-terminated string as the  
37741 *pattern* argument. If the meaning of the string is undefined, the behavior of the function is  
37742 unspecified. IEEE Std 1003.1-200x does not specify how the functions will interpret the pattern;  
37743 they might return error codes, or they might do pattern matching in some completely  
37744 unexpected way, but they should not do something like abort the process.
- 37745 **FUTURE DIRECTIONS**
- 37746 None.
- 37747 **SEE ALSO**
- 37748 *fnmatch()*, *glob()*, the Base Definitions volume of IEEE Std 1003.1-200x, `<regex.h>`, `<sys/types.h>`
- 37749 **CHANGE HISTORY**
- 37750 First released in Issue 4. Derived from the ISO POSIX-2 standard.
- 37751 **Issue 5**
- 37752 Moved from POSIX2 C-language Binding to BASE.
- 37753 **Issue 6**
- 37754 In the SYNOPSIS, the optional include of the `<sys/types.h>` header is removed.
- 37755 The following new requirements on POSIX implementations derive from alignment with the  
37756 Single UNIX Specification:
- 37757 • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was  
37758 required for conforming implementations of previous POSIX specifications, it was not  
37759 required for UNIX applications.
- 37760 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.
- 37761 The REG\_ENOSYS constant is removed.
- 37762 The **restrict** keyword is added to the *regcomp()*, *regerror()*, and *regexec()* prototypes for  
37763 alignment with the ISO/IEC 9899:1999 standard.

37764 **NAME**

37765 remainder, remainderf, remainderl — remainder function

37766 **SYNOPSIS**

37767 #include &lt;math.h&gt;

37768 double remainder(double x, double y);

37769 float remainderf(float x, float y);

37770 long double remainderl(long double x, long double y);

37771 **DESCRIPTION**

37772 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 37773 conflict between the requirements described here and the ISO C standard is unintentional. This  
 37774 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

37775 These functions shall return the floating-point remainder  $r=x-ny$  when  $y$  is non-zero. The value  
 37776  $n$  is the integral value nearest the exact value  $x/y$ . When  $|n-x/y| = 1/2$ , the value  $n$  is chosen to  
 37777 be even.

37778 The behavior of *remainder()* shall be independent of the rounding mode.37779 **RETURN VALUE**37780 Upon successful completion, these functions shall return the floating-point remainder  $r=x-ny$   
37781 when  $y$  is non-zero.37782 **MX** If  $x$  or  $y$  is NaN, a NaN shall be returned.37783 If  $x$  is infinite or  $y$  is 0 and the other is non-NaN, a domain error shall occur, and either a NaN (if  
37784 supported), or an implementation-defined value shall be returned.37785 **ERRORS**

37786 These functions shall fail if:

37787 <b>MX</b>	Domain Error	The $x$ argument is $\pm\text{Inf}$ , or the $y$ argument is $\pm 0$ and the other argument is non-NaN.
-----------------	--------------	---

37789	37790	37791	37792	

37793 **EXAMPLES**

37794 None.

37795 **APPLICATION USAGE**37796 On error, the expressions (math\_errhandling & MATH\_ERRNO) and (math\_errhandling &  
37797 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.37798 **RATIONALE**

37799 None.

37800 **FUTURE DIRECTIONS**

37801 None.

37802 **SEE ALSO**37803 *abs()*, *div()*, *feclearexcept()*, *fetestexcept()*, *ldiv()*, the Base Definitions volume of  
37804 IEEE Std 1003.1-200x, Section 4.18, Treatment of Error Conditions for Mathematical Functions,  
37805 <math.h>

37806 **CHANGE HISTORY**

37807 First released in Issue 4, Version 2.

37808 **Issue 5**

37809 Moved from X/OPEN UNIX extension to BASE.

37810 **Issue 6**

37811 The *remainder()* function is no longer marked as an extension.

37812 The *remainderf()* and *remainderl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.  
37813

37814 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
37815 revised to align with the ISO/IEC 9899:1999 standard.

37816 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
37817 marked.

37818 **NAME**

37819 remove — remove a file

37820 **SYNOPSIS**

37821 #include &lt;stdio.h&gt;

37822 int remove(const char \*path);

37823 **DESCRIPTION**

37824 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 37825 conflict between the requirements described here and the ISO C standard is unintentional. This  
 37826 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

37827 The *remove()* function shall cause the file named by the pathname pointed to by *path* to be no  
 37828 longer accessible by that name. A subsequent attempt to open that file using that name shall fail,  
 37829 unless it is created anew.

37830 CX If *path* does not name a directory, *remove(path)* shall be equivalent to *unlink(path)*.

37831 If *path* names a directory, *remove(path)* shall be equivalent to *rmdir(path)*.

37832 **RETURN VALUE**37833 CX Refer to *rmdir()* or *unlink()*.37834 **ERRORS**37835 CX Refer to *rmdir()* or *unlink()*.37836 **EXAMPLES**37837 **Removing Access to a File**37838 The following example shows how to remove access to a file named `/home/cnd/old_mods`.

37839 #include &lt;stdio.h&gt;

37840 int status;

37841 ...

37842 status = remove("/home/cnd/old\_mods");

37843 **APPLICATION USAGE**

37844 None.

37845 **RATIONALE**

37846 None.

37847 **FUTURE DIRECTIONS**

37848 None.

37849 **SEE ALSO**37850 *rmdir()*, *unlink()*, the Base Definitions volume of IEEE Std 1003.1-200x, <stdio.h>37851 **CHANGE HISTORY**

37852 First released in Issue 3.

37853 Entry included for alignment with the POSIX.1-1988 standard and the ISO C standard.

37854 **Issue 6**

37855 Extensions beyond the ISO C standard are now marked.

37856 The following new requirements on POSIX implementations derive from alignment with the  
 37857 Single UNIX Specification:

37858  
37859  
37860

- The DESCRIPTION, RETURN VALUE, and ERRORS sections are updated so that if *path* is not a directory, *remove()* is equivalent to *unlink()*, and if it is a directory, it is equivalent to *rmdir()*.

37861 **NAME**

37862       remque — remove an element from a queue

37863 **SYNOPSIS**

37864 xSI       #include &lt;search.h&gt;

37865       void remque(void \**element*);

37866

37867 **DESCRIPTION**37868       Refer to *insque()*.

37869 **NAME**

37870 remquo, remquof, remquol — remainder functions

37871 **SYNOPSIS**

37872 #include &lt;math.h&gt;

37873 double remquo(double x, double y, int \*quo);

37874 float remquof(float x, float y, int \*quo);

37875 long double remquol(long double x, long double y, int \*quo);

37876 **DESCRIPTION**

37877 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 37878 conflict between the requirements described here and the ISO C standard is unintentional. This  
 37879 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

37880 The *remquo()*, *remquof()*, and *remquol()* functions shall compute the same remainder as the  
 37881 *remainder()*, *remainderf()*, and *remainderl()* functions, respectively. In the object pointed to by  
 37882 *quo*, they store a value whose sign is the sign of  $x/y$  and whose magnitude is congruent modulo  
 37883  $2^n$  to the magnitude of the integral quotient of  $x/y$ , where  $n$  is an implementation-defined  
 37884 integer greater than or equal to 3.

37885 An application wishing to check for error situations should set *errno* to zero and call  
 37886 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 37887 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 37888 zero, an error has occurred.

37889 **RETURN VALUE**37890 These functions shall return  $x \text{ REM } y$ .37891 **MX** If  $x$  or  $y$  is NaN, a NaN shall be returned.

37892 If  $x$  is  $\pm\text{Inf}$  or  $y$  is zero and the other argument is non-NaN, a domain error shall occur, and either  
 37893 a NaN (if supported), or an implementation-defined value shall be returned.

37894 **ERRORS**

37895 These functions shall fail if:

37896 **MX** Domain Error The  $x$  argument is  $\pm\text{Inf}$ , or the  $y$  argument is  $\pm 0$  and the other argument is  
 37897 non-NaN.

37898 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero, |  
 37899 then *errno* shall be set to [EDOM]. If the integer expression (*math\_errhandling* |  
 37900 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception |  
 37901 shall be raised. |

37902 **EXAMPLES**

37903 None.

37904 **APPLICATION USAGE**

37905 On error, the expressions (*math\_errhandling* & MATH\_ERRNO) and (*math\_errhandling* &  
 37906 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

37907 **RATIONALE**

37908 These functions are intended for implementing argument reductions which can exploit a few  
 37909 low-order bits of the quotient. Note that  $x$  may be so large in magnitude relative to  $y$  that an  
 37910 exact representation of the quotient is not practical.

37911 **FUTURE DIRECTIONS**

37912 None.

37913 **SEE ALSO**

37914 *feclearexcept()*, *fetetestexcept()*, *remainder()*, the Base Definitions volume of IEEE Std 1003.1-200x, |  
37915 Section 4.18, Treatment of Error Conditions for Mathematical Functions, <math.h> |

37916 **CHANGE HISTORY**

37917 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

## 37918 NAME

37919 rename — rename a file

## 37920 SYNOPSIS

37921 #include &lt;stdio.h&gt;

37922 int rename(const char \*old, const char \*new);

## 37923 DESCRIPTION

37924 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
 37925 conflict between the requirements described here and the ISO C standard is unintentional. This  
 37926 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

37927 The *rename()* function shall change the name of a file. The *old* argument points to the pathname  
 37928 of the file to be renamed. The *new* argument points to the new pathname of the file.

37929 cx If either the *old* or *new* argument names a symbolic link, *rename()* shall operate on the symbolic  
 37930 link itself, and shall not resolve the last component of the argument. If the *old* argument and the  
 37931 *new* argument resolve to the same existing file, *rename()* shall return successfully and perform no  
 37932 other action.

37933 If the *old* argument points to the pathname of a file that is not a directory, the *new* argument shall  
 37934 not point to the pathname of a directory. If the link named by the *new* argument exists, it shall be  
 37935 removed and *old* renamed to *new*. In this case, a link named *new* shall remain visible to other  
 37936 processes throughout the renaming operation and refer either to the file referred to by *new* or *old*  
 37937 before the operation began. Write access permission is required for both the directory containing  
 37938 *old* and the directory containing *new*.

37939 If the *old* argument points to the pathname of a directory, the *new* argument shall not point to the  
 37940 pathname of a file that is not a directory. If the directory named by the *new* argument exists, it  
 37941 shall be removed and *old* renamed to *new*. In this case, a link named *new* shall exist throughout  
 37942 the renaming operation and shall refer either to the directory referred to by *new* or *old* before the  
 37943 operation began. If *new* names an existing directory, it shall be required to be an empty directory.

37944 If the *old* argument points to a pathname of a symbolic link, the symbolic link shall be renamed.  
 37945 If the *new* argument points to a pathname of a symbolic link, the symbolic link shall be removed.

37946 The *new* pathname shall not contain a path prefix that names *old*. Write access permission is  
 37947 required for the directory containing *old* and the directory containing *new*. If the *old* argument  
 37948 points to the pathname of a directory, write access permission may be required for the directory  
 37949 named by *old*, and, if it exists, the directory named by *new*.

37950 If the link named by the *new* argument exists and the file's link count becomes 0 when it is  
 37951 removed and no process has the file open, the space occupied by the file shall be freed and the  
 37952 file shall no longer be accessible. If one or more processes have the file open when the last link is  
 37953 removed, the link shall be removed before *rename()* returns, but the removal of the file contents  
 37954 shall be postponed until all references to the file are closed.

37955 Upon successful completion, *rename()* shall mark for update the *st\_ctime* and *st\_mtime* fields of  
 37956 the parent directory of each file.

37957 If the *rename()* function fails for any reason other than [EIO], any file named by *new* shall be  
 37958 unaffected.

## 37959 RETURN VALUE

37960 cx Upon successful completion, *rename()* shall return 0; otherwise,  $-1$  shall be returned, *errno* shall  
 37961 be set to indicate the error, and neither the file named by *old* nor the file named by *new* shall be  
 37962 changed or created.

37963 **ERRORS**37964 The *rename()* function shall fail if:

37965 CX [EACCES] A component of either path prefix denies search permission; or one of the  
 37966 directories containing *old* or *new* denies write permissions; or, write  
 37967 permission is required and is denied for a directory pointed to by the *old* or  
 37968 *new* arguments.

37969 CX [EBUSY] The directory named by *old* or *new* is currently in use by the system or another  
 37970 process, and the implementation considers this an error.

37971 CX [EEXIST] or [ENOTEMPTY]  
 37972 The link named by *new* is a directory that is not an empty directory.

37973 CX [EINVAL] The *new* directory pathname contains a path prefix that names the *old* |  
 37974 directory.

37975 CX [EIO] A physical I/O error has occurred.

37976 CX [EISDIR] The *new* argument points to a directory and the *old* argument points to a file  
 37977 that is not a directory.

37978 CX [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*  
 37979 argument.

37980 CX [EMLINK] The file named by *old* is a directory, and the link count of the parent directory  
 37981 of *new* would exceed {LINK\_MAX}.

37982 CX [ENAMETOOLONG]  
 37983 The length of the *old* or *new* argument exceeds {PATH\_MAX} or a pathname |  
 37984 component is longer than {NAME\_MAX}.

37985 CX [ENOENT] The link named by *old* does not name an existing file, or either *old* or *new*  
 37986 points to an empty string.

37987 CX [ENOSPC] The directory that would contain *new* cannot be extended.

37988 CX [ENOTDIR] A component of either path prefix is not a directory; or the *old* argument  
 37989 names a directory and *new* argument names a non-directory file.

37990 XSI [EPERM] or [EACCES]  
 37991 The S\_ISVTX flag is set on the directory containing the file referred to by *old*  
 37992 and the caller is not the file owner, nor is the caller the directory owner, nor  
 37993 does the caller have appropriate privileges; or *new* refers to an existing file, the  
 37994 S\_ISVTX flag is set on the directory containing this file, and the caller is not  
 37995 the file owner, nor is the caller the directory owner, nor does the caller have  
 37996 appropriate privileges.

37997 CX [EROFS] The requested operation requires writing in a directory on a read-only file  
 37998 system.

37999 CX [EXDEV] The links named by *new* and *old* are on different file systems and the  
 38000 implementation does not support links between file systems.

38001 The *rename()* function may fail if:

38002 XSI [EBUSY] The file named by the *old* or *new* arguments is a named STREAM.

38003 CX [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
 38004 resolution of the *path* argument.

38005 CX [ENAMETOOLONG]  
 38006 As a result of encountering a symbolic link in resolution of the *path* argument,  
 38007 the length of the substituted pathname string exceeded {PATH\_MAX}. |

38008 CX [ETXTBSY] The file to be renamed is a pure procedure (shared text) file that is being  
 38009 executed. |

38010 **EXAMPLES**38011 **Renaming a File**

38012 The following example shows how to rename a file named `/home/cnd/mod1` to  
 38013 `/home/cnd/mod2`.

```
38014 #include <stdio.h>
38015 int status;
38016 ...
38017 status = rename("/home/cnd/mod1", "/home/cnd/mod2");
```

38018 **APPLICATION USAGE**

38019 Some implementations mark for update the *st\_ctime* field of renamed files and some do not. |  
 38020 Applications which make use of the *st\_ctime* field may behave differently with respect to |  
 38021 renamed files unless they are designed to allow for either behavior. |

38022 **RATIONALE**

38023 This *rename()* function is equivalent for regular files to that defined by the ISO C standard. Its  
 38024 inclusion here expands that definition to include actions on directories and specifies behavior  
 38025 when the *new* parameter names a file that already exists. That specification requires that the  
 38026 action of the function be atomic.

38027 One of the reasons for introducing this function was to have a means of renaming directories  
 38028 while permitting implementations to prohibit the use of *link()* and *unlink()* with directories,  
 38029 thus constraining links to directories to those made by *mkdir()*.

38030 The specification that if *old* and *new* refer to the same file is intended to guarantee that:

```
38031 rename("x", "x");
```

38032 does not remove the file.

38033 Renaming dot or dot-dot is prohibited in order to prevent cyclical file system paths.

38034 See also the descriptions of [ENOTEMPTY] and [ENAMETOOLONG] in *rmdir()* and [EBUSY] in  
 38035 *unlink()*. For a discussion of [EXDEV], see *link()*.

38036 **FUTURE DIRECTIONS**

38037 None.

38038 **SEE ALSO**

38039 *link()*, *rmdir()*, *symlink()*, *unlink()*, the Base Definitions volume of IEEE Std 1003.1-200x,  
 38040 `<stdio.h>`

38041 **CHANGE HISTORY**

38042 First released in Issue 3.

38043 Entry included for alignment with the POSIX.1-1988 standard.

38044 **Issue 5**

38045 The [EBUSY] error is added to the “may fail” part of the ERRORS section.

38046 **Issue 6**

38047 Extensions beyond the ISO C standard are now marked.

38048 The following new requirements on POSIX implementations derive from alignment with the |  
38049 Single UNIX Specification:

- 38050 • The [EIO] mandatory error condition is added.
- 38051 • The [ELOOP] mandatory error condition is added.
- 38052 • A second [ENAMETOOLONG] is added as an optional error condition.
- 38053 • The [ETXTBSY] optional error condition is added.

38054 The following changes were made to align with the IEEE P1003.1a draft standard:

- 38055 • Details are added regarding the treatment of symbolic links.
- 38056 • The [ELOOP] optional error condition is added.

38057 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

38058 **NAME**

38059           rewind — reset file position indicator in a stream

38060 **SYNOPSIS**

38061           #include &lt;stdio.h&gt;

38062           void rewind(FILE \*stream);

38063 **DESCRIPTION**

38064 cx       The functionality described on this reference page is aligned with the ISO C standard. Any  
38065       conflict between the requirements described here and the ISO C standard is unintentional. This  
38066       volume of IEEE Std 1003.1-200x defers to the ISO C standard.

38067       The call:

38068       rewind(stream)

38069       shall be equivalent to:

38070       (void) fseek(stream, 0L, SEEK\_SET)

38071       except that *rewind()* shall also clear the error indicator. |

38072 cx       Since *rewind()* does not return a value, an application wishing to detect errors should clear *errno*, |  
38073       then call *rewind()*, and if *errno* is non-zero, assume an error has occurred.

38074 **RETURN VALUE**38075       The *rewind()* function shall not return a value.38076 **ERRORS**38077 cx       Refer to *fseek()* with the exception of [EINVAL] which does not apply.38078 **EXAMPLES**

38079       None.

38080 **APPLICATION USAGE**

38081       None.

38082 **RATIONALE**

38083       None.

38084 **FUTURE DIRECTIONS**

38085       None.

38086 **SEE ALSO**38087       *fseek()*, the Base Definitions volume of IEEE Std 1003.1-200x, <stdio.h>38088 **CHANGE HISTORY**

38089       First released in Issue 1. Derived from Issue 1 of the SVID.

38090 **Issue 6**

38091       Extensions beyond the ISO C standard are now marked.

38092 **NAME**

38093        rewinddir — reset position of directory stream to the beginning of a directory

38094 **SYNOPSIS**

38095        #include <dirent.h>

38096        void rewinddir(DIR \*dirp);

38097 **DESCRIPTION**

38098        The *rewinddir()* function shall reset the position of the directory stream to which *dirp* refers to the beginning of the directory. It shall also cause the directory stream to refer to the current state of the corresponding directory, as a call to *opendir()* would have done. If *dirp* does not refer to a directory stream, the effect is undefined.

38102        After a call to the *fork()* function, either the parent or child (but not both) may continue processing the directory stream using *readdir()*, *rewinddir()*, or *seekdir()*. If both the parent and child processes use these functions, the result is undefined.

38105 **RETURN VALUE**

38106        The *rewinddir()* function shall not return a value.

38107 **ERRORS**

38108        No errors are defined.

38109 **EXAMPLES**

38110        None.

38111 **APPLICATION USAGE**

38112        The *rewinddir()* function should be used in conjunction with *opendir()*, *readdir()*, and *closedir()* to examine the contents of the directory. This method is recommended for portability.

38114 **RATIONALE**

38115        None.

38116 **FUTURE DIRECTIONS**

38117        None.

38118 **SEE ALSO**

38119        *closedir()*, *opendir()*, *readdir()*, the Base Definitions volume of IEEE Std 1003.1-200x, <dirent.h>  
38120        <sys/types.h>

38121 **CHANGE HISTORY**

38122        First released in Issue 2.

38123 **Issue 6**

38124        In the SYNOPSIS, the optional include of the <sys/types.h> header is removed.

38125        The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- 38127        • The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.

38130 **NAME**38131 rindex — character string operations (**LEGACY**)38132 **SYNOPSIS**38133 XSI `#include <strings.h>`38134 `char *rindex(const char *s, int c);`

38135

38136 **DESCRIPTION**38137 The *rindex()* function shall be equivalent to *strchr()*.38138 **RETURN VALUE**38139 Refer to *strchr()*.38140 **ERRORS**38141 Refer to *strchr()*.38142 **EXAMPLES**

38143 None.

38144 **APPLICATION USAGE**38145 *strchr()* is preferred over this function.38146 For maximum portability, it is recommended to replace the function call to *rindex()* as follows:38147 `#define rindex(a,b) strchr((a),(b))`38148 **RATIONALE**

38149 None.

38150 **FUTURE DIRECTIONS**

38151 This function may be withdrawn in a future version.

38152 **SEE ALSO**38153 *strchr()*, the Base Definitions volume of IEEE Std 1003.1-200x, `<strings.h>`38154 **CHANGE HISTORY**

38155 First released in Issue 4, Version 2.

38156 **Issue 5**

38157 Moved from X/OPEN UNIX extension to BASE.

38158 **Issue 6**

38159 This function is marked LEGACY.

38160 **NAME**

38161 rint, rintf, rintl — round-to-nearest integral value

38162 **SYNOPSIS**

38163 #include &lt;math.h&gt;

38164 double rint(double x);

38165 float rintf(float x);

38166 long double rintl(long double x);

38167 **DESCRIPTION**

38168 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 38169 conflict between the requirements described here and the ISO C standard is unintentional. This  
 38170 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

38171 These functions shall return the integral value (represented as a **double**) nearest *x* in the  
 38172 direction of the current rounding mode. The current rounding mode is implementation-defined.

38173 If the current rounding mode rounds toward negative infinity, then *rint()* shall be equivalent to  
 38174 *floor()*. If the current rounding mode rounds toward positive infinity, then *rint()* shall be  
 38175 equivalent to *ceil()*.

38176 These functions differ from the *nearbyint()*, *nearbyintf()*, and *nearbyintl()* functions only in that  
 38177 they may raise the inexact floating-point exception if the result differs in value from the  
 38178 argument.

38179 An application wishing to check for error situations should set *errno* to zero and call  
 38180 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 38181 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 38182 zero, an error has occurred.

38183 **RETURN VALUE**

38184 Upon successful completion, these functions shall return the integer (represented as a double  
 38185 precision number) nearest *x* in the direction of the current rounding mode.

38186 **MX** If *x* is NaN, a NaN shall be returned.

38187 If *x* is  $\pm 0$ , or  $\pm \text{Inf}$ , *x* shall be returned.

38188 **XSI** If the correct value would cause overflow, a range error shall occur and *rint()*, *rintf()*, and *rintl()*  
 38189 shall return the value of the macro HUGE\_VAL, HUGE\_VALF, and HUGE\_VALL, respectively.

38190 **ERRORS**

38191 These functions shall fail if:

38192 **XSI** **Range Error** The result would cause an overflow.

38193 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero,  
 38194 then *errno* shall be set to [ERANGE]. If the integer expression  
 38195 (math\_errhandling & MATH\_ERREXCEPT) is non-zero, then the overflow  
 38196 floating-point exception shall be raised.

38197 **EXAMPLES**

38198 None.

38199 **APPLICATION USAGE**

38200 On error, the expressions `(math_errhandling & MATH_ERRNO)` and `(math_errhandling &`  
38201 `MATH_ERREXCEPT)` are independent of each other, but at least one of them must be non-zero.

38202 **RATIONALE**

38203 None.

38204 **FUTURE DIRECTIONS**

38205 None.

38206 **SEE ALSO**

38207 *abs()*, *ceil()*, *feclearexcept()*, *fetestexcept()*, *nearbyint()*, *floor()*, *isnan()*, the Base Definitions volume |  
38208 of IEEE Std 1003.1-200x, Section 4.18, Treatment of Error Conditions for Mathematical Functions, |  
38209 `<math.h>`

38210 **CHANGE HISTORY**

38211 First released in Issue 4, Version 2.

38212 **Issue 5**

38213 Moved from X/OPEN UNIX extension to BASE.

38214 **Issue 6**

38215 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 38216 • The *rintf()* and *rintl()* functions are added.
- 38217 • The *rint()* function is no longer marked as an extension.
- 38218 • The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
38219 revised to align with the ISO/IEC 9899:1999 standard.
- 38220 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
38221 marked.

38222 **NAME**

38223 rmdir — remove a directory

38224 **SYNOPSIS**

38225 #include &lt;unistd.h&gt;

38226 int rmdir(const char \*path);

38227 **DESCRIPTION**

38228 The *rmdir()* function shall remove a directory whose name is given by *path*. The directory shall  
 38229 be removed only if it is an empty directory. |

38230 If the directory is the root directory or the current working directory of any process, it is  
 38231 unspecified whether the function succeeds, or whether it shall fail and set *errno* to [EBUSY].

38232 If *path* names a symbolic link, then *rmdir()* shall fail and set *errno* to [ENOTDIR].

38233 If the *path* argument refers to a path whose final component is either dot or dot-dot, *rmdir()* shall  
 38234 fail.

38235 If the directory's link count becomes 0 and no process has the directory open, the space occupied  
 38236 by the directory shall be freed and the directory shall no longer be accessible. If one or more  
 38237 processes have the directory open when the last link is removed, the dot and dot-dot entries, if  
 38238 present, shall be removed before *rmdir()* returns and no new entries may be created in the  
 38239 directory, but the directory shall not be removed until all references to the directory are closed.

38240 If the directory is not an empty directory, *rmdir()* shall fail and set *errno* to [EEXIST] or  
 38241 [ENOTEMPTY].

38242 Upon successful completion, the *rmdir()* function shall mark for update the *st\_ctime* and  
 38243 *st\_mtime* fields of the parent directory.

38244 **RETURN VALUE**

38245 Upon successful completion, the function *rmdir()* shall return 0. Otherwise, -1 shall be returned,  
 38246 and *errno* set to indicate the error. If -1 is returned, the named directory shall not be changed.

38247 **ERRORS**

38248 The *rmdir()* function shall fail if:

38249 [EACCES] Search permission is denied on a component of the path prefix, or write  
 38250 permission is denied on the parent directory of the directory to be removed.

38251 [EBUSY] The directory to be removed is currently in use by the system or some process  
 38252 and the implementation considers this to be an error.

38253 [EEXIST] or [ENOTEMPTY]  
 38254 The *path* argument names a directory that is not an empty directory, or there  
 38255 are hard links to the directory other than dot or a single entry in dot-dot.

38256 [EINVAL] The *path* argument contains a last component that is dot.

38257 [EIO] A physical I/O error has occurred.

38258 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*  
 38259 argument.

38260 [ENAMETOOLONG]  
 38261 The length of the *path* argument exceeds {PATH\_MAX} or a pathname  
 38262 component is longer than |

38263 NAME\_MAX

38264	[ENOENT]	A component of <i>path</i> does not name an existing file, or the <i>path</i> argument names a nonexistent directory or points to an empty string.
38265		
38266	[ENOTDIR]	A component of <i>path</i> is not a directory.
38267 XSI	[EPERM] or [EACCES]	
38268		The S_ISVTX flag is set on the parent directory of the directory to be removed and the caller is not the owner of the directory to be removed, nor is the caller the owner of the parent directory, nor does the caller have the appropriate privileges.
38269		
38270		
38271		
38272	[EROFS]	The directory entry to be removed resides on a read-only file system.
38273		The <i>rmdir()</i> function may fail if:
38274	[ELOOP]	More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the <i>path</i> argument.
38275		
38276	[ENAMETOOLONG]	
38277		As a result of encountering a symbolic link in resolution of the <i>path</i> argument,
38278		the length of the substituted pathname string exceeded {PATH_MAX}.

### 38279 EXAMPLES

#### 38280 Removing a Directory

38281 The following example shows how to remove a directory named `/home/cnd/mod1`.

```
38282 #include <unistd.h>
38283
38284 int status;
38285 ...
38286 status = rmdir("/home/cnd/mod1");
```

### 38286 APPLICATION USAGE

38287 None.

### 38288 RATIONALE

38289 The *rmdir()* and *rename()* functions originated in 4.2 BSD, and they used [ENOTEMPTY] for the condition when the directory to be removed does not exist or *new* already exists. When the 1984 /usr/group standard was published, it contained [EEXIST] instead. When these functions were adopted into System V, the 1984 /usr/group standard was used as a reference. Therefore, several existing applications and implementations support/use both forms, and no agreement could be reached on either value. All implementations are required to supply both [EEXIST] and [ENOTEMPTY] in `<errno.h>` with distinct values, so that applications can use both values in C-language `case` statements.

38297 The meaning of deleting *pathname/dot* is unclear, because the name of the file (directory) in the parent directory to be removed is not clear, particularly in the presence of multiple links to a directory.

38300 IEEE Std 1003.1-200x was silent with regard to the behavior of *rmdir()* when there are multiple hard links to the directory being removed. The requirement to set *errno* to [EEXIST] or [ENOTEMPTY] clarifies the behavior in this case.

38303 If the process' current working directory is being removed, that should be an allowed error.

38304 Virtually all existing implementations detect [ENOTEMPTY] or the case of dot-dot. The text in Section 2.3 (on page 471) about returning any one of the possible errors permits that behavior to continue. The [ELOOP] error may be returned if more than {SYMLOOP\_MAX} symbolic links

38307 are encountered during resolution of the *path* argument.

38308 **FUTURE DIRECTIONS**

38309 None.

38310 **SEE ALSO**

38311 *mkdir()*, *remove()*, *unlink()*, the Base Definitions volume of IEEE Std 1003.1-200x, <unistd.h>

38312 **CHANGE HISTORY**

38313 First released in Issue 3.

38314 Entry included for alignment with the POSIX.1-1988 standard.

38315 **Issue 6**

38316 The following new requirements on POSIX implementations derive from alignment with the |  
38317 Single UNIX Specification:

- 38318 • The DESCRIPTION is updated to indicate the results of naming a symbolic link in *path*.
- 38319 • The [EIO] mandatory error condition is added.
- 38320 • The [ELOOP] mandatory error condition is added.
- 38321 • A second [ENAMETOOLONG] is added as an optional error condition.

38322 The following changes were made to align with the IEEE P1003.1a draft standard:

- 38323 • The [ELOOP] optional error condition is added.

38324 **NAME**

38325 round, roundf, roundl — round to nearest integer value in floating-point format

38326 **SYNOPSIS**

38327 #include &lt;math.h&gt;

38328 double round(double x);

38329 float roundf(float x);

38330 long double roundl(long double x);

38331 **DESCRIPTION**38332 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
38333 conflict between the requirements described here and the ISO C standard is unintentional. This  
38334 volume of IEEE Std 1003.1-200x defers to the ISO C standard.38335 These functions shall round their argument to the nearest integer value in floating-point format,  
38336 rounding halfway cases away from zero, regardless of the current rounding direction.38337 An application wishing to check for error situations should set *errno* to zero and call  
38338 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
38339 *etestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
38340 zero, an error has occurred.38341 **RETURN VALUE**

38342 Upon successful completion, these functions shall return the rounded integer value.

38343 **MX** If *x* is NaN, a NaN shall be returned.38344 If *x* is  $\pm 0$ , or  $\pm \text{Inf}$ , *x* shall be returned.38345 **XSI** If the correct value would cause overflow, a range error shall occur and *round()*, *roundf()*, and  
38346 *roundl()* shall return the value of the macro HUGE\_VAL, HUGE\_VALF, and HUGE\_VALL,  
38347 respectively.38348 **ERRORS**

38349 These functions may fail if:

38350 **XSI** **Range Error** The result overflows.38351 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero, |  
38352 then *errno* shall be set to [ERANGE]. If the integer expression |  
38353 (math\_errhandling & MATH\_ERREXCEPT) is non-zero, then the overflow |  
38354 floating-point exception shall be raised. |38355 **EXAMPLES**

38356 None.

38357 **APPLICATION USAGE**38358 On error, the expressions (math\_errhandling & MATH\_ERRNO) and (math\_errhandling &  
38359 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.38360 **RATIONALE**

38361 None.

38362 **FUTURE DIRECTIONS**

38363 None.

38364 **SEE ALSO**

38365 *feclearexcept()*, *fetestexcept()*, the Base Definitions volume of IEEE Std 1003.1-200x, Section 4.18, |  
38366 Treatment of Error Conditions for Mathematical Functions, <math.h> |

38367 **CHANGE HISTORY**

38368 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

## 38369 NAME

38370 scalb — load exponent of a radix-independent floating-point number

## 38371 SYNOPSIS

38372 OB XSI #include &lt;math.h&gt;

38373 double scalb(double x, double n);

38374

## 38375 DESCRIPTION

38376 The *scalb()* function shall compute  $x \cdot r^n$ , where  $r$  is the radix of the machine's floating-point  
 38377 arithmetic. When  $r$  is 2, *scalb()* shall be equivalent to *ldexp()*. The value of  $r$  is FLT\_RADIX  
 38378 which is defined in <float.h>.

38379 An application wishing to check for error situations should set *errno* to zero and call  
 38380 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 38381 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 38382 zero, an error has occurred.

## 38383 RETURN VALUE

38384 Upon successful completion, the *scalb()* function shall return  $x \cdot r^n$ .38385 If  $x$  or  $n$  is NaN, a NaN shall be returned.38386 If  $n$  is zero,  $x$  shall be returned.38387 If  $x$  is  $\pm\text{Inf}$  and  $n$  is not  $-\text{Inf}$ ,  $x$  shall be returned.38388 If  $x$  is  $\pm 0$  and  $n$  is not  $+\text{Inf}$ ,  $x$  shall be returned.

38389 If  $x$  is  $\pm 0$  and  $n$  is  $+\text{Inf}$ , a domain error shall occur, and either a NaN (if supported), or an  
 38390 implementation-defined value shall be returned.

38391 If  $x$  is  $\pm\text{Inf}$  and  $n$  is  $-\text{Inf}$ , a domain error shall occur, and either a NaN (if supported), or an  
 38392 implementation-defined value shall be returned.

38393 If the result would cause an overflow, a range error shall occur and  $\pm\text{HUGE\_VAL}$  (according to  
 38394 the sign of  $x$ ) shall be returned.

38395 If the correct value would cause underflow, and is representable, a range error may occur and  
 38396 the correct value shall be returned.

38397 If the correct value would cause underflow, and is not representable, a range error may occur,  
 38398 and 0.0 shall be returned.

## 38399 ERRORS

38400 The *scalb()* function shall fail if:38401 Domain Error If  $x$  is zero and  $n$  is  $+\text{Inf}$ , or  $x$  is  $\text{Inf}$  and  $n$  is  $-\text{Inf}$ .

38402 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero, |  
 38403 then *errno* shall be set to [EDOM]. If the integer expression (math\_errhandling |  
 38404 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception |  
 38405 shall be raised. |

38406 Range Error The result would overflow.

38407 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero, |  
 38408 then *errno* shall be set to [ERANGE]. If the integer expression |  
 38409 (math\_errhandling & MATH\_ERREXCEPT) is non-zero, then the overflow |  
 38410 floating-point exception shall be raised. |

38411 The *scalb()* function may fail if:

38412 Range Error The result underflows.

38413 If the integer expression (*math\_errhandling* & *MATH\_ERRNO*) is non-zero, |  
 38414 then *errno* shall be set to [*ERANGE*]. If the integer expression |  
 38415 (*math\_errhandling* & *MATH\_ERREXCEPT*) is non-zero, then the underflow |  
 38416 floating-point exception shall be raised. |

38417 **EXAMPLES**

38418 None.

38419 **APPLICATION USAGE**

38420 Applications should use either *scalbln()*, *scalblnf()*, or *scalblnl()* in preference to this function.

38421 IEEE Std 1003.1-200x only defines the behavior for the *scalb()* function when the *n* argument is  
 38422 an integer, a NaN, or Inf. The behavior of other values for the *n* argument is unspecified.

38423 On error, the expressions (*math\_errhandling* & *MATH\_ERRNO*) and (*math\_errhandling* &  
 38424 *MATH\_ERREXCEPT*) are independent of each other, but at least one of them must be non-zero.

38425 **RATIONALE**

38426 None.

38427 **FUTURE DIRECTIONS**

38428 None.

38429 **SEE ALSO**

38430 *feclearexcept()*, *fetestexcept()*, *ilogb()*, *ldexp()*, *logb()*, *scalbln()*, the Base Definitions volume of |  
 38431 IEEE Std 1003.1-200x, Section 4.18, Treatment of Error Conditions for Mathematical Functions, |  
 38432 <*float.h*>, <*math.h*>

38433 **CHANGE HISTORY**

38434 First released in Issue 4, Version 2.

38435 **Issue 5**

38436 Moved from X/OPEN UNIX extension to BASE.

38437 The DESCRIPTION is updated to indicate how an application should check for an error. This  
 38438 text was previously published in the APPLICATION USAGE section.

38439 **Issue 6**

38440 This function is marked obsolescent.

38441 Although this function is not part of the ISO/IEC 9899:1999 standard, the RETURN VALUE and  
 38442 ERROR sections are updated to align with the error handling in ISO/IEC 9899:1999 standard.

## 38443 NAME

38444 scalbn, scalblnf, scalblnl, scalbn, scalbnf, scalbnl, — compute exponent using FLT\_RADIX

## 38445 SYNOPSIS

38446 #include <math.h>

```
38447 double scalbn(double x, long n);
38448 float scalblnf(float x, long n);
38449 long double scalblnl(long double x, long n);
38450 double scalbn(double x, int n);
38451 float scalbnf(float x, int n);
38452 long double scalbnl(long double x, int n);
```

## 38453 DESCRIPTION

38454 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
38455 conflict between the requirements described here and the ISO C standard is unintentional. This  
38456 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

38457 These functions shall compute  $x * FLT\_RADIX^n$  efficiently, not normally by computing  
38458  $FLT\_RADIX^n$  explicitly.

38459 An application wishing to check for error situations should set *errno* to zero and call  
38460 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
38461 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
38462 zero, an error has occurred.

## 38463 RETURN VALUE

38464 Upon successful completion, these functions shall return  $x * FLT\_RADIX^n$ .

38465 If the result would cause overflow, a range error shall occur and these functions shall return  
38466  $\pm HUGUE\_VAL$ ,  $\pm HUGUE\_VALF$ , and  $\pm HUGUE\_VALL$  (according to the sign of *x*) as appropriate for  
38467 the return type of the function.

38468 If the correct value would cause underflow, and is not representable, a range error may occur,  
38469 MX and either 0.0 (if supported), or an implementation-defined value shall be returned.

38470 MX If *x* is NaN, a NaN shall be returned.

38471 If *x* is  $\pm 0$ , or  $\pm Inf$ , *x* shall be returned.

38472 If *n* is 0, *x* shall be returned.

38473 If the correct value would cause underflow, and is representable, a range error may occur and  
38474 the correct value shall be returned.

## 38475 ERRORS

38476 These functions shall fail if:

38477 Range Error The result overflows.

38478 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero, |  
38479 then *errno* shall be set to [ERANGE]. If the integer expression |  
38480 (math\_errhandling & MATH\_ERREXCEPT) is non-zero, then the overflow |  
38481 floating-point exception shall be raised. |

38482 These functions may fail if:

38483 Range Error The result underflows.

38484 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero, |  
38485 then *errno* shall be set to [ERANGE]. If the integer expression |

38486 (math\_errhandling & MATH\_ERREXCEPT) is non-zero, then the underflow |  
38487 floating-point exception shall be raised. |

38488 **EXAMPLES**

38489 None.

38490 **APPLICATION USAGE**

38491 On error, the expressions (math\_errhandling & MATH\_ERRNO) and (math\_errhandling &  
38492 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

38493 **RATIONALE**

38494 These functions are named so as to avoid conflicting with the historical definition of the *scalb()*  
38495 function from the Single UNIX Specification. The difference is that the *scalb()* function has  
38496 second argument of **double** instead of **int**. The *scalb()* function is not part of ISO C standard. |  
38497 The three functions whose second type is **long** are provided because the factor required to scale |  
38498 from the smallest positive floating-point value to the largest finite one, on many |  
38499 implementations, is too large to represent in the minimum-width **int** format.

38500 **FUTURE DIRECTIONS**

38501 None.

38502 **SEE ALSO**

38503 *feclearexcept()*, *fetestexcept()*, *scalb()*, the Base Definitions volume of IEEE Std 1003.1-200x, Section |  
38504 4.18, Treatment of Error Conditions for Mathematical Functions, <math.h> |

38505 **CHANGE HISTORY**

38506 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

38507 **NAME**

38508           scalbn, scalbnf, scalbnl, — compute exponent using FLT\_RADIX

38509 **SYNOPSIS**

38510           #include <math.h>

38511           double scalbn(double *x*, int *n*);

38512           float scalbnf(float *x*, int *n*);

38513           long double scalbnl(long double *x*, int *n*);

38514 **DESCRIPTION**

38515           Refer to *scalbln()*.

38516 **NAME**

38517       scanf — convert formatted input

38518 **SYNOPSIS**

38519       #include &lt;stdio.h&gt;

38520       int scanf(const char \*restrict *format*, ... );38521 **DESCRIPTION**38522       Refer to *fscanf()*.

38523 **NAME**

38524 sched\_get\_priority\_max, sched\_get\_priority\_min — get priority limits (**REALTIME**)

38525 **SYNOPSIS**

```
38526 PS #include <sched.h>
```

```
38527 int sched_get_priority_max(int policy);
```

```
38528 int sched_get_priority_min(int policy);
```

```
38529
```

38530 **DESCRIPTION**

38531 The *sched\_get\_priority\_max()* and *sched\_get\_priority\_min()* functions shall return the appropriate  
38532 maximum or minimum, respectively, for the scheduling policy specified by *policy*.

38533 The value of *policy* shall be one of the scheduling policy values defined in **<sched.h>**.

38534 **RETURN VALUE**

38535 If successful, the *sched\_get\_priority\_max()* and *sched\_get\_priority\_min()* functions shall return the  
38536 appropriate maximum or minimum values, respectively. If unsuccessful, they shall return a  
38537 value of  $-1$  and set *errno* to indicate the error.

38538 **ERRORS**

38539 The *sched\_get\_priority\_max()* and *sched\_get\_priority\_min()* functions shall fail if:

38540 [EINVAL] The value of the *policy* parameter does not represent a defined scheduling  
38541 policy.

38542 **EXAMPLES**

38543 None.

38544 **APPLICATION USAGE**

38545 None.

38546 **RATIONALE**

38547 None.

38548 **FUTURE DIRECTIONS**

38549 None.

38550 **SEE ALSO**

38551 *sched\_getparam()*, *sched\_setparam()*, *sched\_getscheduler()*, *sched\_rr\_get\_interval()*,  
38552 *sched\_setscheduler()*, the Base Definitions volume of IEEE Std 1003.1-200x, **<sched.h>**

38553 **CHANGE HISTORY**

38554 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

38555 **Issue 6**

38556 These functions are marked as part of the Process Scheduling option.

38557 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
38558 implementation does not support the Process Scheduling option.

38559 The [ESRCH] error condition has been removed since these functions do not take a *pid*  
38560 argument.

38561 **NAME**38562 sched\_getparam — get scheduling parameters (**REALTIME**)38563 **SYNOPSIS**

38564 PS #include &lt;sched.h&gt;

38565 int sched\_getparam(pid\_t pid, struct sched\_param \*param);

38566

38567 **DESCRIPTION**38568 The *sched\_getparam()* function shall return the scheduling parameters of a process specified by  
38569 *pid* in the **sched\_param** structure pointed to by *param*.38570 If a process specified by *pid* exists, and if the calling process has permission, the scheduling  
38571 parameters for the process whose process ID is equal to *pid* shall be returned.38572 If *pid* is zero, the scheduling parameters for the calling process shall be returned. The behavior of  
38573 the *sched\_getparam()* function is unspecified if the value of *pid* is negative.38574 **RETURN VALUE**38575 Upon successful completion, the *sched\_getparam()* function shall return zero. If the call to  
38576 *sched\_getparam()* is unsuccessful, the function shall return a value of -1 and set *errno* to indicate  
38577 the error.38578 **ERRORS**38579 The *sched\_getparam()* function shall fail if:38580 [EPERM] The requesting process does not have permission to obtain the scheduling  
38581 parameters of the specified process.38582 [ESRCH] No process can be found corresponding to that specified by *pid*.38583 **EXAMPLES**

38584 None.

38585 **APPLICATION USAGE**

38586 None.

38587 **RATIONALE**

38588 None.

38589 **FUTURE DIRECTIONS**

38590 None.

38591 **SEE ALSO**38592 *sched\_getscheduler()*, *sched\_setparam()*, *sched\_setscheduler()*, the Base Definitions volume of  
38593 IEEE Std 1003.1-200x, <sched.h>38594 **CHANGE HISTORY**

38595 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

38596 **Issue 6**38597 The *sched\_getparam()* function is marked as part of the Process Scheduling option.38598 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
38599 implementation does not support the Process Scheduling option.

38600 **NAME**

38601 sched\_getscheduler — get scheduling policy (**REALTIME**)

38602 **SYNOPSIS**

38603 PS #include <sched.h>

38604 int sched\_getscheduler(pid\_t pid);

38605

38606 **DESCRIPTION**

38607 The *sched\_getscheduler()* function shall return the scheduling policy of the process specified by  
 38608 *pid*. If the value of *pid* is negative, the behavior of the *sched\_getscheduler()* function is  
 38609 unspecified.

38610 The values that can be returned by *sched\_getscheduler()* are defined in the <**sched.h**> header.

38611 If a process specified by *pid* exists, and if the calling process has permission, the scheduling  
 38612 policy shall be returned for the process whose process ID is equal to *pid*.

38613 If *pid* is zero, the scheduling policy shall be returned for the calling process.

38614 **RETURN VALUE**

38615 Upon successful completion, the *sched\_getscheduler()* function shall return the scheduling policy  
 38616 of the specified process. If unsuccessful, the function shall return  $-1$  and set *errno* to indicate the  
 38617 error.

38618 **ERRORS**

38619 The *sched\_getscheduler()* function shall fail if:

38620 [EPERM] The requesting process does not have permission to determine the scheduling  
 38621 policy of the specified process.

38622 [ESRCH] No process can be found corresponding to that specified by *pid*.

38623 **EXAMPLES**

38624 None.

38625 **APPLICATION USAGE**

38626 None.

38627 **RATIONALE**

38628 None.

38629 **FUTURE DIRECTIONS**

38630 None.

38631 **SEE ALSO**

38632 *sched\_getparam()*, *sched\_setparam()*, *sched\_setscheduler()*, the Base Definitions volume of  
 38633 IEEE Std 1003.1-200x, <**sched.h**>

38634 **CHANGE HISTORY**

38635 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

38636 **Issue 6**

38637 The *sched\_getscheduler()* function is marked as part of the Process Scheduling option.

38638 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
 38639 implementation does not support the Process Scheduling option.

38640 **NAME**

38641 sched\_rr\_get\_interval — get execution time limits (**REALTIME**)

38642 **SYNOPSIS**

```
38643 PS #include <sched.h>
```

```
38644 int sched_rr_get_interval(pid_t pid, struct timespec *interval);
```

38645

38646 **DESCRIPTION**

38647 The *sched\_rr\_get\_interval()* function shall update the **timespec** structure referenced by the  
38648 *interval* argument to contain the current execution time limit (that is, time quantum) for the  
38649 process specified by *pid*. If *pid* is zero, the current execution time limit for the calling process  
38650 shall be returned.

38651 **RETURN VALUE**

38652 If successful, the *sched\_rr\_get\_interval()* function shall return zero. Otherwise, it shall return a  
38653 value of  $-1$  and set *errno* to indicate the error.

38654 **ERRORS**

38655 The *sched\_rr\_get\_interval()* function shall fail if:

38656 [ESRCH] No process can be found corresponding to that specified by *pid*.

38657 **EXAMPLES**

38658 None.

38659 **APPLICATION USAGE**

38660 None.

38661 **RATIONALE**

38662 None.

38663 **FUTURE DIRECTIONS**

38664 None.

38665 **SEE ALSO**

38666 *sched\_getparam()*, *sched\_get\_priority\_max()*, *sched\_getscheduler()*, *sched\_setparam()*,  
38667 *sched\_setscheduler()*, the Base Definitions volume of IEEE Std 1003.1-200x, <**sched.h**>

38668 **CHANGE HISTORY**

38669 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

38670 **Issue 6**

38671 The *sched\_rr\_get\_interval()* function is marked as part of the Process Scheduling option.

38672 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
38673 implementation does not support the Process Scheduling option.

38674 **NAME**

38675 sched\_setparam — set scheduling parameters (**REALTIME**)

38676 **SYNOPSIS**

38677 PS #include <sched.h>

38678 int sched\_setparam(pid\_t pid, const struct sched\_param \*param);

38679

38680 **DESCRIPTION**

38681 The *sched\_setparam()* function shall set the scheduling parameters of the process specified by *pid*  
 38682 to the values specified by the **sched\_param** structure pointed to by *param*. The value of the  
 38683 *sched\_priority* member in the **sched\_param** structure shall be any integer within the inclusive  
 38684 priority range for the current scheduling policy of the process specified by *pid*. Higher  
 38685 numerical values for the priority represent higher priorities. If the value of *pid* is negative, the  
 38686 behavior of the *sched\_setparam()* function is unspecified.

38687 If a process specified by *pid* exists, and if the calling process has permission, the scheduling  
 38688 parameters shall be set for the process whose process ID is equal to *pid*.

38689 If *pid* is zero, the scheduling parameters shall be set for the calling process.

38690 The conditions under which one process has permission to change the scheduling parameters of  
 38691 another process are implementation-defined.

38692 Implementations may require the requesting process to have the appropriate privilege to set its  
 38693 own scheduling parameters or those of another process.

38694 The target process, whether it is running or not running, shall be moved to the tail of the thread  
 38695 list for its priority.

38696 If the priority of the process specified by the *pid* argument is set higher than that of the lowest  
 38697 priority running process and if the specified process is ready to run, the process specified by the  
 38698 *pid* argument shall preempt a lowest priority running process. Similarly, if the process calling  
 38699 *sched\_setparam()* sets its own priority lower than that of one or more other non-empty process  
 38700 lists, then the process that is the head of the highest priority list shall also preempt the calling  
 38701 process. Thus, in either case, the originating process might not receive notification of the  
 38702 completion of the requested priority change until the higher priority process has executed.

38703 ss If the scheduling policy of the target process is SCHED\_SPORADIC, the value specified by the  
 38704 *sched\_ss\_low\_priority* member of the *param* argument shall be any integer within the inclusive  
 38705 priority range for the sporadic server policy. The *sched\_ss\_repl\_period* and *sched\_ss\_init\_budget*  
 38706 members of the *param* argument shall represent the time parameters to be used by the sporadic  
 38707 server scheduling policy for the target process. The *sched\_ss\_max\_repl* member of the *param*  
 38708 argument shall represent the maximum number of replenishments that are allowed to be  
 38709 pending simultaneously for the process scheduled under this scheduling policy.

38710 The specified *sched\_ss\_repl\_period* shall be greater than or equal to the specified  
 38711 *sched\_ss\_init\_budget* for the function to succeed; if it is not, then the function shall fail.

38712 The value of *sched\_ss\_max\_repl* shall be within the inclusive range [1,{SS\_REPL\_MAX}] for the  
 38713 function to succeed; if not, the function shall fail.

38714 If the scheduling policy of the target process is either SCHED\_FIFO or SCHED\_RR, the  
 38715 *sched\_ss\_low\_priority*, *sched\_ss\_repl\_period*, and *sched\_ss\_init\_budget* members of the *param*  
 38716 argument shall have no effect on the scheduling behavior. If the scheduling policy of this process  
 38717 is not SCHED\_FIFO, SCHED\_RR, or SCHED\_SPORADIC, the effects of these members are  
 38718 implementation-defined; this case includes the SCHED\_OTHER policy.

38719 If the current scheduling policy for the process specified by *pid* is not SCHED\_FIFO,  
 38720 ss SCHED\_RR, or SCHED\_SPORADIC, the result is implementation-defined; this case includes the  
 38721 SCHED\_OTHER policy.

38722 The effect of this function on individual threads is dependent on the scheduling contention  
 38723 scope of the threads:

- 38724 • For threads with system scheduling contention scope, these functions shall have no effect on  
 38725 their scheduling.

- 38726 • For threads with process scheduling contention scope, the threads' scheduling parameters  
 38727 shall not be affected. However, the scheduling of these threads with respect to threads in  
 38728 other processes may be dependent on the scheduling parameters of their process, which are  
 38729 governed using these functions.

38730 If an implementation supports a two-level scheduling model in which library threads are  
 38731 multiplexed on top of several kernel-scheduled entities, then the underlying kernel-scheduled  
 38732 entities for the system contention scope threads shall not be affected by these functions.

38733 The underlying kernel-scheduled entities for the process contention scope threads shall have  
 38734 their scheduling parameters changed to the value specified in *param*. Kernel scheduled entities  
 38735 for use by process contention scope threads that are created after this call completes shall inherit  
 38736 their scheduling policy and associated scheduling parameters from the process.

38737 This function is not atomic with respect to other threads in the process. Threads may continue to  
 38738 execute while this function call is in the process of changing the scheduling policy for the  
 38739 underlying kernel-scheduled entities used by the process contention scope threads.

#### 38740 RETURN VALUE

38741 If successful, the *sched\_setparam()* function shall return zero.

38742 If the call to *sched\_setparam()* is unsuccessful, the priority shall remain unchanged, and the  
 38743 function shall return a value of -1 and set *errno* to indicate the error.

#### 38744 ERRORS

38745 The *sched\_setparam()* function shall fail if:

38746 [EINVAL] One or more of the requested scheduling parameters is outside the range  
 38747 defined for the scheduling policy of the specified *pid*.

38748 [EPERM] The requesting process does not have permission to set the scheduling  
 38749 parameters for the specified process, or does not have the appropriate  
 38750 privilege to invoke *sched\_setparam()*.

38751 [ESRCH] No process can be found corresponding to that specified by *pid*.

#### 38752 EXAMPLES

38753 None.

#### 38754 APPLICATION USAGE

38755 None.

#### 38756 RATIONALE

38757 None.

#### 38758 FUTURE DIRECTIONS

38759 None.

38760 **SEE ALSO**

38761 *sched\_getparam()*, *sched\_getscheduler()*, *sched\_setscheduler()*, the Base Definitions volume of  
38762 IEEE Std 1003.1-200x, <**sched.h**>

38763 **CHANGE HISTORY**

38764 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

38765 **Issue 6**

38766 The *sched\_setparam()* function is marked as part of the Process Scheduling option.

38767 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
38768 implementation does not support the Process Scheduling option.

38769 The following new requirements on POSIX implementations derive from alignment with the  
38770 Single UNIX Specification:

38771 • In the DESCRIPTION, the effect of this function on a thread's scheduling parameters is  
38772 added.

38773 • Sections describing two-level scheduling and atomicity of the function are added.

38774 The SCHED\_SPORADIC scheduling policy is added for alignment with IEEE Std 1003.1d-1999.

38775 IEEE PASC Interpretation 1003.1 #100 is applied.

## 38776 NAME

38777 sched\_setscheduler — set scheduling policy and parameters (**REALTIME**)

## 38778 SYNOPSIS

38779 PS `#include <sched.h>`38780 `int sched_setscheduler(pid_t pid, int policy,`  
38781 `const struct sched_param *param);`

38782

## 38783 DESCRIPTION

38784 The `sched_setscheduler()` function shall set the scheduling policy and scheduling parameters of  
 38785 the process specified by `pid` to `policy` and the parameters specified in the `sched_param` structure  
 38786 pointed to by `param`, respectively. The value of the `sched_priority` member in the `sched_param`  
 38787 structure shall be any integer within the inclusive priority range for the scheduling policy  
 38788 specified by `policy`. If the value of `pid` is negative, the behavior of the `sched_setscheduler()`  
 38789 function is unspecified.

38790 The possible values for the `policy` parameter are defined in the `<sched.h>` header. |

38791 If a process specified by `pid` exists, and if the calling process has permission, the scheduling  
 38792 policy and scheduling parameters shall be set for the process whose process ID is equal to `pid`.

38793 If `pid` is zero, the scheduling policy and scheduling parameters shall be set for the calling  
 38794 process.

38795 The conditions under which one process has the appropriate privilege to change the scheduling  
 38796 parameters of another process are implementation-defined.

38797 Implementations may require that the requesting process have permission to set its own  
 38798 scheduling parameters or those of another process. Additionally, implementation-defined  
 38799 restrictions may apply as to the appropriate privileges required to set a process' own scheduling  
 38800 policy, or another process' scheduling policy, to a particular value.

38801 The `sched_setscheduler()` function shall be considered successful if it succeeds in setting the |  
 38802 scheduling policy and scheduling parameters of the process specified by `pid` to the values  
 38803 specified by `policy` and the structure pointed to by `param`, respectively.

38804 ss If the scheduling policy specified by `policy` is `SCHED_SPORADIC`, the value specified by the  
 38805 `sched_ss_low_priority` member of the `param` argument shall be any integer within the inclusive  
 38806 priority range for the sporadic server policy. The `sched_ss_repl_period` and `sched_ss_init_budget`  
 38807 members of the `param` argument shall represent the time parameters used by the sporadic server  
 38808 scheduling policy for the target process. The `sched_ss_max_repl` member of the `param` argument  
 38809 shall represent the maximum number of replenishments that are allowed to be pending  
 38810 simultaneously for the process scheduled under this scheduling policy.

38811 The specified `sched_ss_repl_period` shall be greater than or equal to the specified  
 38812 `sched_ss_init_budget` for the function to succeed; if it is not, then the function shall fail.

38813 The value of `sched_ss_max_repl` shall be within the inclusive range `[1, {SS_REPL_MAX}]` for the  
 38814 function to succeed; if not, the function shall fail.

38815 If the scheduling policy specified by `policy` is either `SCHED_FIFO` or `SCHED_RR`, the  
 38816 `sched_ss_low_priority`, `sched_ss_repl_period`, and `sched_ss_init_budget` members of the `param`  
 38817 argument shall have no effect on the scheduling behavior.

38818 The effect of this function on individual threads is dependent on the scheduling contention  
 38819 scope of the threads:

38820           • For threads with system scheduling contention scope, these functions shall have no effect on  
38821           their scheduling.

38822           • For threads with process scheduling contention scope, the threads' scheduling policy and  
38823           associated parameters shall not be affected. However, the scheduling of these threads with  
38824           respect to threads in other processes may be dependent on the scheduling parameters of their  
38825           process, which are governed using these functions.

38826           If an implementation supports a two-level scheduling model in which library threads are  
38827           multiplexed on top of several kernel-scheduled entities, then the underlying kernel-scheduled  
38828           entities for the system contention scope threads shall not be affected by these functions.

38829           The underlying kernel-scheduled entities for the process contention scope threads shall have  
38830           their scheduling policy and associated scheduling parameters changed to the values specified in  
38831           *policy* and *param*, respectively. Kernel scheduled entities for use by process contention scope  
38832           threads that are created after this call completes shall inherit their scheduling policy and  
38833           associated scheduling parameters from the process.

38834           This function is not atomic with respect to other threads in the process. Threads may continue to  
38835           execute while this function call is in the process of changing the scheduling policy and  
38836           associated scheduling parameters for the underlying kernel-scheduled entities used by the  
38837           process contention scope threads.

38838 **RETURN VALUE**

38839           Upon successful completion, the function shall return the former scheduling policy of the  
38840           specified process. If the *sched\_setscheduler()* function fails to complete successfully, the policy  
38841           and scheduling parameters shall remain unchanged, and the function shall return a value of -1  
38842           and set *errno* to indicate the error.

38843 **ERRORS**

38844           The *sched\_setscheduler()* function shall fail if:

38845           [EINVAL]           The value of the *policy* parameter is invalid, or one or more of the parameters  
38846           contained in *param* is outside the valid range for the specified scheduling  
38847           policy.

38848           [EPERM]           The requesting process does not have permission to set either or both of the  
38849           scheduling parameters or the scheduling policy of the specified process.

38850           [ESRCH]           No process can be found corresponding to that specified by *pid*.

38851 **EXAMPLES**

38852           None.

38853 **APPLICATION USAGE**

38854           None.

38855 **RATIONALE**

38856           None.

38857 **FUTURE DIRECTIONS**

38858           None.

38859 **SEE ALSO**

38860           *sched\_getparam()*, *sched\_getscheduler()*, *sched\_setparam()*, the Base Definitions volume of  
38861           IEEE Std 1003.1-200x, <*sched.h*>

38862 **CHANGE HISTORY**

38863 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

38864 **Issue 6**

38865 The *sched\_setscheduler()* function is marked as part of the Process Scheduling option.

38866 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
38867 implementation does not support the Process Scheduling option.

38868 The following new requirements on POSIX implementations derive from alignment with the  
38869 Single UNIX Specification:

38870 • In the DESCRIPTION, the effect of this function on a thread's scheduling parameters is  
38871 added.

38872 • Sections describing two-level scheduling and atomicity of the function are added.

38873 The SCHED\_SPORADIC scheduling policy is added for alignment with IEEE Std 1003.1d-1999.

38874 **NAME**

38875 sched\_yield — yield processor

38876 **SYNOPSIS**

38877 PS|THR #include &lt;sched.h&gt;

38878 int sched\_yield(void);

38879

38880 **DESCRIPTION**38881 The *sched\_yield()* function shall force the running thread to relinquish the processor until it again  
38882 becomes the head of its thread list. It takes no arguments.38883 **RETURN VALUE**38884 The *sched\_yield()* function shall return 0 if it completes successfully; otherwise, it shall return a  
38885 value of -1 and set *errno* to indicate the error.38886 **ERRORS**

38887 No errors are defined.

38888 **EXAMPLES**

38889 None.

38890 **APPLICATION USAGE**

38891 None.

38892 **RATIONALE**

38893 None.

38894 **FUTURE DIRECTIONS**

38895 None.

38896 **SEE ALSO**

38897 The Base Definitions volume of IEEE Std 1003.1-200x, &lt;sched.h&gt;

38898 **CHANGE HISTORY**38899 First released in Issue 5. Included for alignment with the POSIX Realtime Extension and the  
38900 POSIX Threads Extension.38901 **Issue 6**38902 The *sched\_yield()* function is now marked as part of the Process Scheduling and Threads options.

38903 **NAME**

38904           seed48 — seed uniformly distributed pseudo-random non-negative long integer generator

38905 **SYNOPSIS**

38906 xSI       #include <stdlib.h>

38907           unsigned short \*seed48(unsigned short seed16v[3]);

38908

38909 **DESCRIPTION**

38910           Refer to *drand48()*.

38911 **NAME**

38912 seekdir — set position of directory stream

38913 **SYNOPSIS**

38914 XSI #include &lt;dirent.h&gt;

38915 void seekdir(DIR \*dirp, long loc);

38916

38917 **DESCRIPTION**

38918 The *seekdir()* function shall set the position of the next *readdir()* operation on the directory  
38919 stream specified by *dirp* to the position specified by *loc*. The value of *loc* should have been  
38920 returned from an earlier call to *telldir()*. The new position reverts to the one associated with the  
38921 directory stream when *telldir()* was performed.

38922 If the value of *loc* was not obtained from an earlier call to *telldir()*, or if a call to *rewinddir()*  
38923 occurred between the call to *telldir()* and the call to *seekdir()*, the results of subsequent calls to  
38924 *readdir()* are unspecified.

38925 **RETURN VALUE**38926 The *seekdir()* function shall not return a value.38927 **ERRORS**

38928 No errors are defined.

38929 **EXAMPLES**

38930 None.

38931 **APPLICATION USAGE**

38932 None.

38933 **RATIONALE**

38934 The original standard developers perceived that there were restrictions on the use of the  
38935 *seekdir()* and *telldir()* functions related to implementation details, and for that reason these  
38936 functions need not be supported on all POSIX-conforming systems. They are required on  
38937 implementations supporting the XSI extension.

38938 One of the perceived problems of implementation is that returning to a given point in a directory  
38939 is quite difficult to describe formally, in spite of its intuitive appeal, when systems that use B-  
38940 trees, hashing functions, or other similar mechanisms to order their directories are considered.  
38941 The definition of *seekdir()* and *telldir()* does not specify whether, when using these interfaces, a  
38942 given directory entry will be seen at all, or more than once.

38943 On systems not supporting these functions, their capability can sometimes be accomplished by  
38944 saving a filename found by *readdir()* and later using *rewinddir()* and a loop on *readdir()* to  
38945 relocate the position from which the filename was saved.

38946 **FUTURE DIRECTIONS**

38947 None.

38948 **SEE ALSO**

38949 *opendir()*, *readdir()*, *telldir()*, the Base Definitions volume of IEEE Std 1003.1-200x, <dirent.h>,  
38950 <stdio.h>, <sys/types.h>

38951 **CHANGE HISTORY**

38952 First released in Issue 2.

38953 **Issue 6**

38954 In the SYNOPSIS, the inclusion of `<sys/types.h>` is no longer required.

38955 **NAME**

38956           select — synchronous I/O multiplexing

38957 **SYNOPSIS**

38958           #include <sys/time.h>

```
38959           int select(int nfds, fd_set *restrict readfds,  
38960                      fd_set *restrict writefds, fd_set *restrict errorfds,  
38961                      struct timeval *restrict timeout);
```

38962

38963 **DESCRIPTION**

38964           Refer to *pselect()*.

38965 **NAME**38966 sem\_close — close a named semaphore (**REALTIME**)38967 **SYNOPSIS**

38968 SEM #include &lt;semaphore.h&gt;

38969 int sem\_close(sem\_t \*sem);

38970

38971 **DESCRIPTION**

38972 The `sem_close()` function shall indicate that the calling process is finished using the named  
 38973 semaphore indicated by `sem`. The effects of calling `sem_close()` for an unnamed semaphore (one  
 38974 created by `sem_init()`) are undefined. The `sem_close()` function shall deallocate (that is, make  
 38975 available for reuse by a subsequent `sem_open()` by this process) any system resources allocated  
 38976 by the system for use by this process for this semaphore. The effect of subsequent use of the  
 38977 semaphore indicated by `sem` by this process is undefined. If the semaphore has not been  
 38978 removed with a successful call to `sem_unlink()`, then `sem_close()` has no effect on the state of the  
 38979 semaphore. If the `sem_unlink()` function has been successfully invoked for `name` after the most  
 38980 recent call to `sem_open()` with `O_CREAT` for this semaphore, then when all processes that have  
 38981 opened the semaphore close it, the semaphore is no longer accessible.

38982 **RETURN VALUE**

38983 Upon successful completion, a value of zero shall be returned. Otherwise, a value of `-1` shall be  
 38984 returned and `errno` set to indicate the error.

38985 **ERRORS**38986 The `sem_close()` function shall fail if:38987 [EINVAL] The `sem` argument is not a valid semaphore descriptor.38988 **EXAMPLES**

38989 None.

38990 **APPLICATION USAGE**

38991 The `sem_close()` function is part of the Semaphores option and need not be available on all  
 38992 implementations.

38993 **RATIONALE**

38994 None.

38995 **FUTURE DIRECTIONS**

38996 None.

38997 **SEE ALSO**

38998 `semctl()`, `semget()`, `semop()`, `sem_init()`, `sem_open()`, `sem_unlink()`, the Base Definitions volume of  
 38999 IEEE Std 1003.1-200x, <**semaphore.h**>

39000 **CHANGE HISTORY**

39001 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

39002 **Issue 6**39003 The `sem_close()` function is marked as part of the Semaphores option.

39004 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
 39005 implementation does not support the Semaphores option.

39006 **NAME**39007 sem\_destroy — destroy an unnamed semaphore (**REALTIME**)39008 **SYNOPSIS**

39009 SEM #include &lt;semaphore.h&gt;

39010 int sem\_destroy(sem\_t \*sem);

39011

39012 **DESCRIPTION**

39013 The *sem\_destroy()* function shall destroy the unnamed semaphore indicated by *sem*. Only a  
39014 semaphore that was created using *sem\_init()* may be destroyed using *sem\_destroy()*; the effect of  
39015 calling *sem\_destroy()* with a named semaphore is undefined. The effect of subsequent use of the  
39016 semaphore *sem* is undefined until *sem* is reinitialized by another call to *sem\_init()*.

39017 It is safe to destroy an initialized semaphore upon which no threads are currently blocked. The  
39018 effect of destroying a semaphore upon which other threads are currently blocked is undefined.

39019 **RETURN VALUE**

39020 Upon successful completion, a value of zero shall be returned. Otherwise, a value of -1 shall be  
39021 returned and *errno* set to indicate the error.

39022 **ERRORS**39023 The *sem\_destroy()* function shall fail if:39024 [EINVAL] The *sem* argument is not a valid semaphore.39025 The *sem\_destroy()* function may fail if:

39026 [EBUSY] There are currently processes blocked on the semaphore.

39027 **EXAMPLES**

39028 None.

39029 **APPLICATION USAGE**

39030 The *sem\_destroy()* function is part of the Semaphores option and need not be available on all  
39031 implementations.

39032 **RATIONALE**

39033 None.

39034 **FUTURE DIRECTIONS**

39035 None.

39036 **SEE ALSO**

39037 *semctl()*, *semget()*, *semop()*, *sem\_init()*, *sem\_open()*, the Base Definitions volume of  
39038 IEEE Std 1003.1-200x, <semaphore.h>

39039 **CHANGE HISTORY**

39040 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

39041 **Issue 6**39042 The *sem\_destroy()* function is marked as part of the Semaphores option.

39043 The [ENOSYS] error condition has been removed as stubs need not be provided if an

39044 implementation does not support the Semaphores option.

39045 **NAME**

39046 `sem_getvalue` — get the value of a semaphore (**REALTIME**)

39047 **SYNOPSIS**

```
39048 SEM #include <semaphore.h>
```

```
39049 int sem_getvalue(sem_t *restrict sem, int *restrict sval);
```

39050

39051 **DESCRIPTION**

39052 The `sem_getvalue()` function shall update the location referenced by the `sval` argument to have  
39053 the value of the semaphore referenced by `sem` without affecting the state of the semaphore. The  
39054 updated value represents an actual semaphore value that occurred at some unspecified time  
39055 during the call, but it need not be the actual value of the semaphore when it is returned to the  
39056 calling process.

39057 If `sem` is locked, then the value returned by `sem_getvalue()` is either zero or a negative number  
39058 whose absolute value represents the number of processes waiting for the semaphore at some  
39059 unspecified time during the call.

39060 **RETURN VALUE**

39061 Upon successful completion, the `sem_getvalue()` function shall return a value of zero. Otherwise,  
39062 it shall return a value of `-1` and set `errno` to indicate the error.

39063 **ERRORS**

39064 The `sem_getvalue()` function shall fail if:

39065 [EINVAL] The `sem` argument does not refer to a valid semaphore.

39066 **EXAMPLES**

39067 None.

39068 **APPLICATION USAGE**

39069 The `sem_getvalue()` function is part of the Semaphores option and need not be available on all  
39070 implementations.

39071 **RATIONALE**

39072 None.

39073 **FUTURE DIRECTIONS**

39074 None.

39075 **SEE ALSO**

39076 `semctl()`, `semget()`, `semop()`, `sem_post()`, `sem_timedwait()`, `sem_trywait()`, `sem_wait()`, the Base  
39077 Definitions volume of IEEE Std 1003.1-200x, <**semaphore.h**>

39078 **CHANGE HISTORY**

39079 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

39080 **Issue 6**

39081 The `sem_getvalue()` function is marked as part of the Semaphores option.

39082 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
39083 implementation does not support the Semaphores option.

39084 The `sem_timedwait()` function is added to the SEE ALSO section for alignment with  
39085 IEEE Std 1003.1d-1999.

39086 The **restrict** keyword is added to the `sem_getvalue()` prototype for alignment with the  
39087 ISO/IEC 9899:1999 standard.

39088 **NAME**39089 sem\_init — initialize an unnamed semaphore (**REALTIME**)39090 **SYNOPSIS**

39091 SEM #include &lt;semaphore.h&gt;

39092 int sem\_init(sem\_t \*sem, int pshared, unsigned value);

39093

39094 **DESCRIPTION**

39095 The *sem\_init()* function shall initialize the unnamed semaphore referred to by *sem*. The value of  
 39096 the initialized semaphore shall be *value*. Following a successful call to *sem\_init()*, the semaphore  
 39097 may be used in subsequent calls to *sem\_wait()*, *sem\_trywait()*, *sem\_post()*, and *sem\_destroy()*.  
 39098 This semaphore shall remain usable until the semaphore is destroyed.

39099 If the *pshared* argument has a non-zero value, then the semaphore is shared between processes;  
 39100 in this case, any process that can access the semaphore *sem* can use *sem* for performing  
 39101 *sem\_wait()*, *sem\_trywait()*, *sem\_post()*, and *sem\_destroy()* operations.

39102 Only *sem* itself may be used for performing synchronization. The result of referring to copies of  
 39103 *sem* in calls to *sem\_wait()*, *sem\_trywait()*, *sem\_post()*, and *sem\_destroy()*, is undefined.

39104 If the *pshared* argument is zero, then the semaphore is shared between threads of the process; any  
 39105 thread in this process can use *sem* for performing *sem\_wait()*, *sem\_trywait()*, *sem\_post()*, and  
 39106 *sem\_destroy()* operations. The use of the semaphore by threads other than those created in the  
 39107 same process is undefined.

39108 Attempting to initialize an already initialized semaphore results in undefined behavior.

39109 **RETURN VALUE**

39110 Upon successful completion, the *sem\_init()* function shall initialize the semaphore in *sem*.  
 39111 Otherwise, it shall return  $-1$  and set *errno* to indicate the error.

39112 **ERRORS**

39113 The *sem\_init()* function shall fail if:

39114 [EINVAL] The *value* argument exceeds {SEM\_VALUE\_MAX}.

39115 [ENOSPC] A resource required to initialize the semaphore has been exhausted, or the  
 39116 limit on semaphores ({SEM\_NSEMS\_MAX}) has been reached.

39117 [EPERM] The process lacks the appropriate privileges to initialize the semaphore.

39118 **EXAMPLES**

39119 None.

39120 **APPLICATION USAGE**

39121 The *sem\_init()* function is part of the Semaphores option and need not be available on all  
 39122 implementations.

39123 **RATIONALE**

39124 Although this volume of IEEE Std 1003.1-200x fails to specify a successful return value, it is  
 39125 likely that a later version may require the implementation to return a value of zero if the call to  
 39126 *sem\_init()* is successful.

39127 **FUTURE DIRECTIONS**

39128 None.

39129 **SEE ALSO**

39130 *sem\_destroy()*, *sem\_post()*, *sem\_timedwait()*, *sem\_trywait()*, *sem\_wait()*, the Base Definitions  
39131 volume of IEEE Std 1003.1-200x, <**semaphore.h**>

39132 **CHANGE HISTORY**

39133 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

39134 **Issue 6**

39135 The *sem\_init()* function is marked as part of the Semaphores option.

39136 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
39137 implementation does not support the Semaphores option.

39138 The *sem\_timedwait()* function is added to the SEE ALSO section for alignment with  
39139 IEEE Std 1003.1d-1999.

## 39140 NAME

39141 sem\_open — initialize and open a named semaphore (**REALTIME**)

## 39142 SYNOPSIS

39143 SEM #include &lt;semaphore.h&gt;

39144 sem\_t \*sem\_open(const char \*name, int oflag, ...);

39145

## 39146 DESCRIPTION

39147 The *sem\_open()* function shall establish a connection between a named semaphore and a process.  
 39148 Following a call to *sem\_open()* with semaphore name *name*, the process may reference the  
 39149 semaphore associated with *name* using the address returned from the call. This semaphore may  
 39150 be used in subsequent calls to *sem\_wait()*, *sem\_trywait()*, *sem\_post()*, and *sem\_close()*. The  
 39151 semaphore remains usable by this process until the semaphore is closed by a successful call to  
 39152 *sem\_close()*, *\_exit()*, or one of the *exec* functions.

39153 The *oflag* argument controls whether the semaphore is created or merely accessed by the call to  
 39154 *sem\_open()*. The following flag bits may be set in *oflag*:

39155 **O\_CREAT** This flag is used to create a semaphore if it does not already exist. If **O\_CREAT**  
 39156 is set and the semaphore already exists, then **O\_CREAT** has no effect, except as noted  
 39157 under **O\_EXCL**. Otherwise, *sem\_open()* creates a named semaphore. The **O\_CREAT**  
 39158 flag requires a third and a fourth argument: *mode*, which is of type **mode\_t**, and  
 39159 *value*, which is of type **unsigned**. The semaphore is created with an initial value of  
 39160 *value*. Valid initial values for semaphores are less than or equal to  
 39161 {SEM\_VALUE\_MAX}.

39162 The user ID of the semaphore is set to the effective user ID of the process; the  
 39163 group ID of the semaphore is set to a system default group ID or to the effective  
 39164 group ID of the process. The permission bits of the semaphore are set to the value  
 39165 of the *mode* argument except those set in the file mode creation mask of the  
 39166 process. When bits in *mode* other than the file permission bits are specified, the  
 39167 effect is unspecified.

39168 After the semaphore named *name* has been created by *sem\_open()* with the  
 39169 **O\_CREAT** flag, other processes can connect to the semaphore by calling  
 39170 *sem\_open()* with the same value of *name*.

39171 **O\_EXCL** If **O\_EXCL** and **O\_CREAT** are set, *sem\_open()* fails if the semaphore *name* exists.  
 39172 The check for the existence of the semaphore and the creation of the semaphore if  
 39173 it does not exist are atomic with respect to other processes executing *sem\_open()*  
 39174 with **O\_EXCL** and **O\_CREAT** set. If **O\_EXCL** is set and **O\_CREAT** is not set, the  
 39175 effect is undefined.

39176 If flags other than **O\_CREAT** and **O\_EXCL** are specified in the *oflag* parameter, the  
 39177 effect is unspecified.

39178 The *name* argument points to a string naming a semaphore object. It is unspecified whether the  
 39179 name appears in the file system and is visible to functions that take pathnames as arguments. |  
 39180 The *name* argument conforms to the construction rules for a pathname. If *name* begins with the |  
 39181 slash character, then processes calling *sem\_open()* with the same value of *name* shall refer to the |  
 39182 same semaphore object, as long as that name has not been removed. If *name* does not begin with |  
 39183 the slash character, the effect is implementation-defined. The interpretation of slash characters |  
 39184 other than the leading slash character in *name* is implementation-defined.

39185 If a process makes multiple successful calls to *sem\_open()* with the same value for *name*, the |  
 39186 same semaphore address shall be returned for each such successful call, provided that there |

- 39187 have been no calls to *sem\_unlink()* for this semaphore.
- 39188 References to copies of the semaphore produce undefined results.
- 39189 **RETURN VALUE**
- 39190 Upon successful completion, the *sem\_open()* function shall return the address of the semaphore.
- 39191 Otherwise, it shall return a value of SEM\_FAILED and set *errno* to indicate the error. The symbol
- 39192 SEM\_FAILED is defined in the <semaphore.h> header. No successful return from *sem\_open()*
- 39193 shall return the value SEM\_FAILED.
- 39194 **ERRORS**
- 39195 If any of the following conditions occur, the *sem\_open()* function shall return SEM\_FAILED and
- 39196 set *errno* to the corresponding value:
- 39197 [EACCES] The named semaphore exists and the permissions specified by *oflag* are
- 39198 denied, or the named semaphore does not exist and permission to create the
- 39199 named semaphore is denied.
- 39200 [EEXIST] O\_CREAT and O\_EXCL are set and the named semaphore already exists.
- 39201 [EINTR] The *sem\_open()* operation was interrupted by a signal.
- 39202 [EINVAL] The *sem\_open()* operation is not supported for the given name, or O\_CREAT
- 39203 was specified in *oflag* and *value* was greater than {SEM\_VALUE\_MAX}.
- 39204 [EMFILE] Too many semaphore descriptors or file descriptors are currently in use by
- 39205 this process.
- 39206 [ENAMETOOLONG]
- 39207 The length of the *name* argument exceeds {PATH\_MAX} or a pathname
- 39208 component is longer than {NAME\_MAX}.
- 39209 [ENFILE] Too many semaphores are currently open in the system.
- 39210 [ENOENT] O\_CREAT is not set and the named semaphore does not exist.
- 39211 [ENOSPC] There is insufficient space for the creation of the new named semaphore.
- 39212 **EXAMPLES**
- 39213 None.
- 39214 **APPLICATION USAGE**
- 39215 The *sem\_open()* function is part of the Semaphores option and need not be available on all
- 39216 implementations.
- 39217 **RATIONALE**
- 39218 An earlier version of this volume of IEEE Std 1003.1-200x required an error return value of -1
- 39219 with the type **sem\_t** \* for the *sem\_open()* function, which is not guaranteed to be portable across
- 39220 implementations. The revised text provides the symbolic error code SEM\_FAILED to eliminate
- 39221 the type conflict.
- 39222 **FUTURE DIRECTIONS**
- 39223 None.
- 39224 **SEE ALSO**
- 39225 *semctl()*, *semget()*, *semop()*, *sem\_close()*, *sem\_post()*, *sem\_timedwait()*, *sem\_trywait()*, *sem\_unlink()*,
- 39226 *sem\_wait()*, the Base Definitions volume of IEEE Std 1003.1-200x, <semaphore.h>

39227 **CHANGE HISTORY**

39228 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

39229 **Issue 6**

39230 The *sem\_open()* function is marked as part of the Semaphores option.

39231 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
39232 implementation does not support the Semaphores option.

39233 The *sem\_timedwait()* function is added to the SEE ALSO section for alignment with  
39234 IEEE Std 1003.1d-1999.

39235 **NAME**39236 sem\_post — unlock a semaphore (**REALTIME**)39237 **SYNOPSIS**

39238 SEM #include &lt;semaphore.h&gt;

39239 int sem\_post(sem\_t \*sem);

39240

39241 **DESCRIPTION**39242 The *sem\_post()* function shall unlock the semaphore referenced by *sem* by performing a  
39243 semaphore unlock operation on that semaphore.39244 If the semaphore value resulting from this operation is positive, then no threads were blocked  
39245 waiting for the semaphore to become unlocked; the semaphore value is simply incremented.39246 If the value of the semaphore resulting from this operation is zero, then one of the threads  
39247 blocked waiting for the semaphore shall be allowed to return successfully from its call to  
39248 *sem\_wait()*. If the Process Scheduling option is supported, the thread to be unblocked shall be  
39249 chosen in a manner appropriate to the scheduling policies and parameters in effect for the  
39250 blocked threads. In the case of the schedulers SCHED\_FIFO and SCHED\_RR, the highest  
39251 priority waiting thread shall be unblocked, and if there is more than one highest priority thread  
39252 blocked waiting for the semaphore, then the highest priority thread that has been waiting the  
39253 longest shall be unblocked. If the Process Scheduling option is not defined, the choice of a thread  
39254 to unblock is unspecified.39255 SS If the Process Sporadic Server option is supported, and the scheduling policy is  
39256 SCHED\_SPORADIC, the semantics are as per SCHED\_FIFO above.39257 The *sem\_post()* function shall be reentrant with respect to signals and may be invoked from a  
39258 signal-catching function.39259 **RETURN VALUE**39260 If successful, the *sem\_post()* function shall return zero; otherwise, the function shall return -1  
39261 and set *errno* to indicate the error.39262 **ERRORS**39263 The *sem\_post()* function shall fail if:39264 [EINVAL] The *sem* argument does not refer to a valid semaphore.39265 **EXAMPLES**

39266 None.

39267 **APPLICATION USAGE**39268 The *sem\_post()* function is part of the Semaphores option and need not be available on all  
39269 implementations.39270 **RATIONALE**

39271 None.

39272 **FUTURE DIRECTIONS**

39273 None.

39274 **SEE ALSO**39275 *semctl()*, *semget()*, *semop()*, *sem\_timedwait()*, *sem\_trywait()*, *sem\_wait()*, the Base Definitions  
39276 volume of IEEE Std 1003.1-200x, <semaphore.h>

39277 **CHANGE HISTORY**

39278 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

39279 **Issue 6**

39280 The *sem\_post()* function is marked as part of the Semaphores option.

39281 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
39282 implementation does not support the Semaphores option.

39283 The *sem\_timedwait()* function is added to the SEE ALSO section for alignment with  
39284 IEEE Std 1003.1d-1999.

39285 SCHED\_SPORADIC is added to the list of scheduling policies for which the thread that is to be  
39286 unblocked is specified for alignment with IEEE Std 1003.1d-1999.

39287 **NAME**39288 sem\_timedwait — lock a semaphore (**ADVANCED REALTIME**)39289 **SYNOPSIS**

39290 SEM TMO #include &lt;semaphore.h&gt;

39291 #include &lt;time.h&gt;

```
39292 int sem_timedwait(sem_t *restrict sem,
39293                  const struct timespec *restrict abs_timeout);
39294
```

39295 **DESCRIPTION**

39296 The *sem\_timedwait()* function shall lock the semaphore referenced by *sem* as in the *sem\_wait()*  
 39297 function. However, if the semaphore cannot be locked without waiting for another process or  
 39298 thread to unlock the semaphore by performing a *sem\_post()* function, this wait shall be  
 39299 terminated when the specified timeout expires.

39300 The timeout shall expire when the absolute time specified by *abs\_timeout* passes, as measured by  
 39301 the clock on which timeouts are based (that is, when the value of that clock equals or exceeds  
 39302 *abs\_timeout*), or if the absolute time specified by *abs\_timeout* has already been passed at the time  
 39303 of the call.

39304 TMR If the Timers option is supported, the timeout shall be based on the `CLOCK_REALTIME` clock. If  
 39305 the Timers option is not supported, the timeout shall be based on the system clock as returned  
 39306 by the *time()* function. The resolution of the timeout shall be the resolution of the clock on which  
 39307 it is based. The **timespec** data type is defined as a structure in the `<time.h>` header.

39308 Under no circumstance shall the function fail with a timeout if the semaphore can be locked  
 39309 immediately. The validity of the *abs\_timeout* need not be checked if the semaphore can be locked  
 39310 immediately.

39311 **RETURN VALUE**

39312 The *sem\_timedwait()* function shall return zero if the calling process successfully performed the  
 39313 semaphore lock operation on the semaphore designated by *sem*. If the call was unsuccessful, the  
 39314 state of the semaphore shall be unchanged, and the function shall return a value of `-1` and set  
 39315 *errno* to indicate the error.

39316 **ERRORS**39317 The *sem\_timedwait()* function shall fail if:39318 [EINVAL] The *sem* argument does not refer to a valid semaphore.

39319 [EINVAL] The process or thread would have blocked, and the *abs\_timeout* parameter  
 39320 specified a nanoseconds field value less than zero or greater than or equal to  
 39321 1 000 million.

39322 [ETIMEDOUT] The semaphore could not be locked before the specified timeout expired.

39323 The *sem\_timedwait()* function may fail if:

39324 [EDEADLK] A deadlock condition was detected.

39325 [EINTR] A signal interrupted this function.

**39326 EXAMPLES**

39327 None.

**39328 APPLICATION USAGE**

39329 Applications using these functions may be subject to priority inversion, as discussed in the Base  
39330 Definitions volume of IEEE Std 1003.1-200x, Section 3.285, Priority Inversion.

39331 The *sem\_timedwait()* function is part of the Semaphores and Timeouts options and need not be  
39332 provided on all implementations.

**39333 RATIONALE**

39334 None.

**39335 FUTURE DIRECTIONS**

39336 None.

**39337 SEE ALSO**

39338 *sem\_post()*, *sem\_trywait()*, *sem\_wait()*, *semctl()*, *semget()*, *semop()*, *time()*, the Base Definitions  
39339 volume of IEEE Std 1003.1-200x, <**semaphore.h**>, <**time.h**>

**39340 CHANGE HISTORY**

39341 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

39342 **NAME**39343 sem\_trywait, sem\_wait — lock a semaphore (**REALTIME**)39344 **SYNOPSIS**

39345 SEM #include &lt;semaphore.h&gt;

39346 int sem\_trywait(sem\_t \*sem);

39347 int sem\_wait(sem\_t \*sem);

39348

39349 **DESCRIPTION**

39350 The *sem\_trywait()* function shall lock the semaphore referenced by *sem* only if the semaphore is  
 39351 currently not locked; that is, if the semaphore value is currently positive. Otherwise, shall does  
 39352 not lock the semaphore.

39353 The *sem\_wait()* function shall lock the semaphore referenced by *sem* by performing a semaphore  
 39354 lock operation on that semaphore. If the semaphore value is currently zero, then the calling  
 39355 thread shall not return from the call to *sem\_wait()* until it either locks the semaphore or the call is  
 39356 interrupted by a signal.

39357 Upon successful return, the state of the semaphore shall be locked and shall remain locked until  
 39358 the *sem\_post()* function is executed and returns successfully.

39359 The *sem\_wait()* function is interruptible by the delivery of a signal.

39360 **RETURN VALUE**

39361 The *sem\_trywait()* and *sem\_wait()* functions shall return zero if the calling process successfully  
 39362 performed the semaphore lock operation on the semaphore designated by *sem*. If the call was  
 39363 unsuccessful, the state of the semaphore shall be unchanged, and the function shall return a  
 39364 value of  $-1$  and set *errno* to indicate the error.

39365 **ERRORS**

39366 The *sem\_trywait()* and *sem\_wait()* functions shall fail if:

39367 [EAGAIN] The semaphore was already locked, so it cannot be immediately locked by the  
 39368 *sem\_trywait()* operation (*sem\_trywait()* only).

39369 [EINVAL] The *sem* argument does not refer to a valid semaphore.

39370 The *sem\_trywait()* and *sem\_wait()* functions may fail if:

39371 [EDEADLK] A deadlock condition was detected.

39372 [EINTR] A signal interrupted this function.

39373 **EXAMPLES**

39374 None.

39375 **APPLICATION USAGE**

39376 Applications using these functions may be subject to priority inversion, as discussed in the Base  
 39377 Definitions volume of IEEE Std 1003.1-200x, Section 3.285, Priority Inversion.

39378 The *sem\_trywait()* and *sem\_wait()* functions are part of the Semaphores option and need not be  
 39379 provided on all implementations.

39380 **RATIONALE**

39381 None.

39382 **FUTURE DIRECTIONS**

39383       None.

39384 **SEE ALSO**39385       *semctl()*, *semget()*, *semop()*, *sem\_post()*, *sem\_timedwait()*, the Base Definitions volume of  
39386       IEEE Std 1003.1-200x, <**semaphore.h**>39387 **CHANGE HISTORY**

39388       First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

39389 **Issue 6**39390       The *sem\_trywait()* and *sem\_wait()* functions are marked as part of the Semaphores option.39391       The [ENOSYS] error condition has been removed as stubs need not be provided if an  
39392       implementation does not support the Semaphores option.39393       The *sem\_timedwait()* function is added to the SEE ALSO section for alignment with  
39394       IEEE Std 1003.1d-1999.

39395 **NAME**39396 sem\_unlink — remove a named semaphore (**REALTIME**)39397 **SYNOPSIS**

39398 SEM #include &lt;semaphore.h&gt;

39399 int sem\_unlink(const char \*name);

39400

39401 **DESCRIPTION**

39402 The *sem\_unlink()* function shall remove the semaphore named by the string *name*. If the  
 39403 semaphore named by *name* is currently referenced by other processes, then *sem\_unlink()* shall  
 39404 have no effect on the state of the semaphore. If one or more processes have the semaphore open  
 39405 when *sem\_unlink()* is called, destruction of the semaphore is postponed until all references to the  
 39406 semaphore have been destroyed by calls to *sem\_close()*, *\_exit()*, or *exec*. Calls to *sem\_open()* to  
 39407 recreate or reconnect to the semaphore refer to a new semaphore after *sem\_unlink()* is called. The  
 39408 *sem\_unlink()* call shall not block until all references have been destroyed; it shall return  
 39409 immediately.

39410 **RETURN VALUE**

39411 Upon successful completion, the *sem\_unlink()* function shall return a value of 0. Otherwise, the  
 39412 semaphore shall not be changed and the function shall return a value of -1 and set *errno* to  
 39413 indicate the error.

39414 **ERRORS**39415 The *sem\_unlink()* function shall fail if:

39416 [EACCES] Permission is denied to unlink the named semaphore.

39417 [ENAMETOOLONG]

39418 The length of the *name* argument exceeds {PATH\_MAX} or a pathname  
 39419 component is longer than {NAME\_MAX}.

39420 [ENOENT] The named semaphore does not exist.

39421 **EXAMPLES**

39422 None.

39423 **APPLICATION USAGE**

39424 The *sem\_unlink()* function is part of the Semaphores option and need not be available on all  
 39425 implementations.

39426 **RATIONALE**

39427 None.

39428 **FUTURE DIRECTIONS**

39429 None.

39430 **SEE ALSO**

39431 *semctl()*, *semget()*, *semop()*, *sem\_close()*, *sem\_open()*, the Base Definitions volume of  
 39432 IEEE Std 1003.1-200x, <semaphore.h>

39433 **CHANGE HISTORY**

39434 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

39435 **Issue 6**39436 The *sem\_unlink()* function is marked as part of the Semaphores option.

39437  
39438

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Semaphores option.

39439 **NAME**39440 sem\_wait — lock a semaphore (**REALTIME**)39441 **SYNOPSIS**

39442 SEM #include &lt;semaphore.h&gt;

39443 int sem\_wait(sem\_t \*sem);

39444

39445 **DESCRIPTION**39446 Refer to *sem\_trywait()*.

## 39447 NAME

39448 semctl — XSI semaphore control operations

## 39449 SYNOPSIS

39450 XSI 

```
#include <sys/sem.h>
```

39451 

```
int semctl(int semid, int semnum, int cmd, ...);
```

39452

## 39453 DESCRIPTION

39454 The *semctl()* function operates on XSI semaphores (see the Base Definitions volume of |  
 39455 IEEE Std 1003.1-200x, Section 4.15, Semaphore). It is unspecified whether this function |  
 39456 interoperates with the realtime interprocess communication facilities defined in Section 2.8 (on  
 39457 page 491).

39458 The *semctl()* function provides a variety of semaphore control operations as specified by *cmd*.  
 39459 The fourth argument is optional and depends upon the operation requested. If required, it is of  
 39460 type **union semun**, which the application shall explicitly declare:

```
39461 union semun {
39462     int val;
39463     struct semid_ds *buf;
39464     unsigned short *array;
39465 } arg;
```

39466 The following semaphore control operations as specified by *cmd* are executed with respect to the  
 39467 semaphore specified by *semid* and *semnum*. The level of permission required for each operation  
 39468 is shown with each command; see Section 2.7 (on page 489). The symbolic names for the values  
 39469 of *cmd* are defined in the `<sys/sem.h>` header:

39470 GETVAL Return the value of *semval*; see `<sys/sem.h>`. Requires read permission.

39471 SETVAL Set the value of *semval* to *arg.val*, where *arg* is the value of the fourth argument  
 39472 to *semctl()*. When this command is successfully executed, the *semadj* value  
 39473 corresponding to the specified semaphore in all processes is cleared. Requires  
 39474 alter permission; see Section 2.7 (on page 489).

39475 GETPID Return the value of *sempid*. Requires read permission.

39476 GETNCNT Return the value of *semmcnt*. Requires read permission.

39477 GETZCNT Return the value of *semzcnt*. Requires read permission.

39478 The following values of *cmd* operate on each *semval* in the set of semaphores:

39479 GETALL Return the value of *semval* for each semaphore in the semaphore set and place  
 39480 into the array pointed to by *arg.array*, where *arg* is the fourth argument to  
 39481 *semctl()*. Requires read permission.

39482 SETALL Set the value of *semval* for each semaphore in the semaphore set according to  
 39483 the array pointed to by *arg.array*, where *arg* is the fourth argument to *semctl()*.  
 39484 When this command is successfully executed, the *semadj* values corresponding  
 39485 to each specified semaphore in all processes are cleared. Requires alter  
 39486 permission.

39487 The following values of *cmd* are also available:

39488 IPC\_STAT Place the current value of each member of the **semid\_ds** data structure  
 39489 associated with *semid* into the structure pointed to by *arg.buf*, where *arg* is the  
 39490 fourth argument to *semctl()*. The contents of this structure are defined in

39491		<sys/sem.h>. Requires read permission.
39492	IPC_SET	Set the value of the following members of the <b>semid_ds</b> data structure associated with <i>semid</i> to the corresponding value found in the structure pointed to by <i>arg.buf</i> , where <i>arg</i> is the fourth argument to <i>semctl()</i> :
39493		
39494		
39495		<i>sem_perm.uid</i>
39496		<i>sem_perm.gid</i>
39497		<i>sem_perm.mode</i>
39498		The mode bits specified in Section 2.7.1 (on page 490) are copied into the corresponding bits of the <i>sem_perm.mode</i> associated with <i>semid</i> . The stored values of any other bits are unspecified.
39499		
39500		
39501		This command can only be executed by a process that has an effective user ID equal to either that of a process with appropriate privileges or to the value of <i>sem_perm.cuid</i> or <i>sem_perm.uid</i> in the <b>semid_ds</b> data structure associated with <i>semid</i> .
39502		
39503		
39504		
39505	IPC_RMID	Remove the semaphore identifier specified by <i>semid</i> from the system and destroy the set of semaphores and <b>semid_ds</b> data structure associated with it. This command can only be executed by a process that has an effective user ID equal to either that of a process with appropriate privileges or to the value of <i>sem_perm.cuid</i> or <i>sem_perm.uid</i> in the <b>semid_ds</b> data structure associated with <i>semid</i> .
39506		
39507		
39508		
39509		
39510		
39511	<b>RETURN VALUE</b>	
39512		If successful, the value returned by <i>semctl()</i> depends on <i>cmd</i> as follows:
39513	GETVAL	The value of <i>semval</i> .
39514	GETPID	The value of <i>sempid</i> .
39515	GETNCNT	The value of <i>semmcnt</i> .
39516	GETZCNT	The value of <i>semzcnt</i> .
39517	All others	0.
39518		Otherwise, <i>semctl()</i> shall return $-1$ and set <i>errno</i> to indicate the error.
39519	<b>ERRORS</b>	
39520		The <i>semctl()</i> function shall fail if:
39521	[EACCES]	Operation permission is denied to the calling process; see Section 2.7 (on page 489).
39522		
39523	[EINVAL]	The value of <i>semid</i> is not a valid semaphore identifier, or the value of <i>semnum</i> is less than 0 or greater than or equal to <i>sem_nsems</i> , or the value of <i>cmd</i> is not a valid command.
39524		
39525		
39526	[EPERM]	The argument <i>cmd</i> is equal to IPC_RMID or IPC_SET and the effective user ID of the calling process is not equal to that of a process with appropriate privileges and it is not equal to the value of <i>sem_perm.cuid</i> or <i>sem_perm.uid</i> in the data structure associated with <i>semid</i> .
39527		
39528		
39529		
39530	[ERANGE]	The argument <i>cmd</i> is equal to SETVAL or SETALL and the value to which <i>semval</i> is to be set is greater than the system-imposed maximum.
39531		

39532 **EXAMPLES**

39533 None.

39534 **APPLICATION USAGE**

39535 The fourth parameter in the SYNOPSIS section is now specified as ". . ." in order to avoid a  
39536 clash with the ISO C standard when referring to the union *semun* (as defined in Issue 3) and for  
39537 backward compatibility.

39538 The POSIX Realtime Extension defines alternative interfaces for interprocess communication.  
39539 Application developers who need to use IPC should design their applications so that modules  
39540 using the IPC routines described in Section 2.7 (on page 489) can be easily modified to use the  
39541 alternative interfaces.

39542 **RATIONALE**

39543 None.

39544 **FUTURE DIRECTIONS**

39545 None.

39546 **SEE ALSO**

39547 *semget()*, *semop()*, *sem\_close()*, *sem\_destroy()*, *sem\_getvalue()*, *sem\_init()*, *sem\_open()*, *sem\_post()*,  
39548 *sem\_unlink()*, *sem\_wait()*, the Base Definitions volume of IEEE Std 1003.1-200x, `<sys/sem.h>`,  
39549 Section 2.7 (on page 489)

39550 **CHANGE HISTORY**

39551 First released in Issue 2. Derived from Issue 2 of the SVID.

39552 **Issue 5**

39553 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE  
39554 DIRECTIONS to the APPLICATION USAGE section.

39555 **NAME**39556 `semget` — get set of XSI semaphores39557 **SYNOPSIS**39558 XSI 

```
#include <sys/sem.h>
```

39559 

```
int semget(key_t key, int nsems, int semflg);
```

39560

39561 **DESCRIPTION**

39562 The `semget()` function operates on XSI semaphores (see the Base Definitions volume of |  
 39563 IEEE Std 1003.1-200x, Section 4.15, Semaphore). It is unspecified whether this function |  
 39564 interoperates with the realtime interprocess communication facilities defined in Section 2.8 (on |  
 39565 page 491).

39566 The `semget()` function shall return the semaphore identifier associated with *key*.

39567 A semaphore identifier with its associated **semid\_ds** data structure and its associated set of |  
 39568 *nsems* semaphores (see <sys/sem.h>) is created for *key* if one of the following is true:

- 39569 • The argument *key* is equal to `IPC_PRIVATE`.
- 39570 • The argument *key* does not already have a semaphore identifier associated with it and (*semflg* |  
 39571 &`IPC_CREAT`) is non-zero.

39572 Upon creation, the **semid\_ds** data structure associated with the new semaphore identifier is |  
 39573 initialized as follows:

- 39574 • In the operation permissions structure *sem\_perm.cuid*, *sem\_perm.uid*, *sem\_perm.cgid*, and |  
 39575 *sem\_perm.gid* shall be set equal to the effective user ID and effective group ID, respectively, of |  
 39576 the calling process.
- 39577 • The low-order 9 bits of *sem\_perm.mode* shall be set equal to the low-order 9 bits of *semflg*. |
- 39578 • The variable *sem\_nsems* shall be set equal to the value of *nsems*. |
- 39579 • The variable *sem\_otime* shall be set equal to 0 and *sem\_ctime* shall be set equal to the current |  
 39580 time. |
- 39581 • The data structure associated with each semaphore in the set shall not be initialized. The |  
 39582 *semctl()* function with the command `SETVAL` or `SETALL` can be used to initialize each |  
 39583 semaphore.

39584 **RETURN VALUE**

39585 Upon successful completion, `semget()` shall return a non-negative integer, namely a semaphore |  
 39586 identifier; otherwise, it shall return `-1` and set *errno* to indicate the error.

39587 **ERRORS**

39588 The `semget()` function shall fail if:

- 39589 [EACCES] A semaphore identifier exists for *key*, but operation permission as specified by |  
 39590 the low-order 9 bits of *semflg* would not be granted; see Section 2.7 (on page |  
 39591 489).
- 39592 [EEXIST] A semaphore identifier exists for the argument *key* but ((*semflg* &`IPC_CREAT`) |  
 39593 &&( *semflg* &`IPC_EXCL`)) is non-zero.
- 39594 [EINVAL] The value of *nsems* is either less than or equal to 0 or greater than the system- |  
 39595 imposed limit, or a semaphore identifier exists for the argument *key*, but the |  
 39596 number of semaphores in the set associated with it is less than *nsems* and |  
 39597 *nsems* is not equal to 0.

39598 [ENOENT] A semaphore identifier does not exist for the argument *key* and (*semflg*  
 39599 &IPC\_CREAT) is equal to 0.

39600 [ENOSPC] A semaphore identifier is to be created but the system-imposed limit on the  
 39601 maximum number of allowed semaphores system-wide would be exceeded.

39602 **EXAMPLES**39603 **Creating a Semaphore Identifier**

39604 The following example gets a unique semaphore key using the *ftok()* function, then gets a  
 39605 semaphore ID associated with that key using the *semget()* function (the first call also tests to  
 39606 make sure the semaphore exists). If the semaphore does not exist, the program creates it, as  
 39607 shown by the second call to *semget()*. In creating the semaphore for the queuing process, the  
 39608 program attempts to create one semaphore with read/write permission for all. It also uses the  
 39609 IPC\_EXCL flag, which forces *semget()* to fail if the semaphore already exists.

39610 After creating the semaphore, the program uses a call to *semop()* to initialize it to the values in  
 39611 the *sbuf* array. The number of processes that can execute concurrently without queuing is  
 39612 initially set to 2. The final call to *semget()* creates a semaphore identifier that can be used later in  
 39613 the program.

```

39614 #include <sys/types.h>
39615 #include <stdio.h>
39616 #include <sys/ipc.h>
39617 #include <sys/sem.h>
39618 #include <sys/stat.h>
39619 #include <errno.h>
39620 #include <unistd.h>
39621 #include <stdlib.h>
39622 #include <pwd.h>
39623 #include <fcntl.h>
39624 #include <limits.h>
39625 ...
39626 key_t semkey;
39627 int semid, pfd, fv;
39628 struct sembuf sbuf;
39629 char *lgn;
39630 char filename[PATH_MAX+1];
39631 struct stat outstat;
39632 struct passwd *pw;
39633 ...
39634 /* Get unique key for semaphore. */
39635 if ((semkey = ftok("/tmp", 'a')) == (key_t) -1) {
39636     perror("IPC error: ftok"); exit(1);
39637 }
39638 /* Get semaphore ID associated with this key. */
39639 if ((semid = semget(semkey, 0, 0)) == -1) {
39640     /* Semaphore does not exist - Create. */
39641     if ((semid = semget(semkey, 1, IPC_CREAT | IPC_EXCL | S_IRUSR |
39642         S_IWUSR | S_IRGRP | S_IWGRP | S_IROTH | S_IWOTH)) != -1)
39643     {
39644         /* Initialize the semaphore. */
39645         sbuf.sem_num = 0;

```

```
39646         sbuf.sem_op = 2; /* This is the number of runs without queuing. */
39647         sbuf.sem_flg = 0;
39648         if (semop(semid, &sbuf, 1) == -1) {
39649             perror("IPC error: semop"); exit(1);
39650         }
39651     }
39652     else if (errno == EEXIST) {
39653         if ((semid = semget(semkey, 0, 0)) == -1) {
39654             perror("IPC error 1: semget"); exit(1);
39655         }
39656     }
39657     else {
39658         perror("IPC error 2: semget"); exit(1);
39659     }
39660 }
39661 ...
```

#### 39662 APPLICATION USAGE

39663 The POSIX Realtime Extension defines alternative interfaces for interprocess communication.  
39664 Application developers who need to use IPC should design their applications so that modules  
39665 using the IPC routines described in Section 2.7 (on page 489) can be easily modified to use the  
39666 alternative interfaces.

#### 39667 RATIONALE

39668 None.

#### 39669 FUTURE DIRECTIONS

39670 None.

#### 39671 SEE ALSO

39672 *semctl()*, *semop()*, *sem\_close()*, *sem\_destroy()*, *sem\_getvalue()*, *sem\_init()*, *sem\_open()*, *sem\_post()*,  
39673 *sem\_unlink()*, *sem\_wait()*, the Base Definitions volume of IEEE Std 1003.1-200x, <sys/sem.h>,  
39674 Section 2.7 (on page 489).

#### 39675 CHANGE HISTORY

39676 First released in Issue 2. Derived from Issue 2 of the SVID.

#### 39677 Issue 5

39678 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE  
39679 DIRECTIONS to a new APPLICATION USAGE section.

## 39680 NAME

39681 semop — XSI semaphore operations

## 39682 SYNOPSIS

39683 XSI #include &lt;sys/sem.h&gt;

39684 int semop(int *semid*, struct sembuf \**sops*, size\_t *nsops*);

39685

## 39686 DESCRIPTION

39687 The *semop()* function operates on XSI semaphores (see the Base Definitions volume of |  
 39688 IEEE Std 1003.1-200x, Section 4.15, Semaphore). It is unspecified whether this function |  
 39689 interoperates with the realtime interprocess communication facilities defined in Section 2.8 (on |  
 39690 page 491).

39691 The *semop()* function shall perform atomically a user-defined array of semaphore operations on |  
 39692 the set of semaphores associated with the semaphore identifier specified by the argument *semid*. |

39693 The argument *sops* is a pointer to a user-defined array of semaphore operation structures. The |  
 39694 implementation shall not modify elements of this array unless the application uses |  
 39695 implementation-defined extensions.

39696 The argument *nsops* is the number of such structures in the array.

39697 Each structure, **sembuf**, includes the following members:

39698

39699

39700

39701

39702

Member Type	Member Name	Description
short	<i>sem_num</i>	Semaphore number.
short	<i>sem_op</i>	Semaphore operation.
short	<i>sem_flg</i>	Operation flags.

39703 Each semaphore operation specified by *sem\_op* is performed on the corresponding semaphore |  
 39704 specified by *semid* and *sem\_num*.

39705 The variable *sem\_op* specifies one of three semaphore operations:

39706 1. If *sem\_op* is a negative integer and the calling process has alter permission, one of the |  
 39707 following shall occur:

39708 • If *semval* (see <sys/sem.h>) is greater than or equal to the absolute value of *sem\_op*, the |  
 39709 absolute value of *sem\_op* is subtracted from *semval*. Also, if (*sem\_flg* &SEM\_UNDO) is |  
 39710 non-zero, the absolute value of *sem\_op* shall be added to the calling process' *semadj* |  
 39711 value for the specified semaphore.

39712 • If *semval* is less than the absolute value of *sem\_op* and (*sem\_flg* &IPC\_NOWAIT) is non- |  
 39713 zero, *semop()* shall return immediately.

39714 • If *semval* is less than the absolute value of *sem\_op* and (*sem\_flg* &IPC\_NOWAIT) is 0, |  
 39715 *semop()* shall increment the *semncnt* associated with the specified semaphore and |  
 39716 suspend execution of the calling thread until one of the following conditions occurs:

39717 — The value of *semval* becomes greater than or equal to the absolute value of *sem\_op*. |  
 39718 When this occurs, the value of *semncnt* associated with the specified semaphore |  
 39719 shall be decremented, the absolute value of *sem\_op* shall be subtracted from *semval* |  
 39720 and, if (*sem\_flg* &SEM\_UNDO) is non-zero, the absolute value of *sem\_op* shall be |  
 39721 added to the calling process' *semadj* value for the specified semaphore. |

39722 — The *semid* for which the calling thread is awaiting action is removed from the |  
 39723 system. When this occurs, *errno* shall be set equal to [EIDRM] and *-1* shall be

39724 returned.

39725 — The calling thread receives a signal that is to be caught. When this occurs, the value  
39726 of *semncnt* associated with the specified semaphore shall be decremented, and the  
39727 calling thread shall resume execution in the manner prescribed in *sigaction()*.

39728 2. If *sem\_op* is a positive integer and the calling process has alter permission, the value of  
39729 *sem\_op* shall be added to *semval* and, if (*sem\_flg* &SEM\_UNDO) is non-zero, the value of  
39730 *sem\_op* shall be subtracted from the calling process' *semadj* value for the specified  
39731 semaphore.

39732 3. If *sem\_op* is 0 and the calling process has read permission, one of the following shall occur:

39733 • If *semval* is 0, *semop()* shall return immediately.

39734 • If *semval* is non-zero and (*sem\_flg* &IPC\_NOWAIT) is non-zero, *semop()* shall return  
39735 immediately.

39736 • If *semval* is non-zero and (*sem\_flg* &IPC\_NOWAIT) is 0, *semop()* shall increment the  
39737 *semzcnt* associated with the specified semaphore and suspend execution of the calling  
39738 thread until one of the following occurs:

39739 — The value of *semval* becomes 0, at which time the value of *semzcnt* associated with  
39740 the specified semaphore shall be decremented.

39741 — The *semid* for which the calling thread is awaiting action is removed from the  
39742 system. When this occurs, *errno* shall be set equal to [EIDRM] and -1 shall be  
39743 returned.

39744 — The calling thread receives a signal that is to be caught. When this occurs, the value  
39745 of *semzcnt* associated with the specified semaphore shall be decremented, and the  
39746 calling thread shall resume execution in the manner prescribed in *sigaction()*.

39747 Upon successful completion, the value of *sempid* for each semaphore specified in the array  
39748 pointed to by *sops* shall be set equal to the process ID of the calling process.

39749 **RETURN VALUE**

39750 Upon successful completion, *semop()* shall return 0; otherwise, it shall return -1 and set *errno* to  
39751 indicate the error.

39752 **ERRORS**

39753 The *semop()* function shall fail if:

39754 [E2BIG] The value of *nsops* is greater than the system-imposed maximum.

39755 [EACCES] Operation permission is denied to the calling process; see Section 2.7 (on page  
39756 489).

39757 [EAGAIN] The operation would result in suspension of the calling process but (*sem\_flg*  
39758 &IPC\_NOWAIT) is non-zero.

39759 [EFBIG] The value of *sem\_num* is less than 0 or greater than or equal to the number of  
39760 semaphores in the set associated with *semid*.

39761 [EIDRM] The semaphore identifier *semid* is removed from the system.

39762 [EINTR] The *semop()* function was interrupted by a signal.

39763 [EINVAL] The value of *semid* is not a valid semaphore identifier, or the number of  
39764 individual semaphores for which the calling process requests a SEM\_UNDO  
39765 would exceed the system-imposed limit.

39766 [ENOSPC] The limit on the number of individual processes requesting a SEM\_UNDO  
 39767 would be exceeded.

39768 [ERANGE] An operation would cause a *semval* to overflow the system-imposed limit, or  
 39769 an operation would cause a *semadj* value to overflow the system-imposed  
 39770 limit.

### 39771 EXAMPLES

#### 39772 Setting Values in Semaphores

39773 The following example sets the values of the two semaphores associated with the *semid*  
 39774 identifier to the values contained in the *sb* array.

```
39775 #include <sys/sem.h>
39776 ...
39777 int semid;
39778 struct sembuf sb[2];
39779 int nsops = 2;
39780 int result;

39781 /* Adjust value of semaphore in the semaphore array semid. */
39782 sb[0].sem_num = 0;
39783 sb[0].sem_op = -1;
39784 sb[0].sem_flg = SEM_UNDO | IPC_NOWAIT;
39785 sb[1].sem_num = 1;
39786 sb[1].sem_op = 1;
39787 sb[1].sem_flg = 0;

39788 result = semop(semid, sb, nsops);
```

#### 39789 Creating a Semaphore Identifier

39790 The following example gets a unique semaphore key using the *ftok()* function, then gets a  
 39791 semaphore ID associated with that key using the *semget()* function (the first call also tests to  
 39792 make sure the semaphore exists). If the semaphore does not exist, the program creates it, as  
 39793 shown by the second call to *semget()*. In creating the semaphore for the queuing process, the  
 39794 program attempts to create one semaphore with read/write permission for all. It also uses the  
 39795 IPC\_EXCL flag, which forces *semget()* to fail if the semaphore already exists.

39796 After creating the semaphore, the program uses a call to *semop()* to initialize it to the values in  
 39797 the *sbuf* array. The number of processes that can execute concurrently without queuing is  
 39798 initially set to 2. The final call to *semget()* creates a semaphore identifier that can be used later in  
 39799 the program.

39800 The final call to *semop()* acquires the semaphore and waits until it is free; the SEM\_UNDO  
 39801 option releases the semaphore when the process exits, waiting until there are less than two  
 39802 processes running concurrently.

```
39803 #include <sys/types.h>
39804 #include <stdio.h>
39805 #include <sys/ipc.h>
39806 #include <sys/sem.h>
39807 #include <sys/stat.h>
39808 #include <errno.h>
39809 #include <unistd.h>
39810 #include <stdlib.h>
```

```

39811     #include <pwd.h>
39812     #include <fcntl.h>
39813     #include <limits.h>
39814     ...
39815     key_t semkey;
39816     int semid, pfd, fv;
39817     struct sembuf sbuf;
39818     char *lgn;
39819     char filename[PATH_MAX+1];
39820     struct stat outstat;
39821     struct passwd *pw;
39822     ...
39823     /* Get unique key for semaphore. */
39824     if ((semkey = ftok("/tmp", 'a')) == (key_t) -1) {
39825         perror("IPC error: ftok"); exit(1);
39826     }
39827     /* Get semaphore ID associated with this key. */
39828     if ((semid = semget(semkey, 0, 0)) == -1) {
39829         /* Semaphore does not exist - Create. */
39830         if ((semid = semget(semkey, 1, IPC_CREAT | IPC_EXCL | S_IRUSR |
39831             S_IWUSR | S_IRGRP | S_IWGRP | S_IROTH | S_IWOTH)) != -1)
39832         {
39833             /* Initialize the semaphore. */
39834             sbuf.sem_num = 0;
39835             sbuf.sem_op = 2; /* This is the number of runs without queuing. */
39836             sbuf.sem_flg = 0;
39837             if (semop(semid, &sbuf, 1) == -1) {
39838                 perror("IPC error: semop"); exit(1);
39839             }
39840         }
39841         else if (errno == EEXIST) {
39842             if ((semid = semget(semkey, 0, 0)) == -1) {
39843                 perror("IPC error 1: semget"); exit(1);
39844             }
39845         }
39846         else {
39847             perror("IPC error 2: semget"); exit(1);
39848         }
39849     }
39850     ...
39851     sbuf.sem_num = 0;
39852     sbuf.sem_op = -1;
39853     sbuf.sem_flg = SEM_UNDO;
39854     if (semop(semid, &sbuf, 1) == -1) {
39855         perror("IPC Error: semop"); exit(1);
39856     }

```

### 39857 APPLICATION USAGE

39858 The POSIX Realtime Extension defines alternative interfaces for interprocess communication.  
39859 Application developers who need to use IPC should design their applications so that modules  
39860 using the IPC routines described in Section 2.7 (on page 489) can be easily modified to use the  
39861 alternative interfaces.

39862 **RATIONALE**

39863           None.

39864 **FUTURE DIRECTIONS**

39865           None.

39866 **SEE ALSO**

39867           *exec*, *exit()*, *fork()*, *semctl()*, *semget()*, *sem\_close()*, *sem\_destroy()*, *sem\_getvalue()*, *sem\_init()*,  
39868           *sem\_open()*, *sem\_post()*, *sem\_unlink()*, *sem\_wait()*, the Base Definitions volume of  
39869           IEEE Std 1003.1-200x, <sys/ipc.h>, <sys/sem.h>, <sys/types.h>, Section 2.7 (on page 489)

39870 **CHANGE HISTORY**

39871           First released in Issue 2. Derived from Issue 2 of the SVID.

39872 **Issue 5**

39873           The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE  
39874           DIRECTIONS to a new APPLICATION USAGE section.

39875 **NAME**39876 `send` — send a message on a socket39877 **SYNOPSIS**39878 `#include <sys/socket.h>`39879 `ssize_t send(int socket, const void *buffer, size_t length, int flags);`39880 **DESCRIPTION**39881 The `send()` function shall initiate transmission of a message from the specified socket to its peer.39882 The `send()` function shall send a message only when the socket is connected (including when the peer of a connectionless socket has been set via `connect()`).39884 The `send()` functions takes the following arguments:39885 *socket* Specifies the socket file descriptor.39886 *buffer* Points to the buffer containing the message to send.39887 *length* Specifies the length of the message in bytes.39888 *flags* Specifies the type of message transmission. Values of this argument are formed by logically OR'ing zero or more of the following flags:39890 `MSG_EOR` Terminates a record (if supported by the protocol).39891 `MSG_OOB` Sends out-of-band data on sockets that support out-of-band communications. The significance and semantics of out-of-band data are protocol-specific.39894 The length of the message to be sent is specified by the *length* argument. If the message is too long to pass through the underlying protocol, `send()` shall fail and no data shall be transmitted.39896 Successful completion of a call to `send()` does not guarantee delivery of the message. A return value of `-1` indicates only locally-detected errors.39898 If space is not available at the sending socket to hold the message to be transmitted, and the socket file descriptor does not have `O_NONBLOCK` set, `send()` shall block until space is available. If space is not available at the sending socket to hold the message to be transmitted, and the socket file descriptor does have `O_NONBLOCK` set, `send()` shall fail. The `select()` and `poll()` functions can be used to determine when it is possible to send more data.39903 The socket in use may require the process to have appropriate privileges to use the `send()` function.39905 **RETURN VALUE**39906 Upon successful completion, `send()` shall return the number of bytes sent. Otherwise, `-1` shall be returned and `errno` set to indicate the error.39908 **ERRORS**39909 The `send()` function shall fail if:39910 `[EAGAIN]` or `[EWOULDBLOCK]`39911 The socket's file descriptor is marked `O_NONBLOCK` and the requested operation would block.39913 `[EBADF]` The *socket* argument is not a valid file descriptor.39914 `[ECONNRESET]` A connection was forcibly closed by a peer.39915 `[EDESTADDRREQ]`

39916 The socket is not connection-mode and no peer address is set.

- 39917 [EINTR] A signal interrupted *send()* before any data was transmitted.
- 39918 [EMSGSIZE] The message is too large to be sent all at once, as the socket requires.
- 39919 [ENOTCONN] The socket is not connected or otherwise has not had the peer pre-specified.
- 39920 [ENOTSOCK] The *socket* argument does not refer to a socket.
- 39921 [EOPNOTSUPP] The *socket* argument is associated with a socket that does not support one or  
39922 more of the values set in *flags*.
- 39923 [EPIPE] The socket is shut down for writing, or the socket is connection-mode and is  
39924 no longer connected. In the latter case, and if the socket is of type  
39925 SOCK\_STREAM, the SIGPIPE signal is generated to the calling thread.
- 39926 The *send()* function may fail if:
- 39927 [EACCES] The calling process does not have the appropriate privileges.
- 39928 [EIO] An I/O error occurred while reading from or writing to the file system.
- 39929 [ENETDOWN] The local network interface used to reach the destination is down.
- 39930 [ENETUNREACH]  
39931 No route to the network is present.
- 39932 [ENOBUFS] Insufficient resources were available in the system to perform the operation.
- 39933 **EXAMPLES**
- 39934 None.
- 39935 **APPLICATION USAGE**
- 39936 The *send()* function is equivalent to *sendto()* with a null pointer *dest\_len* argument, and to *write()* |  
39937 if no flags are used.
- 39938 **RATIONALE**
- 39939 None.
- 39940 **FUTURE DIRECTIONS**
- 39941 None.
- 39942 **SEE ALSO**
- 39943 *connect()*, *getsockopt()*, *poll()*, *recv()*, *recvfrom()*, *recvmsg()*, *select()*, *sendmsg()*, *sendto()*,  
39944 *setsockopt()*, *shutdown()*, *socket()*, the Base Definitions volume of IEEE Std 1003.1-200x,  
39945 <sys/socket.h>
- 39946 **CHANGE HISTORY**
- 39947 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

39948 **NAME**

39949 sendmsg — send a message on a socket using a message structure

39950 **SYNOPSIS**

39951 #include &lt;sys/socket.h&gt;

39952 ssize\_t sendmsg(int *socket*, const struct msghdr \**message*, int *flags*);39953 **DESCRIPTION**

39954 The *sendmsg()* function shall send a message through a connection-mode or connectionless-  
 39955 mode socket. If the socket is connectionless-mode, the message shall be sent to the address  
 39956 specified by **msghdr**. If the socket is connection-mode, the destination address in **msghdr** shall  
 39957 be ignored.

39958 The *sendmsg()* function takes the following arguments:

39959	<i>socket</i>	Specifies the socket file descriptor.
39960	<i>message</i>	Points to a <b>msghdr</b> structure, containing both the destination address and the buffers for the outgoing message. The length and format of the address depend on the address family of the socket. The <i>msg_flags</i> member is ignored.
39963	<i>flags</i>	Specifies the type of message transmission. The application may specify 0 or the following flag:
39965	MSG_EOR	Terminates a record (if supported by the protocol).
39966	MSG_OOB	Sends out-of-band data on sockets that support out-of-band data. The significance and semantics of out-of-band data are protocol-specific.

39969 The *msg\_iov* and *msg\_iovlen* fields of *message* specify zero or more buffers containing the data to  
 39970 be sent. *msg\_iov* points to an array of **iovec** structures; *msg\_iovlen* shall be set to the dimension of  
 39971 this array. In each **iovec** structure, the *iov\_base* field specifies a storage area and the *iov\_len* field  
 39972 gives its size in bytes. Some of these sizes can be zero. The data from each storage area indicated  
 39973 by *msg\_iov* is sent in turn.

39974 Successful completion of a call to *sendmsg()* does not guarantee delivery of the message. A  
 39975 return value of  $-1$  indicates only locally-detected errors.

39976 If space is not available at the sending socket to hold the message to be transmitted and the  
 39977 socket file descriptor does not have **O\_NONBLOCK** set, *sendmsg()* function shall block until  
 39978 space is available. If space is not available at the sending socket to hold the message to be  
 39979 transmitted and the socket file descriptor does have **O\_NONBLOCK** set, *sendmsg()* function  
 39980 shall fail.

39981 If the socket protocol supports broadcast and the specified address is a broadcast address for the  
 39982 socket protocol, *sendmsg()* shall fail if the **SO\_BROADCAST** option is not set for the socket.

39983 The socket in use may require the process to have appropriate privileges to use the *sendmsg()*  
 39984 function.

39985 **RETURN VALUE**

39986 Upon successful completion, *sendmsg()* shall return the number of bytes sent. Otherwise,  $-1$   
 39987 shall be returned and *errno* set to indicate the error.

39988 **ERRORS**39989 The *sendmsg()* function shall fail if:

39990 [EAGAIN] or [EWOULDBLOCK]

39991 The socket's file descriptor is marked **O\_NONBLOCK** and the requested

39992		operation would block.
39993	[EAFNOSUPPORT]	
39994		Addresses in the specified address family cannot be used with this socket.
39995	[EBADF]	The <i>socket</i> argument is not a valid file descriptor.
39996	[ECONNRESET]	A connection was forcibly closed by a peer.
39997	[EINTR]	A signal interrupted <i>sendmsg()</i> before any data was transmitted.
39998	[EINVAL]	The sum of the <i>iov_len</i> values overflows an <i>ssize_t</i> .
39999	[EMSGSIZE]	The message is too large to be sent all at once (as the socket requires), or the <i>msg_iovlen</i> member of the <i>msg_hdr</i> structure pointed to by <i>message</i> is less than or equal to 0 or is greater than {IOV_MAX}.
40000		
40001		
40002	[ENOTCONN]	The socket is connection-mode but is not connected.
40003	[ENOTSOCK]	The <i>socket</i> argument does not refer a socket.
40004	[EOPNOTSUPP]	The <i>socket</i> argument is associated with a socket that does not support one or more of the values set in <i>flags</i> .
40005		
40006	[EPIPE]	The socket is shut down for writing, or the socket is connection-mode and is no longer connected. In the latter case, and if the socket is of type SOCK_STREAM, the SIGPIPE signal is generated to the calling thread.
40007		
40008		
40009		If the address family of the socket is AF_UNIX, then <i>sendmsg()</i> shall fail if:
40010	[EIO]	An I/O error occurred while reading from or writing to the file system.
40011	[ELOOP]	A loop exists in symbolic links encountered during resolution of the pathname
40012		in the socket address.
40013	[ENAMETOOLONG]	
40014		A component of a pathname exceeded {NAME_MAX} characters, or an entire
40015		pathname exceeded {PATH_MAX} characters.
40016	[ENOENT]	A component of the pathname does not name an existing file or the path name
40017		is an empty string.
40018	[ENOTDIR]	A component of the path prefix of the pathname in the socket address is not a
40019		directory.
40020		The <i>sendmsg()</i> function may fail if:
40021	[EACCES]	Search permission is denied for a component of the path prefix; or write
40022		access to the named socket is denied.
40023	[EDESTADDRREQ]	
40024		The socket is not connection-mode and does not have its peer address set, and
40025		no destination address was specified.
40026	[EHOSTUNREACH]	
40027		The destination host cannot be reached (probably because the host is down or
40028		a remote router cannot reach it).
40029	[EIO]	An I/O error occurred while reading from or writing to the file system.
40030	[EISCONN]	A destination address was specified and the socket is already connected.
40031	[ENETDOWN]	The local network interface used to reach the destination is down.

- 40032 [ENETUNREACH]  
40033 No route to the network is present.
- 40034 [ENOBUFS] Insufficient resources were available in the system to perform the operation.
- 40035 [ENOMEM] Insufficient memory was available to fulfill the request.
- 40036 If the address family of the socket is AF\_UNIX, then *sendmsg()* may fail if:
- 40037 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during |  
40038 resolution of the pathname in the socket address. |
- 40039 [ENAMETOOLONG]  
40040 Pathname resolution of a symbolic link produced an intermediate result |  
40041 whose length exceeds {PATH\_MAX}. |
- 40042 **EXAMPLES**  
40043 Done.
- 40044 **APPLICATION USAGE**  
40045 The *select()* and *poll()* functions can be used to determine when it is possible to send more data.
- 40046 **RATIONALE**  
40047 None.
- 40048 **FUTURE DIRECTIONS**  
40049 None.
- 40050 **SEE ALSO**  
40051 *getsockopt()*, *poll()*, *recv()*, *recvfrom()*, *recvmsg()*, *select()*, *send()*, *sendto()*, *setsockopt()*,  
40052 *shutdown()*, *socket()*, the Base Definitions volume of IEEE Std 1003.1-200x, <sys/socket.h>
- 40053 **CHANGE HISTORY**  
40054 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.  
40055 The wording of the mandatory [ELOOP] error condition is updated, and a second optional  
40056 [ELOOP] error condition is added.

## 40057 NAME

40058 sendto — send a message on a socket

## 40059 SYNOPSIS

40060 #include &lt;sys/socket.h&gt;

40061 ssize\_t sendto(int socket, const void \*message, size\_t length,

40062 int flags, const struct sockaddr \*dest\_addr,

40063 socklen\_t dest\_len);

## 40064 DESCRIPTION

40065 The *sendto()* function shall send a message through a connection-mode or connectionless-mode  
 40066 socket. If the socket is connectionless-mode, the message shall be sent to the address specified by  
 40067 *dest\_addr*. If the socket is connection-mode, *dest\_addr* shall be ignored.

40068 The *sendto()* function takes the following arguments:

40069	<i>socket</i>	Specifies the socket file descriptor.
40070	<i>message</i>	Points to a buffer containing the message to be sent.
40071	<i>length</i>	Specifies the size of the message in bytes.
40072	<i>flags</i>	Specifies the type of message transmission. Values of this argument are 40073 formed by logically OR'ing zero or more of the following flags:
40074	MSG_EOR	Terminates a record (if supported by the protocol).
40075	MSG_OOB	Sends out-of-band data on sockets that support out-of-band 40076 data. The significance and semantics of out-of-band data are 40077 protocol-specific.
40078	<i>dest_addr</i>	Points to a <b>sockaddr</b> structure containing the destination address. The length 40079 and format of the address depend on the address family of the socket.
40080	<i>dest_len</i>	Specifies the length of the <b>sockaddr</b> structure pointed to by the <i>dest_addr</i> 40081 argument.

40082 If the socket protocol supports broadcast and the specified address is a broadcast address for the  
 40083 socket protocol, *sendto()* shall fail if the SO\_BROADCAST option is not set for the socket.

40084 The *dest\_addr* argument specifies the address of the target. The *length* argument specifies the  
 40085 length of the message.

40086 Successful completion of a call to *sendto()* does not guarantee delivery of the message. A return  
 40087 value of  $-1$  indicates only locally-detected errors.

40088 If space is not available at the sending socket to hold the message to be transmitted and the  
 40089 socket file descriptor does not have O\_NONBLOCK set, *sendto()* shall block until space is  
 40090 available. If space is not available at the sending socket to hold the message to be transmitted  
 40091 and the socket file descriptor does have O\_NONBLOCK set, *sendto()* shall fail.

40092 The socket in use may require the process to have appropriate privileges to use the *sendto()*  
 40093 function.

## 40094 RETURN VALUE

40095 Upon successful completion, *sendto()* shall return the number of bytes sent. Otherwise,  $-1$  shall  
 40096 be returned and *errno* set to indicate the error.

40097 **ERRORS**

- 40098       The *sendto()* function shall fail if:
- 40099       [EAFNOSUPPORT]
- 40100               Addresses in the specified address family cannot be used with this socket.
- 40101       [EAGAIN] or [EWOULDBLOCK]
- 40102               The socket's file descriptor is marked O\_NONBLOCK and the requested
- 40103               operation would block.
- 40104       [EBADF]       The *socket* argument is not a valid file descriptor.
- 40105       [ECONNRESET] A connection was forcibly closed by a peer.
- 40106       [EINTR]       A signal interrupted *sendto()* before any data was transmitted.
- 40107       [EMSGSIZE]    The message is too large to be sent all at once, as the socket requires.
- 40108       [ENOTCONN]    The socket is connection-mode but is not connected.
- 40109       [ENOTSOCK]    The *socket* argument does not refer to a socket.
- 40110       [EOPNOTSUPP] The *socket* argument is associated with a socket that does not support one or
- 40111               more of the values set in *flags*.
- 40112       [EPIPE]        The socket is shut down for writing, or the socket is connection-mode and is
- 40113               no longer connected. In the latter case, and if the socket is of type
- 40114               SOCK\_STREAM, the SIGPIPE signal is generated to the calling thread.
- 40115       If the address family of the socket is AF\_UNIX, then *sendto()* shall fail if:
- 40116       [EIO]         An I/O error occurred while reading from or writing to the file system.
- 40117       [ELOOP]        A loop exists in symbolic links encountered during resolution of the pathname |
- 40118               in the socket address. |
- 40119       [ENAMETOOLONG]
- 40120               A component of a pathname exceeded {NAME\_MAX} characters, or an entire |
- 40121               pathname exceeded {PATH\_MAX} characters. |
- 40122       [ENOENT]       A component of the pathname does not name an existing file or the pathname |
- 40123               is an empty string. |
- 40124       [ENOTDIR]     A component of the path prefix of the pathname in the socket address is not a |
- 40125               directory. |
- 40126       The *sendto()* function may fail if:
- 40127       [EACCES]        Search permission is denied for a component of the path prefix; or write
- 40128               access to the named socket is denied.
- 40129       [EDESTADDRREQ]
- 40130               The socket is not connection-mode and does not have its peer address set, and
- 40131               no destination address was specified.
- 40132       [EHOSTUNREACH]
- 40133               The destination host cannot be reached (probably because the host is down or
- 40134               a remote router cannot reach it).
- 40135       [EINVAL]        The *dest\_len* argument is not a valid length for the address family.
- 40136       [EIO]         An I/O error occurred while reading from or writing to the file system.

- 40137 [EISCONN] A destination address was specified and the socket is already connected. This  
40138 error may or may not be returned for connection mode sockets.
- 40139 [ENETDOWN] The local network interface used to reach the destination is down.
- 40140 [ENETUNREACH]  
40141 No route to the network is present.
- 40142 [ENOBUFS] Insufficient resources were available in the system to perform the operation.
- 40143 [ENOMEM] Insufficient memory was available to fulfill the request.
- 40144 If the address family of the socket is AF\_UNIX, then *sendto()* may fail if:
- 40145 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during |  
40146 resolution of the pathname in the socket address. |
- 40147 [ENAMETOOLONG]  
40148 Pathname resolution of a symbolic link produced an intermediate result |  
40149 whose length exceeds {PATH\_MAX}.
- 40150 **EXAMPLES**  
40151 None.
- 40152 **APPLICATION USAGE**  
40153 The *select()* and *poll()* functions can be used to determine when it is possible to send more data.
- 40154 **RATIONALE**  
40155 None.
- 40156 **FUTURE DIRECTIONS**  
40157 None.
- 40158 **SEE ALSO**  
40159 *getsockopt()*, *poll()*, *recv()*, *recvfrom()*, *recvmsg()*, *select()*, *send()*, *sendmsg()*, *setsockopt()*,  
40160 *shutdown()*, *socket()*, the Base Definitions volume of IEEE Std 1003.1-200x, <sys/socket.h>
- 40161 **CHANGE HISTORY**  
40162 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.
- 40163 The wording of the mandatory [ELOOP] error condition is updated, and a second optional  
40164 [ELOOP] error condition is added.

40165 **NAME**

40166        setbuf — assign buffering to a stream

40167 **SYNOPSIS**

40168        #include &lt;stdio.h&gt;

40169        void setbuf(FILE \*restrict stream, char \*restrict buf);

40170 **DESCRIPTION**

40171 cx        The functionality described on this reference page is aligned with the ISO C standard. Any  
40172        conflict between the requirements described here and the ISO C standard is unintentional. This  
40173        volume of IEEE Std 1003.1-200x defers to the ISO C standard.

40174        Except that it returns no value, the function call:

40175        setbuf(stream, buf)

40176        shall be equivalent to:

40177        setvbuf(stream, buf, \_IOFBF, BUFSIZ)

40178        if *buf* is not a null pointer, or to:

40179        setvbuf(stream, buf, \_IONBF, BUFSIZ)

40180        if *buf* is a null pointer.40181 **RETURN VALUE**40182        The *setbuf()* function shall not return a value.40183 **ERRORS**

40184        No errors are defined.

40185 **EXAMPLES**

40186        None.

40187 **APPLICATION USAGE**

40188        A common source of error is allocating buffer space as an “automatic” variable in a code block,  
40189        and then failing to close the stream in the same block.

40190        With *setbuf()*, allocating a buffer of BUFSIZ bytes does not necessarily imply that all of BUFSIZ  
40191        bytes are used for the buffer area.

40192 **RATIONALE**

40193        None.

40194 **FUTURE DIRECTIONS**

40195        None.

40196 **SEE ALSO**40197        *fopen()*, *setvbuf()*, the Base Definitions volume of IEEE Std 1003.1-200x, <stdio.h>40198 **CHANGE HISTORY**

40199        First released in Issue 1. Derived from Issue 1 of the SVID.

40200 **Issue 6**40201        The prototype for *setbuf()* is updated for alignment with the ISO/IEC 9899:1999 standard.

40202 **NAME**

40203           setcontext — set current user context

40204 **SYNOPSIS**

40205 xSI       #include <ucontext.h>

40206           int setcontext(const ucontext\_t \*ucp);

40207

40208 **DESCRIPTION**

40209           Refer to *getcontext()*.

40210 **NAME**

40211 setegid — set effective group ID

40212 **SYNOPSIS**

40213 #include &lt;unistd.h&gt;

40214 int setegid(gid\_t gid);

40215 **DESCRIPTION**

40216 If *gid* is equal to the real group ID or the saved set-group-ID, or if the process has appropriate  
40217 privileges, *setegid()* shall set the effective group ID of the calling process to *gid*; the real group  
40218 ID, saved set-group-ID, and any supplementary group IDs shall remain unchanged.

40219 The *setegid()* function shall not affect the supplementary group list in any way.

40220 **RETURN VALUE**

40221 Upon successful completion, 0 shall be returned; otherwise, -1 shall be returned and *errno* set to  
40222 indicate the error.

40223 **ERRORS**

40224 The *setegid()* function shall fail if:

40225 [EINVAL] The value of the *gid* argument is invalid and is not supported by the  
40226 implementation.

40227 [EPERM] The process does not have appropriate privileges and *gid* does not match the  
40228 real group ID or the saved set-group-ID.

40229 **EXAMPLES**

40230 None.

40231 **APPLICATION USAGE**

40232 None.

40233 **RATIONALE**40234 Refer to the RATIONALE section in *setuid()*.40235 **FUTURE DIRECTIONS**

40236 None.

40237 **SEE ALSO**

40238 *exec*, *getegid()*, *geteuid()*, *getgid()*, *getuid()*, *seteuid()*, *setgid()*, *setregid()*, *setreuid()*, *setuid()*, the  
40239 Base Definitions volume of IEEE Std 1003.1-200x, <sys/types.h>, <unistd.h>

40240 **CHANGE HISTORY**

40241 First released in Issue 6. Derived from the IEEE P1003.1a draft standard.

40242 **NAME**

40243        setenv — add or change environment variable

40244 **SYNOPSIS**

40245 cx        #include &lt;stdlib.h&gt;

40246        int setenv(const char \*envname, const char \*envval, int overwrite);

40247

40248 **DESCRIPTION**40249        The *setenv()* function shall update or add a variable in the environment of the calling process.40250        The *envname* argument points to a string containing the name of an environment variable to be40251        added or altered. The environment variable shall be set to the value to which *envval* points. The40252        function shall fail if *envname* points to a string which contains an '=' character. If the40253        environment variable named by *envname* already exists and the value of *overwrite* is non-zero,

40254        the function shall return success and the environment shall be updated. If the environment

40255        variable named by *envname* already exists and the value of *overwrite* is zero, the function shall

40256        return success and the environment shall remain unchanged.

40257        If the application modifies *environ* or the pointers to which it points, the behavior of *setenv()* is40258        undefined. The *setenv()* function shall update the list of pointers to which *environ* points.40259        The strings described by *envname* and *envval* are copied by this function.40260        The *setenv()* function need not be reentrant. A function that is not required to be reentrant is not

40261        required to be thread-safe.

40262 **RETURN VALUE**40263        Upon successful completion, zero shall be returned. Otherwise, -1 shall be returned, *errno* set to

40264        indicate the error, and the environment shall be unchanged.

40265 **ERRORS**40266        The *setenv()* function shall fail if:40267        [EINVAL]        The *name* argument is a null pointer, points to an empty string, or points to a  
40268        string containing an '=' character.40269        [ENOMEM]        Insufficient memory was available to add a variable or its value to the  
40270        environment.40271 **EXAMPLES**

40272        None.

40273 **APPLICATION USAGE**

40274        None.

40275 **RATIONALE**40276        Unanticipated results may occur if *setenv()* changes the external variable *environ*. In particular,40277        if the optional *envp* argument to *main()* is present, it is not changed, and thus may point to an40278        obsolete copy of the environment (as may any other copy of *environ*). However, other than the

40279        aforementioned restriction, the developers of IEEE Std 1003.1-200x intended that the traditional

40280        method of walking through the environment by way of the *environ* pointer must be supported.40281        It was decided that *setenv()* should be required by this revision because it addresses a piece of

40282        missing functionality, and does not impose a significant burden on the implementor.

40283        There was considerable debate as to whether the System V *putenv()* function or the BSD *setenv()*40284        function should be required as a mandatory function. The *setenv()* function was chosen because40285        it permitted the implementation of *unsetenv()* function to delete environmental variables,40286        without specifying an additional interface. The *putenv()* function is available as an XSI

40287 extension.

40288 The standard developers considered requiring that *setenv()* indicate an error when a call to it  
40289 would result in exceeding {ARG\_MAX}. The requirement was rejected since the condition might  
40290 be temporary, with the application eventually reducing the environment size. The ultimate  
40291 success or failure depends on the size at the time of a call to *exec*, which returns an indication of  
40292 this error condition.

40293 **FUTURE DIRECTIONS**

40294 None.

40295 **SEE ALSO**

40296 *getenv()*, *unsetenv()*, the Base Definitions volume of IEEE Std 1003.1-200x, `<stdlib.h>`,  
40297 `<sys/types.h>`, `<unistd.h>`

40298 **CHANGE HISTORY**

40299 First released in Issue 6. Derived from the IEEE P1003.1a draft standard.

40300 **NAME**

40301           seteuid — set effective user ID

40302 **SYNOPSIS**

40303           #include &lt;unistd.h&gt;

40304           int seteuid(uid\_t uid);

40305 **DESCRIPTION**

40306           If *uid* is equal to the real user ID or the saved set-user-ID, or if the process has appropriate  
40307           privileges, *seteuid()* shall set the effective user ID of the calling process to *uid*; the real user ID  
40308           and saved set-user-ID shall remain unchanged.

40309           The *seteuid()* function shall not affect the supplementary group list in any way.

40310 **RETURN VALUE**

40311           Upon successful completion, 0 shall be returned; otherwise, -1 shall be returned and *errno* set to  
40312           indicate the error.

40313 **ERRORS**

40314           The *seteuid()* function shall fail if:

40315           [EINVAL]           The value of the *uid* argument is invalid and is not supported by the  
40316           implementation.

40317           [EPERM]           The process does not have appropriate privileges and *uid* does not match the  
40318           real group ID or the saved set-group-ID.

40319 **EXAMPLES**

40320           None.

40321 **APPLICATION USAGE**

40322           None.

40323 **RATIONALE**

40324           Refer to the RATIONALE section in *setuid()*.

40325 **FUTURE DIRECTIONS**

40326           None.

40327 **SEE ALSO**

40328           *exec*, *getegid()*, *geteuid()*, *getgid()*, *getuid()*, *setegid()*, *setgid()*, *setregid()*, *setreuid()*, *setuid()*, the  
40329           Base Definitions volume of IEEE Std 1003.1-200x, <sys/types.h>, <unistd.h>

40330 **CHANGE HISTORY**

40331           First released in Issue 6. Derived from the IEEE P1003.1a draft standard.

40332 **NAME**

40333           setgid — set-group-ID

40334 **SYNOPSIS**

40335           #include &lt;unistd.h&gt;

40336           int setgid(gid\_t *gid*);40337 **DESCRIPTION**40338           If the process has appropriate privileges, *setgid()* shall set the real group ID, effective group ID, and the saved set-group-ID of the calling process to *gid*.40340           If the process does not have appropriate privileges, but *gid* is equal to the real group ID or the saved set-group-ID, *setgid()* shall set the effective group ID to *gid*; the real group ID and saved set-group-ID shall remain unchanged.40343           The *setgid()* function shall not affect the supplementary group list in any way.

40344           Any supplementary group IDs of the calling process shall remain unchanged.

40345 **RETURN VALUE**40346           Upon successful completion, 0 is returned. Otherwise, -1 shall be returned and *errno* set to indicate the error.40348 **ERRORS**40349           The *setgid()* function shall fail if:40350           [EINVAL]           The value of the *gid* argument is invalid and is not supported by the implementation.40352           [EPERM]           The process does not have appropriate privileges and *gid* does not match the real group ID or the saved set-group-ID.40354 **EXAMPLES**

40355           None.

40356 **APPLICATION USAGE**

40357           None.

40358 **RATIONALE**40359           Refer to the RATIONALE section in *setuid()*.40360 **FUTURE DIRECTIONS**

40361           None.

40362 **SEE ALSO**40363           *exec*, *getegid()*, *geteuid()*, *getgid()*, *getuid()*, *setegid()*, *seteuid()*, *setregid()*, *setreuid()*, *setuid()*, the Base Definitions volume of IEEE Std 1003.1-200x, <sys/types.h>, <unistd.h>40365 **CHANGE HISTORY**

40366           First released in Issue 1. Derived from Issue 1 of the SVID.

40367 **Issue 6**

40368           In the SYNOPSIS, the optional include of the &lt;sys/types.h&gt; header is removed.

40369           The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- 40371           • The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.

- 40374 • Functionality associated with `_POSIX_SAVED_IDS` is now mandated. This is a FIPS  
40375 requirement.

40376 The following changes were made to align with the IEEE P1003.1a draft standard:

- 40377 • The effects of `setgid()` in processes without appropriate privileges are changed
- 40378 • A requirement that the supplementary group list is not affected is added.

40379 **NAME**

40380 setgrent — reset group database to first entry

40381 **SYNOPSIS**

40382 xSI #include &lt;grp.h&gt;

40383 void setgrent(void);

40384

40385 **DESCRIPTION**40386 Refer to *endgrent()*.

40387 **NAME**

40388           sethostent — network host database functions

40389 **SYNOPSIS**

40390           #include <netdb.h>

40391           void sethostent(int *stayopen*);

40392 **DESCRIPTION**

40393           Refer to *endhostent()*.

40394 **NAME**

40395 setitimer — set value of interval timer

40396 **SYNOPSIS**

40397 xSI #include &lt;sys/time.h&gt;

40398 int setitimer(int *which*, const struct itimerval \*restrict *value*,  
40399 struct itimerval \*restrict *ovalue*);

40400

40401 **DESCRIPTION**40402 Refer to *getitimer()*.

40403 **NAME**

40404 setjmp — set jump point for a non-local goto

40405 **SYNOPSIS**

40406 #include &lt;setjmp.h&gt;

40407 int setjmp(jmp\_buf env);

40408 **DESCRIPTION**

40409 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
40410 conflict between the requirements described here and the ISO C standard is unintentional. This  
40411 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

40412 A call to *setjmp()*, shall save the calling environment in its *env* argument for later use by  
40413 *longjmp()*.

40414 It is unspecified whether *setjmp()* is a macro or a function. If a macro definition is suppressed in  
40415 order to access an actual function, or a program defines an external identifier with the name  
40416 *setjmp*, the behavior is undefined.

40417 An application shall ensure that an invocation of *setjmp()* appears in one of the following  
40418 contexts only:

- 40419 • The entire controlling expression of a selection or iteration statement
- 40420 • One operand of a relational or equality operator with the other operand an integral constant  
40421 expression, with the resulting expression being the entire controlling expression of a  
40422 selection or iteration statement
- 40423 • The operand of a unary '!' operator with the resulting expression being the entire  
40424 controlling expression of a selection or iteration
- 40425 • The entire expression of an expression statement (possibly cast to **void**)

40426 If the invocation appears in any other context, the behavior is undefined.

40427 **RETURN VALUE**

40428 If the return is from a direct invocation, *setjmp()* shall return 0. If the return is from a call to  
40429 *longjmp()*, *setjmp()* shall return a non-zero value.

40430 **ERRORS**

40431 No errors are defined.

40432 **EXAMPLES**

40433 None.

40434 **APPLICATION USAGE**

40435 In general, *sigsetjmp()* is more useful in dealing with errors and interrupts encountered in a low-  
40436 level subroutine of a program.

40437 **RATIONALE**

40438 None.

40439 **FUTURE DIRECTIONS**

40440 None.

40441 **SEE ALSO**

40442 *longjmp()*, *sigsetjmp()*, the Base Definitions volume of IEEE Std 1003.1-200x, <setjmp.h>

40443 **CHANGE HISTORY**

40444 First released in Issue 1. Derived from Issue 1 of the SVID.

40445 **Issue 6**

40446 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

40447 **NAME**40448 setkey — set encoding key (**CRYPT**)40449 **SYNOPSIS**40450 XSI `#include <stdlib.h>`40451 `void setkey(const char *key);`

40452

40453 **DESCRIPTION**

40454 The *setkey()* function provides access to an implementation-defined encoding algorithm. The  
40455 argument of *setkey()* is an array of length 64 bytes containing only the bytes with numerical  
40456 value of 0 and 1. If this string is divided into groups of 8, the low-order bit in each group is  
40457 ignored; this gives a 56-bit key which is used by the algorithm. This is the key that shall be used  
40458 with the algorithm to encode a string *block* passed to *encrypt()*.

40459 The *setkey()* function shall not change the setting of *errno* if successful. An application wishing to  
40460 check for error situations should set *errno* to 0 before calling *setkey()*. If *errno* is non-zero on  
40461 return, an error has occurred.

40462 The *setkey()* function need not be reentrant. A function that is not required to be reentrant is not  
40463 required to be thread-safe.

40464 **RETURN VALUE**

40465 No values are returned.

40466 **ERRORS**40467 The *setkey()* function shall fail if:

40468 [ENOSYS] The functionality is not supported on this implementation.

40469 **EXAMPLES**

40470 None.

40471 **APPLICATION USAGE**

40472 Decoding need not be implemented in all environments. This is related to government  
40473 restrictions in some countries on encryption and decryption routines. Historical practice has  
40474 been to ship a different version of the encryption library without the decryption feature in the  
40475 routines supplied. Thus the exported version of *encrypt()* does encoding but not decoding.

40476 **RATIONALE**

40477 None.

40478 **FUTURE DIRECTIONS**

40479 None.

40480 **SEE ALSO**40481 *crypt()*, *encrypt()*, the Base Definitions volume of IEEE Std 1003.1-200x, `<stdlib.h>`40482 **CHANGE HISTORY**

40483 First released in Issue 1. Derived from Issue 1 of the SVID.

40484 **Issue 5**40485 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.

## 40486 NAME

40487 setlocale — set program locale

## 40488 SYNOPSIS

40489 #include &lt;locale.h&gt;

40490 char \*setlocale(int *category*, const char \**locale*);

## 40491 DESCRIPTION

40492 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 40493 conflict between the requirements described here and the ISO C standard is unintentional. This  
 40494 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

40495 The *setlocale()* function selects the appropriate piece of the program's locale, as specified by the  
 40496 *category* and *locale* arguments, and may be used to change or query the program's entire locale or  
 40497 portions thereof. The value *LC\_ALL* for *category* names the program's entire locale; other values  
 40498 for *category* name only a part of the program's locale:

40499 *LC\_COLLATE* Affects the behavior of regular expressions and the collation functions.

40500 *LC\_CTYPE* Affects the behavior of regular expressions, character classification, character  
 40501 conversion functions, and wide-character functions.

40502 CX *LC\_MESSAGES* Affects what strings are expected by commands and utilities as affirmative or  
 40503 negative responses.

40504 XSI It also affects what strings are given by commands and utilities as affirmative  
 40505 or negative responses, and the content of messages.

40506 *LC\_MONETARY* Affects the behavior of functions that handle monetary values.

40507 *LC\_NUMERIC* Affects the behavior of functions that handle numeric values.

40508 *LC\_TIME* Affects the behavior of the time conversion functions.

40509 The *locale* argument is a pointer to a character string containing the required setting of *category*.  
 40510 The contents of this string are implementation-defined. In addition, the following preset values  
 40511 of *locale* are defined for all settings of *category*:

40512 CX "POSIX" Specifies the minimal environment for C-language translation called POSIX  
 40513 locale. If *setlocale()* is not invoked, the POSIX locale is the default at entry to  
 40514 *main()*.

40515 "C" Equivalent to "POSIX".

40516 CX "" Specifies an implementation-defined native environment. This corresponds to  
 40517 the value of the associated environment variables, *LC\_\** and *LANG*; see the  
 40518 Base Definitions volume of IEEE Std 1003.1-200x, Chapter 7, Locale and the  
 40519 Base Definitions volume of IEEE Std 1003.1-200x, Chapter 8, Environment  
 40520 Variables.

40521 A null pointer Used to direct *setlocale()* to query the current internationalized environment  
 40522 and return the name of the *locale*.

40523 THR The locale state is common to all threads within a process.

## 40524 RETURN VALUE

40525 Upon successful completion, *setlocale()* shall return the string associated with the specified  
 40526 category for the new locale. Otherwise, *setlocale()* shall return a null pointer and the program's  
 40527 locale is not changed.

40528 A null pointer for *locale* causes *setlocale()* to return a pointer to the string associated with the  
 40529 *category* for the program's current locale. The program's locale shall not be changed.

40530 The string returned by *setlocale()* is such that a subsequent call with that string and its associated  
 40531 *category* shall restore that part of the program's locale. The application shall not modify the string  
 40532 returned which may be overwritten by a subsequent call to *setlocale()*.

#### 40533 ERRORS

40534 No errors are defined.

#### 40535 EXAMPLES

40536 None.

#### 40537 APPLICATION USAGE

40538 The following code illustrates how a program can initialize the international environment for  
 40539 one language, while selectively modifying the program's locale such that regular expressions  
 40540 and string operations can be applied to text recorded in a different language:

```
40541 setlocale(LC_ALL, "De");
40542 setlocale(LC_COLLATE, "Fr@dict");
```

40543 Internationalized programs must call *setlocale()* to initiate a specific language operation. This can  
 40544 be done by calling *setlocale()* as follows:

```
40545 setlocale(LC_ALL, "");
```

40546 Changing the setting of *LC\_MESSAGES* has no effect on catalogs that have already been opened  
 40547 by calls to *catopen()*.

#### 40548 RATIONALE

40549 The ISO C standard defines a collection of functions to support internationalization. One of the  
 40550 most significant aspects of these functions is a facility to set and query the *international*  
 40551 *environment*. The international environment is a repository of information that affects the  
 40552 behavior of certain functionality, namely:

- 40553 1. Character handling
- 40554 2. Collating
- 40555 3. Date/time formatting
- 40556 4. Numeric editing
- 40557 5. Monetary formatting
- 40558 6. Messaging

40559 The *setlocale()* function provides the application developer with the ability to set all or portions,  
 40560 called *categories*, of the international environment. These categories correspond to the areas of  
 40561 functionality, mentioned above. The syntax for *setlocale()* is as follows:

```
40562 char *setlocale(int category, const char *locale);
```

40563 where *category* is the name of one of following categories, namely:

```
40564     LC_COLLATE
40565     LC_CTYPE
40566     LC_MESSAGES
40567     LC_MONETARY
40568     LC_NUMERIC
40569     LC_TIME
```

40570 In addition, a special value called *LC\_ALL* directs *setlocale()* to set all categories.

40571 There are two primary uses of *setlocale()*:

- 40572 1. Querying the international environment to find out what it is set to
- 40573 2. Setting the international environment, or *locale*, to a specific value

40574 The behavior of *setlocale()* in these two areas is described below. Since it is difficult to describe  
40575 the behavior in words, examples are used to illustrate the behavior of specific uses.

40576 To query the international environment, *setlocale()* is invoked with a specific category and the  
40577 NULL pointer as the locale. The NULL pointer is a special directive to *setlocale()* that tells it to  
40578 query rather than set the international environment. The following syntax is used to query the  
40579 name of the international environment:

```
40580 setlocale({LC_ALL, LC_COLLATE, LC_CTYPE, LC_MESSAGES, LC_MONETARY, \
40581          LC_NUMERIC, LC_TIME}, (char *) NULL);
```

40582 The *setlocale()* function shall return the string corresponding to the current international  
40583 environment. This value may be used by a subsequent call to *setlocale()* to reset the international  
40584 environment to this value. However, it should be noted that the return value from *setlocale()*  
40585 may be a pointer to a static area within the function and is not guaranteed to remain unchanged  
40586 (that is, it may be modified by a subsequent call to *setlocale()*). Therefore, if the purpose of  
40587 calling *setlocale()* is to save the value of the current international environment so it can be  
40588 changed and reset later, the return value should be copied to an array of **char** in the calling  
40589 program.

40590 There are three ways to set the international environment with *setlocale()*:

40591 *setlocale(category, string)*

40592 This usage sets a specific *category* in the international environment to a specific value  
40593 corresponding to the value of the *string*. A specific example is provided below:

```
40594 setlocale(LC_ALL, "fr_FR.ISO-8859-1");
```

40595 In this example, all categories of the international environment are set to the locale  
40596 corresponding to the string "fr\_FR.ISO-8859-1", or to the French language as spoken in  
40597 France using the ISO/IEC 8859-1:1998 standard codeset.

40598 If the string does not correspond to a valid locale, *setlocale()* shall return a NULL pointer  
40599 and the international environment is not changed. Otherwise, *setlocale()* shall return the  
40600 name of the locale just set.

40601 *setlocale(category, "C")*

40602 The ISO C standard states that one locale must exist on all conforming implementations.  
40603 The name of the locale is C and corresponds to a minimal international environment needed  
40604 to support the C programming language.

40605 *setlocale(category, "")*

40606 This sets a specific category to an implementation-defined default. This corresponds to the  
40607 value of the environment variables.

#### 40608 FUTURE DIRECTIONS

40609 None.

#### 40610 SEE ALSO

40611 *exec*, *isalnum()*, *isalpha()*, *isblank()*, *iscntrl()*, *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*,  
40612 *isspace()*, *isupper()*, *iswalnum()*, *iswalpha()*, *iswblank()*, *iswcntrl()*, *iswctype()*, *iswdigit()*,  
40613 *iswgraph()*, *iswlower()*, *iswprint()*, *iswpunct()*, *iswspace()*, *iswupper()*, *iswxdigit()*, *isxdigit()*,

40614 *localeconv()*, *mblen()*, *mbstowcs()*, *mbtowc()*, *nl\_langinfo()*, *printf()*, *scanf()*, *setlocale()*, *strcoll()*,  
40615 *strerror()*, *strfmon()*, *strtod()*, *strxfrm()*, *tolower()*, *toupper()*, *towlower()*, *towupper()*, *wscoll()*,  
40616 *wctod()*, *wcstombs()*, *wcsxfrm()*, *wctomb()*, the Base Definitions volume of IEEE Std 1003.1-200x,  
40617 **<langinfo.h>**, **<locale.h>**

40618 **CHANGE HISTORY**

40619 First released in Issue 3.

40620 **Issue 5**

40621 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

40622 **Issue 6**

40623 Extensions beyond the ISO C standard are now marked.

40624 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

40625 **NAME**

40626 setlogmask — set log priority mask

40627 **SYNOPSIS**40628 XSI `#include <syslog.h>`40629 `int setlogmask(int maskpri);`

40630

40631 **DESCRIPTION**40632 Refer to *closelog()*.

## setnetent()

40633 **NAME**

40634           setnetent — network database function

40635 **SYNOPSIS**

40636           #include <netdb.h>

40637           void setnetent(int stayopen);

40638 **DESCRIPTION**

40639           Refer to *endnetent()*.

40640 **NAME**

40641 setpgid — set process group ID for job control

40642 **SYNOPSIS**

40643 #include &lt;unistd.h&gt;

40644 int setpgid(pid\_t pid, pid\_t pgid);

40645 **DESCRIPTION**

40646 The *setpgid()* function shall either join an existing process group or create a new process group  
 40647 within the session of the calling process. The process group ID of a session leader shall not  
 40648 change. Upon successful completion, the process group ID of the process with a process ID that  
 40649 matches *pid* shall be set to *pgid*. As a special case, if *pid* is 0, the process ID of the calling process  
 40650 shall be used. Also, if *pgid* is 0, the process group ID of the indicated process shall be used.

40651 **RETURN VALUE**

40652 Upon successful completion, *setpgid()* shall return 0; otherwise, -1 shall be returned and *errno*  
 40653 shall be set to indicate the error.

40654 **ERRORS**40655 The *setpgid()* function shall fail if:

40656 [EACCES] The value of the *pid* argument matches the process ID of a child process of the  
 40657 calling process and the child process has successfully executed one of the *exec*  
 40658 functions.

40659 [EINVAL] The value of the *pgid* argument is less than 0, or is not a value supported by  
 40660 the implementation.

40661 [EPERM] The process indicated by the *pid* argument is a session leader.

40662 [EPERM] The value of the *pid* argument matches the process ID of a child process of the  
 40663 calling process and the child process is not in the same session as the calling  
 40664 process.

40665 [EPERM] The value of the *pgid* argument is valid but does not match the process ID of  
 40666 the process indicated by the *pid* argument and there is no process with a  
 40667 process group ID that matches the value of the *pgid* argument in the same  
 40668 session as the calling process.

40669 [ESRCH] The value of the *pid* argument does not match the process ID of the calling  
 40670 process or of a child process of the calling process.

40671 **EXAMPLES**

40672 None.

40673 **APPLICATION USAGE**

40674 None.

40675 **RATIONALE**

40676 The *setpgid()* function shall group processes together for the purpose of signaling, placement in  
 40677 foreground or background, and other job control actions.

40678 The *setpgid()* function is similar to the *setpgrp()* function of 4.2 BSD, except that 4.2 BSD allowed  
 40679 the specified new process group to assume any value. This presents certain security problems  
 40680 and is more flexible than necessary to support job control.

40681 To provide tighter security, *setpgid()* only allows the calling process to join a process group  
 40682 already in use inside its session or create a new process group whose process group ID was  
 40683 equal to its process ID.

40684 When a job control shell spawns a new job, the processes in the job must be placed into a new  
40685 process group via *setpgid()*. There are two timing constraints involved in this action:

- 40686 1. The new process must be placed in the new process group before the appropriate program  
40687 is launched via one of the *exec* functions.
- 40688 2. The new process must be placed in the new process group before the shell can correctly  
40689 send signals to the new process group.

40690 To address these constraints, the following actions are performed. The new processes call  
40691 *setpgid()* to alter their own process groups after *fork()* but before *exec*. This satisfies the first  
40692 constraint. Under 4.3 BSD, the second constraint is satisfied by the synchronization property of  
40693 *vfork()*; that is, the shell is suspended until the child has completed the *exec*, thus ensuring that  
40694 the child has completed the *setpgid()*. A new version of *fork()* with this same synchronization  
40695 property was considered, but it was decided instead to merely allow the parent shell process to  
40696 adjust the process group of its child processes via *setpgid()*. Both timing constraints are now  
40697 satisfied by having both the parent shell and the child attempt to adjust the process group of the  
40698 child process; it does not matter which succeeds first.

40699 Since it would be confusing to an application to have its process group change after it began  
40700 executing (that is, after *exec*), and because the child process would already have adjusted its  
40701 process group before this, the [EACCES] error was added to disallow this.

40702 One non-obvious use of *setpgid()* is to allow a job control shell to return itself to its original  
40703 process group (the one in effect when the job control shell was executed). A job control shell  
40704 does this before returning control back to its parent when it is terminating or suspending itself as  
40705 a way of restoring its job control “state” back to what its parent would expect. (Note that the  
40706 original process group of the job control shell typically matches the process group of its parent,  
40707 but this is not necessarily always the case.)

#### 40708 FUTURE DIRECTIONS

40709 None.

#### 40710 SEE ALSO

40711 *exec*, *getpgrp()*, *setsid()*, *tcsetpgrp()*, the Base Definitions volume of IEEE Std 1003.1-200x,  
40712 <sys/types.h>, <unistd.h>

#### 40713 CHANGE HISTORY

40714 First released in Issue 3.

40715 Entry included for alignment with the POSIX.1-1988 standard.

#### 40716 Issue 6

40717 In the SYNOPSIS, the optional include of the <sys/types.h> header is removed.

40718 The following new requirements on POSIX implementations derive from alignment with the  
40719 Single UNIX Specification:

- 40720 • The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was  
40721 required for conforming implementations of previous POSIX specifications, it was not  
40722 required for UNIX applications.
- 40723 • The *setpgid()* function is mandatory since `_POSIX_JOB_CONTROL` is required to be defined  
40724 in this issue. This is a FIPS requirement.

40725 **NAME**

40726           setpgrp — set process group ID

40727 **SYNOPSIS**

40728 XSI       #include &lt;unistd.h&gt;

40729           pid\_t setpgrp(void);

40730

40731 **DESCRIPTION**

40732           If the calling process is not already a session leader, *setpgrp()* sets the process group ID of the  
40733           calling process to the process ID of the calling process. If *setpgrp()* creates a new session, then  
40734           the new session has no controlling terminal.

40735           The *setpgrp()* function has no effect when the calling process is a session leader.

40736 **RETURN VALUE**40737           Upon completion, *setpgrp()* shall return the process group ID.40738 **ERRORS**

40739           No errors are defined.

40740 **EXAMPLES**

40741           None.

40742 **APPLICATION USAGE**

40743           None.

40744 **RATIONALE**

40745           None.

40746 **FUTURE DIRECTIONS**

40747           None.

40748 **SEE ALSO**

40749           *exec*, *fork()*, *getpid()*, *getsid()*, *kill()*, *setpgid()*, *setsid()*, the Base Definitions volume of  
40750           IEEE Std 1003.1-200x, <unistd.h>

40751 **CHANGE HISTORY**

40752           First released in Issue 4, Version 2.

40753 **Issue 5**

40754           Moved from X/OPEN UNIX extension to BASE.

40755 **NAME**

40756           setpriority — set the nice value

40757 **SYNOPSIS**

40758 XSI       #include <sys/resource.h>

40759           int setpriority(int *which*, id\_t *who*, int *nice*);

40760

40761 **DESCRIPTION**

40762           Refer to *getpriority*().

40763 **NAME**

40764           setprotoent — network protocol database functions

40765 **SYNOPSIS**

40766           #include <netdb.h>

40767           void setprotoent(int *stayopen*);

40768 **DESCRIPTION**

40769           Refer to *endprotoent()*.

40770 **NAME**

40771           setpwent — user database function

40772 **SYNOPSIS**

40773 XSI       #include <pwd.h>

40774           void setpwent(void);

40775

40776 **DESCRIPTION**

40777           Refer to *endpwent()*.

40778 **NAME**

40779 setregid — set real and effective group IDs

40780 **SYNOPSIS**

40781 XSI #include &lt;unistd.h&gt;

40782 int setregid(gid\_t rgid, gid\_t egid);

40783

40784 **DESCRIPTION**40785 The *setregid()* function shall set the real and effective group IDs of the calling process.40786 If *rgid* is  $-1$ , the real group ID shall not be changed; if *egid* is  $-1$ , the effective group ID shall not  
40787 be changed.

40788 The real and effective group IDs may be set to different values in the same call.

40789 Only a process with appropriate privileges can set the real group ID and the effective group ID  
40790 to any valid value.40791 A non-privileged process can set either the real group ID to the saved set-group-ID from one of  
40792 the *exec* family of functions, or the effective group ID to the saved set-group-ID or the real group  
40793 ID.

40794 Any supplementary group IDs of the calling process remain unchanged.

40795 **RETURN VALUE**40796 Upon successful completion, 0 shall be returned. Otherwise,  $-1$  shall be returned and *errno* set to  
40797 indicate the error, and neither of the group IDs are changed.40798 **ERRORS**40799 The *setregid()* function shall fail if:40800 [EINVAL] The value of the *rgid* or *egid* argument is invalid or out-of-range.40801 [EPERM] The process does not have appropriate privileges and a change other than  
40802 changing the real group ID to the saved set-group-ID, or changing the  
40803 effective group ID to the real group ID or the saved set-group-ID, was  
40804 requested.40805 **EXAMPLES**

40806 None.

40807 **APPLICATION USAGE**40808 If a set-group-ID process sets its effective group ID to its real group ID, it can still set its effective  
40809 group ID back to the saved set-group-ID.40810 **RATIONALE**

40811 None.

40812 **FUTURE DIRECTIONS**

40813 None.

40814 **SEE ALSO**40815 *exec*, *getegid()*, *geteuid()*, *getgid()*, *getuid()*, *setegid()*, *seteuid()*, *setgid()*, *setreuid()*, *setuid()*, the  
40816 Base Definitions volume of IEEE Std 1003.1-200x, <unistd.h>40817 **CHANGE HISTORY**

40818 First released in Issue 4, Version 2.

40819 **Issue 5**

40820 Moved from X/OPEN UNIX extension to BASE.

40821 The DESCRIPTION is updated to indicate that the saved set-group-ID can be set by any of the  
40822 *exec* family of functions, not just *execev()*.

40823 **NAME**

40824 setreuid — set real and effective user IDs

40825 **SYNOPSIS**40826 XSI `#include <unistd.h>`40827 `int setreuid(uid_t ruid, uid_t euid);`

40828

40829 **DESCRIPTION**

40830 The `setreuid()` function shall set the real and effective user IDs of the current process to the  
 40831 values specified by the `ruid` and `euid` arguments. If `ruid` or `euid` is `-1`, the corresponding effective  
 40832 or real user ID of the current process shall be left unchanged.

40833 A process with appropriate privileges can set either ID to any value. An unprivileged process  
 40834 can only set the effective user ID if the `euid` argument is equal to either the real, effective, or  
 40835 saved user ID of the process.

40836 It is unspecified whether a process without appropriate privileges is permitted to change the real  
 40837 user ID to match the current real, effective, or saved set-user-ID of the process.

40838 **RETURN VALUE**

40839 Upon successful completion, 0 shall be returned. Otherwise, `-1` shall be returned and `errno` set to  
 40840 indicate the error.

40841 **ERRORS**40842 The `setreuid()` function shall fail if:40843 [EINVAL] The value of the `ruid` or `euid` argument is invalid or out-of-range.

40844 [EPERM] The current process does not have appropriate privileges, and either an  
 40845 attempt was made to change the effective user ID to a value other than the  
 40846 real user ID or the saved set-user-ID or an attempt was made to change the  
 40847 real user ID to a value not permitted by the implementation.

40848 **EXAMPLES**40849 **Setting the Effective User ID to the Real User ID**

40850 The following example sets the effective user ID of the calling process to the real user ID, so that  
 40851 files created later will be owned by the current user.

```
40852 #include <unistd.h>
40853 #include <sys/types.h>
40854 ...
40855 setreuid(getuid(), getuid());
40856 ...
```

40857 **APPLICATION USAGE**

40858 None.

40859 **RATIONALE**

40860 None.

40861 **FUTURE DIRECTIONS**

40862 None.

40863 **SEE ALSO**

40864 *getegid()*, *geteuid()*, *getgid()*, *getuid()*, *setegid()*, *seteuid()*, *setgid()*, *setregid()*, *setuid()*, the Base  
40865 Definitions volume of IEEE Std 1003.1-200x, <**unistd.h**>

40866 **CHANGE HISTORY**

40867 First released in Issue 4, Version 2.

40868 **Issue 5**

40869 Moved from X/OPEN UNIX extension to BASE.

40870 **NAME**

40871 setrlimit — control maximum resource consumption

40872 **SYNOPSIS**

40873 XSI #include &lt;sys/resource.h&gt;

40874 int setrlimit(int resource, const struct rlimit \*rlp);

40875

40876 **DESCRIPTION**40877 Refer to *getrlimit()*.

40878 **NAME**

40879           setservent — network services database functions

40880 **SYNOPSIS**

40881           #include <netdb.h>

40882           void setservent(int *stayopen*);

40883 **DESCRIPTION**

40884           Refer to *endservent()*.

40885 **NAME**

40886 setsid — create session and set process group ID

40887 **SYNOPSIS**

40888 #include &lt;unistd.h&gt;

40889 pid\_t setsid(void);

40890 **DESCRIPTION**

40891 The *setsid()* function shall create a new session, if the calling process is not a process group  
40892 leader. Upon return the calling process shall be the session leader of this new session, shall be  
40893 the process group leader of a new process group, and shall have no controlling terminal. The  
40894 process group ID of the calling process shall be set equal to the process ID of the calling process.  
40895 The calling process shall be the only process in the new process group and the only process in  
40896 the new session.

40897 **RETURN VALUE**

40898 Upon successful completion, *setsid()* shall return the value of the new process group ID of the  
40899 calling process. Otherwise, it shall return (**pid\_t**)-1 and set *errno* to indicate the error.

40900 **ERRORS**40901 The *setsid()* function shall fail if:

40902 [EPERM] The calling process is already a process group leader, or the process group ID  
40903 of a process other than the calling process matches the process ID of the  
40904 calling process.

40905 **EXAMPLES**

40906 None.

40907 **APPLICATION USAGE**

40908 None.

40909 **RATIONALE**

40910 The *setsid()* function is similar to the *setpgrp()* function of System V. System V, without job  
40911 control, groups processes into process groups and creates new process groups via *setpgrp()*; only  
40912 one process group may be part of a login session.

40913 Job control allows multiple process groups within a login session. In order to limit job control  
40914 actions so that they can only affect processes in the same login session, this volume of  
40915 IEEE Std 1003.1-200x adds the concept of a session that is created via *setsid()*. The *setsid()*  
40916 function also creates the initial process group contained in the session. Additional process  
40917 groups can be created via the *setpgid()* function. A System V process group would correspond to  
40918 a POSIX System Interfaces session containing a single POSIX process group. Note that this  
40919 function requires that the calling process not be a process group leader. The usual way to ensure  
40920 this is true is to create a new process with *fork()* and have it call *setsid()*. The *fork()* function  
40921 guarantees that the process ID of the new process does not match any existing process group ID.

40922 **FUTURE DIRECTIONS**

40923 None.

40924 **SEE ALSO**

40925 *getsid()*, *setpgid()*, *setpgrp()*, the Base Definitions volume of IEEE Std 1003.1-200x, <sys/types.h>,  
40926 <unistd.h>

40927 **CHANGE HISTORY**

40928 First released in Issue 3.

40929 Entry included for alignment with the POSIX.1-1988 standard.

40930 **Issue 6**40931 In the SYNOPSIS, the optional include of the `<sys/types.h>` header is removed.40932 The following new requirements on POSIX implementations derive from alignment with the  
40933 Single UNIX Specification:

- 40934
- The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was  
40935 required for conforming implementations of previous POSIX specifications, it was not  
40936 required for UNIX applications.

## 40937 NAME

40938 setsockopt — set the socket options

## 40939 SYNOPSIS

40940 #include &lt;sys/socket.h&gt;

```
40941 int setsockopt(int socket, int level, int option_name,
40942               const void *option_value, socklen_t option_len);
```

## 40943 DESCRIPTION

40944 The *setsockopt()* function shall set the option specified by the *option\_name* argument, at the |  
 40945 protocol level specified by the *level* argument, to the value pointed to by the *option\_value* |  
 40946 argument for the socket associated with the file descriptor specified by the *socket* argument.

40947 The *level* argument specifies the protocol level at which the option resides. To set options at the |  
 40948 socket level, specify the *level* argument as SOL\_SOCKET. To set options at other levels, supply |  
 40949 the appropriate *level* identifier for the protocol controlling the option. For example, to indicate |  
 40950 that an option is interpreted by the TCP (Transport Control Protocol), set *level* to IPPROTO\_TCP |  
 40951 as defined in the <netinet/in.h> header.

40952 The *option\_name* argument specifies a single option to set. The *option\_name* argument and any |  
 40953 specified options are passed uninterpreted to the appropriate protocol module for |  
 40954 interpretations. The <sys/socket.h> header defines the socket-level options. The options are as |  
 40955 follows:

40956 SO\_DEBUG Turns on recording of debugging information. This option enables or |  
 40957 disables debugging in the underlying protocol modules. This option takes |  
 40958 an **int** value. This is a Boolean option.

40959 SO\_BROADCAST Permits sending of broadcast messages, if this is supported by the |  
 40960 protocol. This option takes an **int** value. This is a Boolean option.

40961 SO\_REUSEADDR Specifies that the rules used in validating addresses supplied to *bind()* |  
 40962 should allow reuse of local addresses, if this is supported by the protocol. |  
 40963 This option takes an **int** value. This is a Boolean option.

40964 SO\_KEEPALIVE Keeps connections active by enabling the periodic transmission of |  
 40965 messages, if this is supported by the protocol. This option takes an **int** |  
 40966 value.

40967 If the connected socket fails to respond to these messages, the connection |  
 40968 is broken and threads writing to that socket are notified with a SIGPIPE |  
 40969 signal.

40970 This is a Boolean option.

40971 SO\_LINGER Lingers on a *close()* if data is present. This option controls the action |  
 40972 taken when unsent messages queue on a socket and *close()* is performed. |  
 40973 If SO\_LINGER is set, the system shall block the process during *close()* |  
 40974 until it can transmit the data or until the time expires. If SO\_LINGER is |  
 40975 not specified, and *close()* is issued, the system handles the call in a way |  
 40976 that allows the process to continue as quickly as possible. This option |  
 40977 takes a **linger** structure, as defined in the <sys/socket.h> header, to |  
 40978 specify the state of the option and linger interval.

40979 SO\_OOBINLINE Leaves received out-of-band data (data marked urgent) inline. This |  
 40980 option takes an **int** value. This is a Boolean option.

40981	SO_SNDBUF	Sets send buffer size. This option takes an <b>int</b> value.
40982	SO_RCVBUF	Sets receive buffer size. This option takes an <b>int</b> value.
40983	SO_DONTROUTE	Requests that outgoing messages bypass the standard routing facilities. The destination shall be on a directly-connected network, and messages are directed to the appropriate network interface according to the destination address. The effect, if any, of this option depends on what protocol is in use. This option takes an <b>int</b> value. This is a Boolean option.
40984		
40985		
40986		
40987		
40988	SO_RCVLOWAT	Sets the minimum number of bytes to process for socket input operations. The default value for SO_RCVLOWAT is 1. If SO_RCVLOWAT is set to a larger value, blocking receive calls normally wait until they have received the smaller of the low water mark value or the requested amount. (They may return less than the low water mark if an error occurs, a signal is caught, or the type of data next in the receive queue is different from that returned; for example, out-of-band data.) This option takes an <b>int</b> value. Note that not all implementations allow this option to be set.
40989		
40990		
40991		
40992		
40993		
40994		
40995		
40996	SO_RCVTIMEO	Sets the timeout value that specifies the maximum amount of time an input function waits until it completes. It accepts a <b>timeval</b> structure with the number of seconds and microseconds specifying the limit on how long to wait for an input operation to complete. If a receive operation has blocked for this much time without receiving additional data, it shall return with a partial count or <i>errno</i> set to [EAGAIN] or [EWOULDBLOCK] if no data is received. The default for this option is zero, which indicates that a receive operation shall not time out. This option takes a <b>timeval</b> structure. Note that not all implementations allow this option to be set.
40997		
40998		
40999		
41000		
41001		
41002		
41003		
41004		
41005		
41006	SO_SNDLOWAT	Sets the minimum number of bytes to process for socket output operations. Non-blocking output operations shall process no data if flow control does not allow the smaller of the send low water mark value or the entire request to be processed. This option takes an <b>int</b> value. Note that not all implementations allow this option to be set.
41007		
41008		
41009		
41010		
41011	SO_SNDTIMEO	Sets the timeout value specifying the amount of time that an output function blocks because flow control prevents data from being sent. If a send operation has blocked for this time, it shall return with a partial count or with <i>errno</i> set to [EAGAIN] or [EWOULDBLOCK] if no data is sent. The default for this option is zero, which indicates that a send operation shall not time out. This option stores a <b>timeval</b> structure. Note that not all implementations allow this option to be set.
41012		
41013		
41014		
41015		
41016		
41017		
41018		
41019		
41020		
41020		Options at other protocol levels vary in format and name.
41021		
41021	<b>RETURN VALUE</b>	
41022		Upon successful completion, <i>setsockopt()</i> shall return 0. Otherwise, -1 shall be returned and <i>errno</i> set to indicate the error.
41023		
41024	<b>ERRORS</b>	
41025		The <i>setsockopt()</i> function shall fail if:
41026	[EBADF]	The <i>socket</i> argument is not a valid file descriptor.

- 41027 [EDOM] The send and receive timeout values are too big to fit into the timeout fields in  
41028 the socket structure.
- 41029 [EINVAL] The specified option is invalid at the specified socket level or the socket has  
41030 been shut down.
- 41031 [EISCONN] The socket is already connected, and a specified option cannot be set while the  
41032 socket is connected.
- 41033 [ENOPROTOOPT]  
41034 The option is not supported by the protocol.
- 41035 [ENOTSOCK] The *socket* argument does not refer to a socket.
- 41036 The *setsockopt()* function may fail if:
- 41037 [ENOMEM] There was insufficient memory available for the operation to complete.
- 41038 [ENOBUFS] Insufficient resources are available in the system to complete the call.
- 41039 **EXAMPLES**
- 41040 None.
- 41041 **APPLICATION USAGE**
- 41042 The *setsockopt()* function provides an application program with the means to control socket  
41043 behavior. An application program can use *setsockopt()* to allocate buffer space, control timeouts,  
41044 or permit socket data broadcasts. The `<sys/socket.h>` header defines the socket-level options  
41045 available to *setsockopt()*.
- 41046 Options may exist at multiple protocol levels. The SO\_ options are always present at the  
41047 uppermost socket level.
- 41048 **RATIONALE**
- 41049 None.
- 41050 **FUTURE DIRECTIONS**
- 41051 None.
- 41052 **SEE ALSO**
- 41053 Section 2.10 (on page 508), *bind()*, *endprotoent()*, *getsockopt()*, *socket()*, the Base Definitions  
41054 volume of IEEE Std 1003.1-200x, `<netinet/in.h>`, `<sys/socket.h>`
- 41055 **CHANGE HISTORY**
- 41056 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

41057 **NAME**

41058            setstate — switch pseudo-random number generator state arrays

41059 **SYNOPSIS**

41060 xSI        #include <stdlib.h>

41061            char \*setstate(const char \*state);

41062

41063 **DESCRIPTION**

41064            Refer to *initstate()*.

41065 **NAME**

41066        setuid — set user ID

41067 **SYNOPSIS**

41068        #include &lt;unistd.h&gt;

41069        int setuid(uid\_t uid);

41070 **DESCRIPTION**41071        If the process has appropriate privileges, *setuid()* shall set the real user ID, effective user ID, and  
41072        the saved set-user-ID of the calling process to *uid*.41073        If the process does not have appropriate privileges, but *uid* is equal to the real user ID or the  
41074        saved set-user-ID, *setuid()* shall set the effective user ID to *uid*; the real user ID and saved set-  
41075        user-ID shall remain unchanged.41076        The *setuid()* function shall not affect the supplementary group list in any way.41077 **RETURN VALUE**41078        Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to  
41079        indicate the error.41080 **ERRORS**41081        The *setuid()* function shall fail, return -1, and set *errno* to the corresponding value if one or more  
41082        of the following are true:41083        [EINVAL]        The value of the *uid* argument is invalid and not supported by the  
41084        implementation.41085        [EPERM]        The process does not have appropriate privileges and *uid* does not match the  
41086        real user ID or the saved set-user-ID.41087 **EXAMPLES**

41088        None.

41089 **APPLICATION USAGE**

41090        None.

41091 **RATIONALE**41092        The various behaviors of the *setuid()* and *setgid()* functions when called by non-privileged  
41093        processes reflect the behavior of different historical implementations. For portability, it is  
41094        recommended that new non-privileged applications use the *seteuid()* and *setegid()* functions  
41095        instead.41096        The saved set-user-ID capability allows a program to regain the effective user ID established at  
41097        the last *exec* call. Similarly, the saved set-group-ID capability allows a program to regain the  
41098        effective group ID established at the last *exec* call. These capabilities are derived from System V.  
41099        Without them, a program might have to run as superuser in order to perform the same |  
41100        functions, because superuser can write on the user's files. This is a problem because such a |  
41101        program can write on any user's files, and so must be carefully written to emulate the |  
41102        permissions of the calling process properly. In System V, these capabilities have traditionally |  
41103        been implemented only via the *setuid()* and *setgid()* functions for non-privileged processes. The  
41104        fact that the behavior of those functions was different for privileged processes made them  
41105        difficult to use. The POSIX.1-1990 standard defined the *setuid()* function to behave differently  
41106        for privileged and unprivileged users. When the caller had the appropriate privilege, the  
41107        function set the calling process' real user ID, effective user ID, and saved set-user ID on |  
41108        implementations that supported it. When the caller did not have the appropriate privilege, the  
41109        function set only the effective user ID, subject to permission checks. The former use is generally  
41110        needed for utilities like *login* and *su*, which are not conforming applications and thus outside the |

41111 scope of IEEE Std 1003.1-200x. These utilities wish to change the user ID irrevocably to a new |  
41112 value, generally that of an unprivileged user. The latter use is needed for conforming |  
41113 applications that are installed with the set-user-ID bit and need to perform operations using the |  
41114 real user ID.

41115 IEEE Std 1003.1-200x augments the latter functionality with a mandatory feature named |  
41116 `_POSIX_SAVED_IDS`. This feature permits a set-user-ID application to switch its effective user |  
41117 ID back and forth between the values of its *exec*-time real user ID and effective user ID. |  
41118 Unfortunately, the POSIX.1-1990 standard did not permit a conforming application using this |  
41119 feature to work properly when it happened to be executed with the (implementation-defined) |  
41120 appropriate privilege. Furthermore, the application did not even have a means to tell whether it |  
41121 had this privilege. Since the saved set-user-ID feature is quite desirable for applications, as |  
41122 evidenced by the fact that NIST required it in FIPS 151-2, it has been mandated by |  
41123 IEEE Std 1003.1-200x. However, there are implementors who have been reluctant to support it |  
41124 given the limitation described above.

41125 The 4.3BSD system handles the problem by supporting separate functions: *setuid()* (which |  
41126 always sets both the real and effective user IDs, like *setuid()* in IEEE Std 1003.1-200x for |  
41127 privileged users), and *seteuid()* (which always sets just the effective user ID, like *setuid()* in |  
41128 IEEE Std 1003.1-200x for non-privileged users). This separation of functionality into distinct |  
41129 functions seems desirable. 4.3BSD does not support the saved set-user-ID feature. It supports |  
41130 similar functionality of switching the effective user ID back and forth via *setreuid()*, which |  
41131 permits reversing the real and effective user IDs. This model seems less desirable than the saved |  
41132 set-user-ID because the real user ID changes as a side effect. The current 4.4BSD includes saved |  
41133 effective IDs and uses them for *seteuid()* and *setegid()* as described above. The *setreuid()* and |  
41134 *setregid()* functions will be deprecated or removed.

41135 The solution here is:

- 41136 • Require that all implementations support the functionality of the saved set-user-ID, which is |  
41137 set by the *exec* functions and by privileged calls to *setuid()*.
- 41138 • Add the *seteuid()* and *setegid()* functions as portable alternatives to *setuid()* and *setgid()* for |  
41139 non-privileged and privileged processes.

41140 Historical systems have provided two mechanisms for a set-user-ID process to change its |  
41141 effective user ID to be the same as its real user ID in such a way that it could return to the |  
41142 original effective user ID: the use of the *setuid()* function in the presence of a saved set-user-ID, |  
41143 or the use of the BSD *setreuid()* function, which was able to swap the real and effective user IDs. |  
41144 The changes included in IEEE Std 1003.1-200x provide a new mechanism using *seteuid()* |  
41145 in conjunction with a saved set-user-ID. Thus, all implementations with the new *seteuid()* |  
41146 mechanism will have a saved set-user-ID for each process, and most of the behavior controlled |  
41147 by `_POSIX_SAVED_IDS` has been changed to agree with the case where the option was defined. |  
41148 The *kill()* function is an exception. Implementors of the new *seteuid()* mechanism will generally |  
41149 be required to maintain compatibility with the older mechanisms previously supported by their |  
41150 systems. However, compatibility with this use of *setreuid()* and with the `_POSIX_SAVED_IDS` |  
41151 behavior of *kill()* is unfortunately complicated. If an implementation with a saved set-user-ID |  
41152 allows a process to use *setreuid()* to swap its real and effective user IDs, but were to leave the |  
41153 saved set-user-ID unmodified, the process would then have an effective user ID equal to the |  
41154 original real user ID, and both real and saved set-user-ID would be equal to the original effective |  
41155 user ID. In that state, the real user would be unable to kill the process, even though the effective |  
41156 user ID of the process matches that of the real user, if the *kill()* behavior of `_POSIX_SAVED_IDS` |  
41157 was used. This is obviously not acceptable. The alternative choice, which is used in at least one |  
41158 implementation, is to change the saved set-user-ID to the effective user ID during most calls to |  
41159 *setreuid()*. The standard developers considered that alternative to be less correct than the

41160 retention of the old behavior of *kill()* in such systems. Current conforming applications shall  
41161 accommodate either behavior from *kill()*, and there appears to be no strong reason for *kill()* to  
41162 check the saved set-user-ID rather than the effective user ID.

41163 **FUTURE DIRECTIONS**

41164 None.

41165 **SEE ALSO**

41166 *exec*, *getegid()*, *geteuid()*, *getgid()*, *getuid()*, *setegid()*, *seteuid()*, *setgid()*, *setregid()*, *setreuid()*, the  
41167 Base Definitions volume of IEEE Std 1003.1-200x, <sys/types.h>, <unistd.h>

41168 **CHANGE HISTORY**

41169 First released in Issue 1. Derived from Issue 1 of the SVID.

41170 **Issue 6**

41171 In the SYNOPSIS, the optional include of the <sys/types.h> header is removed.

41172 The following new requirements on POSIX implementations derive from alignment with the  
41173 Single UNIX Specification:

- 41174 • The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was  
41175 required for conforming implementations of previous POSIX specifications, it was not  
41176 required for UNIX applications.
- 41177 • The functionality associated with \_POSIX\_SAVED\_IDS is now mandatory. This is a FIPS  
41178 requirement.

41179 The following changes were made to align with the IEEE P1003.1a draft standard:

- 41180 • The effects of *setuid()* in processes without appropriate privileges are changed.
- 41181 • A requirement that the supplementary group list is not affected is added.

41182 **NAME**

41183           setutxent — reset user accounting database to first entry

41184 **SYNOPSIS**

41185 XSI       #include <utmpx.h>

41186           void setutxent(void);

41187

41188 **DESCRIPTION**

41189           Refer to *endutxent()*.

41190 **NAME**

41191 setvbuf — assign buffering to a stream

41192 **SYNOPSIS**

41193 #include &lt;stdio.h&gt;

41194 int setvbuf(FILE \*restrict *stream*, char \*restrict *buf*, int *type*,  
41195 size\_t *size*);41196 **DESCRIPTION**41197 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
41198 conflict between the requirements described here and the ISO C standard is unintentional. This  
41199 volume of IEEE Std 1003.1-200x defers to the ISO C standard.41200 The *setvbuf()* function may be used after the stream pointed to by *stream* is associated with an  
41201 open file but before any other operation (other than an unsuccessful call to *setvbuf()*) is  
41202 performed on the stream. The argument *type* determines how *stream* shall be buffered, as  
41203 follows:

- 41204 • {\_IOFBF} shall cause input/output to be fully buffered.
- 41205 • {\_IOLBF} shall cause input/output to be line buffered.
- 41206 • {\_IONBF} shall cause input/output to be unbuffered.

41207 If *buf* is not a null pointer, the array it points to may be used instead of a buffer allocated by  
41208 *setvbuf()* and the argument *size* specifies the size of the array; otherwise, *size* may determine the  
41209 size of a buffer allocated by the *setvbuf()* function. The contents of the array at any time are  
41210 unspecified. |

41211 For information about streams, see Section 2.5 (on page 484).

41212 **RETURN VALUE**41213 Upon successful completion, *setvbuf()* shall return 0. Otherwise, it shall return a non-zero value  
41214 **CX** if an invalid value is given for *type* or if the request cannot be honored, and may set *errno* to  
41215 indicate the error.41216 **ERRORS**41217 The *setvbuf()* function may fail if:41218 **CX** [EBADF] The file descriptor underlying *stream* is not valid.41219 **EXAMPLES**

41220 None.

41221 **APPLICATION USAGE**41222 A common source of error is allocating buffer space as an “automatic” variable in a code block,  
41223 and then failing to close the stream in the same block.41224 With *setvbuf()*, allocating a buffer of *size* bytes does not necessarily imply that all of *size* bytes are  
41225 used for the buffer area.41226 Applications should note that many implementations only provide line buffering on input from  
41227 terminal devices.41228 **RATIONALE**

41229 None.

41230 **FUTURE DIRECTIONS**

41231 None.

41232 **SEE ALSO**41233 *fopen()*, *setbuf()*, the Base Definitions volume of IEEE Std 1003.1-200x, <**stdio.h**>41234 **CHANGE HISTORY**

41235 First released in Issue 1. Derived from Issue 1 of the SVID.

41236 **Issue 6**

41237 Extensions beyond the ISO C standard are now marked.

41238 The *setvbuf()* prototype is updated for alignment with the ISO/IEC 9899: 1999 standard.

## 41239 NAME

41240 shm\_open — open a shared memory object (**REALTIME**)

## 41241 SYNOPSIS

41242 SHM #include &lt;sys/mman.h&gt;

41243 int shm\_open(const char \*name, int oflag, mode\_t mode);

41244

## 41245 DESCRIPTION

41246 The *shm\_open()* function shall establish a connection between a shared memory object and a file  
 41247 descriptor. It shall create an open file description that refers to the shared memory object and a  
 41248 file descriptor that refers to that open file description. The file descriptor is used by other  
 41249 functions to refer to that shared memory object. The *name* argument points to a string naming a  
 41250 shared memory object. It is unspecified whether the name appears in the file system and is  
 41251 visible to other functions that take pathnames as arguments. The *name* argument conforms to the  
 41252 construction rules for a pathname. If *name* begins with the slash character, then processes calling  
 41253 *shm\_open()* with the same value of *name* refer to the same shared memory object, as long as that  
 41254 name has not been removed. If *name* does not begin with the slash character, the effect is  
 41255 implementation-defined. The interpretation of slash characters other than the leading slash  
 41256 character in *name* is implementation-defined.

41257 If successful, *shm\_open()* shall return a file descriptor for the shared memory object that is the  
 41258 lowest numbered file descriptor not currently open for that process. The open file description is  
 41259 new, and therefore the file descriptor does not share it with any other processes. It is unspecified  
 41260 whether the file offset is set. The FD\_CLOEXEC file descriptor flag associated with the new file  
 41261 descriptor is set.

41262 The file status flags and file access modes of the open file description are according to the value  
 41263 of *oflag*. The *oflag* argument is the bitwise-inclusive OR of the following flags defined in the  
 41264 <fcntl.h> header. Applications specify exactly one of the first two values (access modes) below  
 41265 in the value of *oflag*:

41266 O\_RDONLY Open for read access only.

41267 O\_RDWR Open for read or write access.

41268 Any combination of the remaining flags may be specified in the value of *oflag*:

41269 O\_CREAT If the shared memory object exists, this flag has no effect, except as noted  
 41270 under O\_EXCL below. Otherwise, the shared memory object is created; the user ID of the shared memory object shall be set to the effective user ID of the  
 41271 process; the group ID of the shared memory object is set to a system default group ID or to the effective group ID of the process. The permission bits of the  
 41272 shared memory object shall be set to the value of the *mode* argument except  
 41273 those set in the file mode creation mask of the process. When bits in *mode*  
 41274 other than the file permission bits are set, the effect is unspecified. The *mode*  
 41275 argument does not affect whether the shared memory object is opened for  
 41276 reading, for writing, or for both. The shared memory object has a size of zero.

41279 O\_EXCL If O\_EXCL and O\_CREAT are set, *shm\_open()* fails if the shared memory  
 41280 object exists. The check for the existence of the shared memory object and the  
 41281 creation of the object if it does not exist is atomic with respect to other  
 41282 processes executing *shm\_open()* naming the same shared memory object with  
 41283 O\_EXCL and O\_CREAT set. If O\_EXCL is set and O\_CREAT is not set, the  
 41284 result is undefined.

41285 O\_TRUNC If the shared memory object exists, and it is successfully opened O\_RDWR,  
 41286 the object shall be truncated to zero length and the mode and owner shall be  
 41287 unchanged by this function call. The result of using O\_TRUNC with  
 41288 O\_RDONLY is undefined.

41289 When a shared memory object is created, the state of the shared memory object, including all  
 41290 data associated with the shared memory object, persists until the shared memory object is  
 41291 unlinked and all other references are gone. It is unspecified whether the name and shared  
 41292 memory object state remain valid after a system reboot.

#### 41293 RETURN VALUE

41294 Upon successful completion, the *shm\_open()* function shall return a non-negative integer  
 41295 representing the lowest numbered unused file descriptor. Otherwise, it shall return  $-1$  and set  
 41296 *errno* to indicate the error.

#### 41297 ERRORS

41298 The *shm\_open()* function shall fail if:

41299 [EACCES] The shared memory object exists and the permissions specified by *oflag* are  
 41300 denied, or the shared memory object does not exist and permission to create  
 41301 the shared memory object is denied, or O\_TRUNC is specified and write  
 41302 permission is denied.

41303 [EEXIST] O\_CREAT and O\_EXCL are set and the named shared memory object already  
 41304 exists.

41305 [EINTR] The *shm\_open()* operation was interrupted by a signal.

41306 [EINVAL] The *shm\_open()* operation is not supported for the given name.

41307 [EMFILE] Too many file descriptors are currently in use by this process.

41308 [ENAMETOOLONG]

41309 The length of the *name* argument exceeds {PATH\_MAX} or a pathname  
 41310 component is longer than {NAME\_MAX}. |

41311 [ENFILE] Too many shared memory objects are currently open in the system.

41312 [ENOENT] O\_CREAT is not set and the named shared memory object does not exist.

41313 [ENOSPC] There is insufficient space for the creation of the new shared memory object.

#### 41314 EXAMPLES

41315 None.

#### 41316 APPLICATION USAGE

41317 None.

#### 41318 RATIONALE

41319 When the Memory Mapped Files option is supported, the normal *open()* call is used to obtain a  
 41320 descriptor to a file to be mapped according to existing practice with *mmap()*. When the Shared  
 41321 Memory Objects option is supported, the *shm\_open()* function shall obtain a descriptor to the  
 41322 shared memory object to be mapped. |

41323 There is ample precedent for having a file descriptor represent several types of objects. In the  
 41324 POSIX.1-1990 standard, a file descriptor can represent a file, a pipe, a FIFO, a tty, or a directory.  
 41325 Many implementations simply have an operations vector, which is indexed by the file descriptor  
 41326 type and does very different operations. Note that in some cases the file descriptor passed to  
 41327 generic operations on file descriptors are returned by *open()* or *creat()* and in some cases  
 41328 returned by alternate functions, such as *pipe()*. The latter technique is used by *shm\_open()*.

41329 Note that such shared memory objects can actually be implemented as mapped files. In both  
41330 cases, the size can be set after the open using *ftruncate()*. The *shm\_open()* function itself does not  
41331 create a shared object of a specified size because this would duplicate an extant function that set  
41332 the size of an object referenced by a file descriptor.

41333 On implementations where memory objects are implemented using the existing file system, the  
41334 *shm\_open()* function may be implemented using a macro that invokes *open()*, and the  
41335 *shm\_unlink()* function may be implemented using a macro that invokes *unlink()*.

41336 For implementations without a permanent file system, the definition of the name of the memory  
41337 objects is allowed not to survive a system reboot. Note that this allows systems with a  
41338 permanent file system to implement memory objects as data structures internal to the  
41339 implementation as well.

41340 On implementations that choose to implement memory objects using memory directly, a  
41341 *shm\_open()* followed by a *ftruncate()* and *close()* can be used to preallocate a shared memory  
41342 area and to set the size of that preallocation. This may be necessary for systems without virtual  
41343 memory hardware support in order to ensure that the memory is contiguous.

41344 The set of valid open flags to *shm\_open()* was restricted to *O\_RDONLY*, *O\_RDWR*, *O\_CREAT*,  
41345 and *O\_TRUNC* because these could be easily implemented on most memory mapping systems.  
41346 This volume of IEEE Std 1003.1-200x is silent on the results if the implementation cannot supply  
41347 the requested file access because of implementation-defined reasons, including hardware ones.

41348 The error conditions [EACCES] and [ENOTSUP] are provided to inform the application that the  
41349 implementation cannot complete a request.

41350 [EACCES] indicates for implementation-defined reasons, probably hardware-related, that the  
41351 implementation cannot comply with a requested mode because it conflicts with another  
41352 requested mode. An example might be that an application desires to open a memory object two  
41353 times, mapping different areas with different access modes. If the implementation cannot map a  
41354 single area into a process space in two places, which would be required if different access modes  
41355 were required for the two areas, then the implementation may inform the application at the time  
41356 of the second open.

41357 [ENOTSUP] indicates for implementation-defined reasons, probably hardware-related, that the  
41358 implementation cannot comply with a requested mode at all. An example would be that the  
41359 hardware of the implementation cannot support write-only shared memory areas.

41360 On all implementations, it may be desirable to restrict the location of the memory objects to  
41361 specific file systems for performance (such as a RAM disk) or implementation-defined reasons  
41362 (shared memory supported directly only on certain file systems). The *shm\_open()* function may  
41363 be used to enforce these restrictions. There are a number of methods available to the application  
41364 to determine an appropriate name of the file or the location of an appropriate directory. One  
41365 way is from the environment via *getenv()*. Another would be from a configuration file.

41366 This volume of IEEE Std 1003.1-200x specifies that memory objects have initial contents of zero  
41367 when created. This is consistent with current behavior for both files and newly allocated  
41368 memory. For those implementations that use physical memory, it would be possible that such  
41369 implementations could simply use available memory and give it to the process uninitialized.  
41370 This, however, is not consistent with standard behavior for the uninitialized data area, the stack,  
41371 and of course, files. Finally, it is highly desirable to set the allocated memory to zero for security  
41372 reasons. Thus, initializing memory objects to zero is required.

41373 **FUTURE DIRECTIONS**

41374 None.

41375 **SEE ALSO**

41376 *close()*, *dup()*, *exec*, *fcntl()*, *mmap()*, *shmat()*, *shmctl()*, *shmdt()*, *shm\_unlink()*, *umask()*, the Base  
41377 Definitions volume of IEEE Std 1003.1-200x, <fcntl.h>, <sys/mman.h>

41378 **CHANGE HISTORY**

41379 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

41380 **Issue 6**

41381 The *shm\_open()* function is marked as part of the Shared Memory Objects option.

41382 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
41383 implementation does not support the Shared Memory Objects option.

41384 **NAME**41385 shm\_unlink — remove a shared memory object (**REALTIME**)41386 **SYNOPSIS**

41387 SHM #include &lt;sys/mman.h&gt;

41388 int shm\_unlink(const char \*name);

41389

41390 **DESCRIPTION**41391 The *shm\_unlink()* function shall remove the name of the shared memory object named by the  
41392 string pointed to by *name*.41393 If one or more references to the shared memory object exist when the object is unlinked, the  
41394 name shall be removed before *shm\_unlink()* returns, but the removal of the memory object  
41395 contents shall be postponed until all open and map references to the shared memory object have  
41396 been removed.41397 Even if the object continues to exist after the last *shm\_unlink()*, reuse of the name shall  
41398 subsequently cause *shm\_open()* to behave as if no shared memory object of this name exists (that  
41399 is, *shm\_open()* will fail if *O\_CREAT* is not set, or will create a new shared memory object if  
41400 *O\_CREAT* is set).41401 **RETURN VALUE**41402 Upon successful completion, a value of zero shall be returned. Otherwise, a value of -1 shall be  
41403 returned and *errno* set to indicate the error. If -1 is returned, the named shared memory object  
41404 shall not be changed by this function call.41405 **ERRORS**41406 The *shm\_unlink()* function shall fail if:

41407 [EACCES] Permission is denied to unlink the named shared memory object.

41408 [ENAMETOOLONG]

41409 The length of the *name* argument exceeds {PATH\_MAX} or a pathname  
41410 component is longer than {NAME\_MAX}.

41411 [ENOENT] The named shared memory object does not exist.

41412 **EXAMPLES**

41413 None.

41414 **APPLICATION USAGE**41415 Names of memory objects that were allocated with *open()* are deleted with *unlink()* in the usual  
41416 fashion. Names of memory objects that were allocated with *shm\_open()* are deleted with  
41417 *shm\_unlink()*. Note that the actual memory object is not destroyed until the last close and  
41418 unmap on it have occurred if it was already in use.41419 **RATIONALE**

41420 None.

41421 **FUTURE DIRECTIONS**

41422 None.

41423 **SEE ALSO**41424 *close()*, *mmap()*, *munmap()*, *shmat()*, *shmctl()*, *shmdt()*, *shm\_open()*, the Base Definitions volume  
41425 of IEEE Std 1003.1-200x, <sys/mman.h>

## 41426 CHANGE HISTORY

41427 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

## 41428 Issue 6

41429 The *shm\_unlink()* function is marked as part of the Shared Memory Objects option.

41430 In the DESCRIPTION, text is added to clarify that reusing the same name after a *shm\_unlink()*  
41431 will not attach to the old shared memory object.

41432 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
41433 implementation does not support the Shared Memory Objects option.

41434 **NAME**41435 `shmat` — XSI shared memory attach operation41436 **SYNOPSIS**41437 XSI 

```
#include <sys/shm.h>
```

41438 

```
void *shmat(int shmid, const void *shmaddr, int shmflg);
```

41439

41440 **DESCRIPTION**

41441 The `shmat()` function operates on XSI shared memory (see the Base Definitions volume of  
 41442 IEEE Std 1003.1-200x, Section 3.340, Shared Memory Object). It is unspecified whether this  
 41443 function interoperates with the realtime interprocess communication facilities defined in Section  
 41444 2.8 (on page 491).

41445 The `shmat()` function attaches the shared memory segment associated with the shared memory  
 41446 identifier specified by `shmid` to the address space of the calling process. The segment is attached  
 41447 at the address specified by one of the following criteria:

- 41448 • If `shmaddr` is a null pointer, the segment is attached at the first available address as selected  
 41449 by the system.
- 41450 • If `shmaddr` is not a null pointer and `(shmflg & SHM_RND)` is non-zero, the segment is attached  
 41451 at the address given by `(shmaddr - ((uintptr_t)shmaddr % SHMLBA))`. The character `'%'` is the  
 41452 C-language remainder operator.
- 41453 • If `shmaddr` is not a null pointer and `(shmflg & SHM_RND)` is 0, the segment is attached at the  
 41454 address given by `shmaddr`.
- 41455 • The segment is attached for reading if `(shmflg & SHM_RDONLY)` is non-zero and the calling  
 41456 process has read permission; otherwise, if it is 0 and the calling process has read and write  
 41457 permission, the segment is attached for reading and writing.

41458 **RETURN VALUE**

41459 Upon successful completion, `shmat()` shall increment the value of `shm_nattch` in the data  
 41460 structure associated with the shared memory ID of the attached shared memory segment and  
 41461 return the segment's start address.

41462 Otherwise, the shared memory segment shall not be attached, `shmat()` shall return `-1`, and `errno`  
 41463 shall be set to indicate the error.

41464 **ERRORS**41465 The `shmat()` function shall fail if:

- |   |          |   |
|---|----------|---|
| 41466<br>41467                            | [EACCES] | Operation permission is denied to the calling process; see Section 2.7 (on page 489).   |
| 41468<br>41469<br>41470<br>41471<br>41472 | [EINVAL] | The value of <code>shmid</code> is not a valid shared memory identifier, the <code>shmaddr</code> is not a null pointer, and the value of <code>(shmaddr - ((uintptr_t)shmaddr % SHMLBA))</code> is an illegal address for attaching shared memory; or the <code>shmaddr</code> is not a null pointer, <code>(shmflg &amp; SHM_RND)</code> is 0, and the value of <code>shmaddr</code> is an illegal address for attaching shared memory. |
| 41473<br>41474                            | [EMFILE] | The number of shared memory segments attached to the calling process would exceed the system-imposed limit.   |
| 41475<br>41476                            | [ENOMEM] | The available data space is not large enough to accommodate the shared memory segment.  |

41477 **EXAMPLES**

41478 None.

41479 **APPLICATION USAGE**

41480 The POSIX Realtime Extension defines alternative interfaces for interprocess communication.  
41481 Application developers who need to use IPC should design their applications so that modules  
41482 using the IPC routines described in Section 2.7 (on page 489) can be easily modified to use the  
41483 alternative interfaces.

41484 **RATIONALE**

41485 None.

41486 **FUTURE DIRECTIONS**

41487 None.

41488 **SEE ALSO**

41489 *exec*, *exit()*, *fork()*, *shmctl()*, *shmdt()*, *shmget()*, *shm\_open()*, *shm\_unlink()*, the Base Definitions  
41490 volume of IEEE Std 1003.1-200x, <sys/shm.h>, Section 2.7 (on page 489)

41491 **CHANGE HISTORY**

41492 First released in Issue 2. Derived from Issue 2 of the SVID.

41493 **Issue 5**

41494 Moved from SHARED MEMORY to BASE.

41495 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE  
41496 DIRECTIONS to a new APPLICATION USAGE section.

41497 **Issue 6**

41498 The Open Group Corrigendum U021/13 is applied.

41499 **NAME**

41500 shmctl — XSI shared memory control operations

41501 **SYNOPSIS**41502 XSI 

```
#include <sys/shm.h>
```

41503 

```
int shmctl(int shmid, int cmd, struct shm_id_ds *buf);
```

41504

41505 **DESCRIPTION**

41506 The *shmctl()* function operates on XSI shared memory (see the Base Definitions volume of  
 41507 IEEE Std 1003.1-200x, Section 3.340, Shared Memory Object). It is unspecified whether this  
 41508 function interoperates with the realtime interprocess communication facilities defined in Section  
 41509 2.8 (on page 491).

41510 The *shmctl()* function provides a variety of shared memory control operations as specified by  
 41511 *cmd*. The following values for *cmd* are available:

41512 **IPC\_STAT** Place the current value of each member of the **shm\_id\_ds** data structure  
 41513 associated with *shmid* into the structure pointed to by *buf*. The contents of the  
 41514 structure are defined in `<sys/shm.h>`.

41515 **IPC\_SET** Set the value of the following members of the **shm\_id\_ds** data structure  
 41516 associated with *shmid* to the corresponding value found in the structure  
 41517 pointed to by *buf*:

41518 shm\_perm.uid  
 41519 shm\_perm.gid  
 41520 shm\_perm.mode Low-order nine bits.

41521 **IPC\_SET** can only be executed by a process that has an effective user ID equal  
 41522 to either that of a process with appropriate privileges or to the value of  
 41523 *shm\_perm.cuid* or *shm\_perm.uid* in the **shm\_id\_ds** data structure associated with  
 41524 *shmid*.

41525 **IPC\_RMID** Remove the shared memory identifier specified by *shmid* from the system and  
 41526 destroy the shared memory segment and **shm\_id\_ds** data structure associated  
 41527 with it. **IPC\_RMID** can only be executed by a process that has an effective user  
 41528 ID equal to either that of a process with appropriate privileges or to the value  
 41529 of *shm\_perm.cuid* or *shm\_perm.uid* in the **shm\_id\_ds** data structure associated  
 41530 with *shmid*.

41531 **RETURN VALUE**

41532 Upon successful completion, *shmctl()* shall return 0; otherwise, it shall return -1 and set *errno* to  
 41533 indicate the error.

41534 **ERRORS**

41535 The *shmctl()* function shall fail if:

41536 **[EACCES]** The argument *cmd* is equal to **IPC\_STAT** and the calling process does not have  
 41537 read permission; see Section 2.7 (on page 489).

41538 **[EINVAL]** The value of *shmid* is not a valid shared memory identifier, or the value of *cmd*  
 41539 is not a valid command.

41540 **[EPERM]** The argument *cmd* is equal to **IPC\_RMID** or **IPC\_SET** and the effective user ID  
 41541 of the calling process is not equal to that of a process with appropriate  
 41542 privileges and it is not equal to the value of *shm\_perm.cuid* or *shm\_perm.uid* in  
 41543 the data structure associated with *shmid*.

- 41544 The *shmctl()* function may fail if:
- 41545 [EOVERFLOW] The *cmd* argument is `IPC_STAT` and the *gid* or *uid* value is too large to be  
41546 stored in the structure pointed to by the *buf* argument.
- 41547 **EXAMPLES**
- 41548 None.
- 41549 **APPLICATION USAGE**
- 41550 The POSIX Realtime Extension defines alternative interfaces for interprocess communication.  
41551 Application developers who need to use IPC should design their applications so that modules  
41552 using the IPC routines described in Section 2.7 (on page 489) can be easily modified to use the  
41553 alternative interfaces.
- 41554 **RATIONALE**
- 41555 None.
- 41556 **FUTURE DIRECTIONS**
- 41557 None.
- 41558 **SEE ALSO**
- 41559 *shmat()*, *shmdt()*, *shmget()*, *shm\_open()*, *shm\_unlink()*, the Base Definitions volume of  
41560 IEEE Std 1003.1-200x, <sys/shm.h>, Section 2.7 (on page 489)
- 41561 **CHANGE HISTORY**
- 41562 First released in Issue 2. Derived from Issue 2 of the SVID.
- 41563 **Issue 5**
- 41564 Moved from SHARED MEMORY to BASE.
- 41565 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE  
41566 DIRECTIONS to a new APPLICATION USAGE section.

41567 **NAME**

41568           shmdt — XSI shared memory detach operation

41569 **SYNOPSIS**

41570 XSI       #include &lt;sys/shm.h&gt;

41571           int shmdt(const void \*shmaddr);

41572

41573 **DESCRIPTION**

41574       The *shmdt()* function operates on XSI shared memory (see the Base Definitions volume of  
 41575       IEEE Std 1003.1-200x, Section 3.340, Shared Memory Object). It is unspecified whether this  
 41576       function interoperates with the realtime interprocess communication facilities defined in Section  
 41577       2.8 (on page 491).

41578       The *shmdt()* function detaches the shared memory segment located at the address specified by  
 41579       *shmaddr* from the address space of the calling process.

41580 **RETURN VALUE**

41581       Upon successful completion, *shmdt()* shall decrement the value of *shm\_nattch* in the data  
 41582       structure associated with the shared memory ID of the attached shared memory segment and  
 41583       return 0.

41584       Otherwise, the shared memory segment shall not be detached, *shmdt()* shall return  $-1$ , and *errno*  
 41585       shall be set to indicate the error.

41586 **ERRORS**41587       The *shmdt()* function shall fail if:

41588       [EINVAL]       The value of *shmaddr* is not the data segment start address of a shared  
 41589       memory segment.

41590 **EXAMPLES**

41591       None.

41592 **APPLICATION USAGE**

41593       The POSIX Realtime Extension defines alternative interfaces for interprocess communication.  
 41594       Application developers who need to use IPC should design their applications so that modules  
 41595       using the IPC routines described in Section 2.7 (on page 489) can be easily modified to use the  
 41596       alternative interfaces.

41597 **RATIONALE**

41598       None.

41599 **FUTURE DIRECTIONS**

41600       None.

41601 **SEE ALSO**

41602       *exec*, *exit()*, *fork()*, *shmat()*, *shmctl()*, *shmget()*, *shm\_open()*, *shm\_unlink()*, the Base Definitions  
 41603       volume of IEEE Std 1003.1-200x, <sys/shm.h>, Section 2.7 (on page 489)

41604 **CHANGE HISTORY**

41605       First released in Issue 2. Derived from Issue 2 of the SVID.

41606 **Issue 5**

41607       Moved from SHARED MEMORY to BASE.

41608       The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE  
 41609       DIRECTIONS to a new APPLICATION USAGE section.

## 41610 NAME

41611 shmget — get XSI shared memory segment

## 41612 SYNOPSIS

41613 XSI 

```
#include <sys/shm.h>
```

41614 

```
int shmget(key_t key, size_t size, int shmflg);
```

41615

## 41616 DESCRIPTION

41617 The *shmget()* function operates on XSI shared memory (see the Base Definitions volume of  
 41618 IEEE Std 1003.1-200x, Section 3.340, Shared Memory Object). It is unspecified whether this  
 41619 function interoperates with the realtime interprocess communication facilities defined in Section  
 41620 2.8 (on page 491).

41621 The *shmget()* function shall return the shared memory identifier associated with *key*.

41622 A shared memory identifier, associated data structure, and shared memory segment of at least  
 41623 *size* bytes (see <sys/shm.h>) are created for *key* if one of the following is true:

- 41624 • The argument *key* is equal to `IPC_PRIVATE`.
- 41625 • The argument *key* does not already have a shared memory identifier associated with it and  
 41626 (*shmflg* & `IPC_CREAT`) is non-zero.

41627 Upon creation, the data structure associated with the new shared memory identifier shall be  
 41628 initialized as follows:

- 41629 • The values of *shm\_perm.cuid*, *shm\_perm.uid*, *shm\_perm.cgid*, and *shm\_perm.gid* are set equal to  
 41630 the effective user ID and effective group ID, respectively, of the calling process.
- 41631 • The low-order nine bits of *shm\_perm.mode* are set equal to the low-order nine bits of *shmflg*.  
 41632 The value of *shm\_segsz* is set equal to the value of *size*.
- 41633 • The values of *shm\_lpid*, *shm\_nattch*, *shm\_atime*, and *shm\_dtime* are set equal to 0.
- 41634 • The value of *shm\_ctime* is set equal to the current time.

41635 When the shared memory segment is created, it shall be initialized with all zero values.

## 41636 RETURN VALUE

41637 Upon successful completion, *shmget()* shall return a non-negative integer, namely a shared  
 41638 memory identifier; otherwise, it shall return `-1` and set *errno* to indicate the error.

## 41639 ERRORS

41640 The *shmget()* function shall fail if:

- |       |          |   |
|-------|----------|---|
| 41641 | [EACCES] | A shared memory identifier exists for <i>key</i> but operation permission as<br>41642 specified by the low-order nine bits of <i>shmflg</i> would not be granted; see<br>41643 Section 2.7 (on page 489).           |
| 41644 | [EEXIST] | A shared memory identifier exists for the argument <i>key</i> but ( <i>shmflg</i><br>41645 & <code>IPC_CREAT</code> ) && ( <i>shmflg</i> & <code>IPC_EXCL</code> ) is non-zero.                                     |
| 41646 | [EINVAL] | A shared memory segment is to be created and the value of <i>size</i> is less than  <br>41647 the system-imposed minimum or greater than the system-imposed maximum.  |
| 41648 | [EINVAL] | No shared memory segment is to be created and a shared memory segment  <br>41649 exists for <i>key</i> but the size of the segment associated with it is less than <i>size</i> and  <br>41650 <i>size</i> is not 0. |

- 41651 [ENOENT] A shared memory identifier does not exist for the argument *key* and (*shmflg*  
41652 &IPC\_CREAT) is 0.
- 41653 [ENOMEM] A shared memory identifier and associated shared memory segment shall be  
41654 created, but the amount of available physical memory is not sufficient to fill  
41655 the request.
- 41656 [ENOSPC] A shared memory identifier is to be created, but the system-imposed limit on  
41657 the maximum number of allowed shared memory identifiers system-wide  
41658 would be exceeded.

**41659 EXAMPLES**

41660 None.

**41661 APPLICATION USAGE**

41662 The POSIX Realtime Extension defines alternative interfaces for interprocess communication.  
41663 Application developers who need to use IPC should design their applications so that modules  
41664 using the IPC routines described in Section 2.7 (on page 489) can be easily modified to use the  
41665 alternative interfaces.

**41666 RATIONALE**

41667 None.

**41668 FUTURE DIRECTIONS**

41669 None.

**41670 SEE ALSO**

41671 *shmat()*, *shmctl()*, *shmdt()*, *shm\_open()*, *shm\_unlink()*, the Base Definitions volume of  
41672 IEEE Std 1003.1-200x, <sys/shm.h>, Section 2.7 (on page 489)

**41673 CHANGE HISTORY**

41674 First released in Issue 2. Derived from Issue 2 of the SVID.

**41675 Issue 5**

41676 Moved from SHARED MEMORY to BASE.

41677 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE  
41678 DIRECTIONS to a new APPLICATION USAGE section.

41679 **NAME**

41680 shutdown — shut down socket send and receive operations

41681 **SYNOPSIS**

41682 #include <sys/socket.h>

41683 int shutdown(int *socket*, int *how*);

41684 **DESCRIPTION**

41685 The *shutdown()* function shall cause all or part of a full-duplex connection on the socket  
41686 associated with the file descriptor *socket* to be shut down.

41687 The *shutdown()* function takes the following arguments:

41688 *socket* Specifies the file descriptor of the socket.

41689 *how* Specifies the type of shutdown. The values are as follows:

41690 SHUT\_RD Disables further receive operations.

41691 SHUT\_WR Disables further send operations.

41692 SHUT\_RDWR Disables further send and receive operations.

41693 The *shutdown()* function disables subsequent send and/or receive operations on a socket,  
41694 depending on the value of the *how* argument.

41695 **RETURN VALUE**

41696 Upon successful completion, *shutdown()* shall return 0; otherwise, -1 shall be returned and *errno*  
41697 set to indicate the error.

41698 **ERRORS**

41699 The *shutdown()* function shall fail if:

41700 [EBADF] The *socket* argument is not a valid file descriptor.

41701 [EINVAL] The *how* argument is invalid.

41702 [ENOTCONN] The socket is not connected.

41703 [ENOTSOCK] The *socket* argument does not refer to a socket.

41704 The *shutdown()* function may fail if:

41705 [ENOBUFS] Insufficient resources were available in the system to perform the operation.

41706 **EXAMPLES**

41707 None.

41708 **APPLICATION USAGE**

41709 None.

41710 **RATIONALE**

41711 None.

41712 **FUTURE DIRECTIONS**

41713 None.

41714 **SEE ALSO**

41715 *getsockopt()*, *read()*, *recv()*, *recvfrom()*, *recvmsg()*, *select()*, *send()*, *sendto()*, *setsockopt()*, *socket()*,  
41716 *write()*, the Base Definitions volume of IEEE Std 1003.1-200x, <sys/socket.h>

41717 **CHANGE HISTORY**

41718 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

## 41719 NAME

41720 sigaction — examine and change signal action

## 41721 SYNOPSIS

41722 cx #include &lt;signal.h&gt;

41723 int sigaction(int sig, const struct sigaction \*restrict act,  
41724 struct sigaction \*restrict oact);

41725

## 41726 DESCRIPTION

41727 The *sigaction()* function allows the calling process to examine and/or specify the action to be  
41728 associated with a specific signal. The argument *sig* specifies the signal; acceptable values are  
41729 defined in <signal.h>.41730 The structure **sigaction**, used to describe an action to be taken, is defined in the <signal.h>  
41731 header to include at least the following members:

41732

41733

41734

41735

41736

41737

41738

41739

41740

Member Type	Member Name	Description
void(*) (int) sigset_t	sa_handler sa_mask	SIG_DFL, SIG_IGN, or pointer to a function. Additional set of signals to be blocked during execution of signal-catching function.
int	sa_flags	Special flags to affect behavior of signal.
void(*) (int, siginfo_t *, void *)	sa_sigaction	Signal-catching function.

41741 The storage occupied by *sa\_handler* and *sa\_sigaction* may overlap, and a conforming application  
41742 shall not use both simultaneously.41743 If the argument *act* is not a null pointer, it points to a structure specifying the action to be  
41744 associated with the specified signal. If the argument *oact* is not a null pointer, the action  
41745 previously associated with the signal is stored in the location pointed to by the argument *oact*. If  
41746 the argument *act* is a null pointer, signal handling is unchanged; thus, the call can be used to  
41747 enquire about the current handling of a given signal. The SIGKILL and SIGSTOP signals shall  
41748 not be added to the signal mask using this mechanism; this restriction shall be enforced by the  
41749 system without causing an error to be indicated.41750 If the SA\_SIGINFO flag (see below) is cleared in the *sa\_flags* field of the **sigaction** structure, the  
41751 XSI|RTS *sa\_handler* field identifies the action to be associated with the specified signal. If the  
41752 SA\_SIGINFO flag is set in the *sa\_flags* field, and the implementation supports the Realtime  
41753 Signals Extension option or the X/Open System Interfaces Extension option, the *sa\_sigaction*  
41754 field specifies a signal-catching function. If the SA\_SIGINFO bit is cleared and the *sa\_handler*  
41755 field specifies a signal-catching function, or if the SA\_SIGINFO bit is set, the *sa\_mask* field  
41756 identifies a set of signals that shall be added to the signal mask of the thread before the signal-  
41757 catching function is invoked. If the *sa\_handler* field specifies a signal-catching function, the  
41758 *sa\_mask* field identifies a set of signals that shall be added to the process' signal mask before the  
41759 signal-catching function is invoked.41760 The *sa\_flags* field can be used to modify the behavior of the specified signal.41761 The following flags, defined in the <signal.h> header, can be set in *sa\_flags*:41762 XSI SA\_NOCLDSTOP Do not generate SIGCHLD when children stop or stopped children  
41763 continue.

41764		If <i>sig</i> is SIGCHLD and the SA_NOCLDSTOP flag is not set in <i>sa_flags</i> , and the implementation supports the SIGCHLD signal, then a SIGCHLD signal shall be generated for the calling process whenever any of its child processes stop and a SIGCHLD signal may be generated for the calling process whenever any of its stopped child processes are continued. If <i>sig</i> is SIGCHLD and the SA_NOCLDSTOP flag is set in <i>sa_flags</i> , then the implementation shall not generate a SIGCHLD signal in this way.
41765		
41766		
41767 XSI		
41768		
41769		
41770		
41771 XSI	SA_ONSTACK	If set and an alternate signal stack has been declared with <i>sigaltstack()</i> or <i>sigstack()</i> , the signal shall be delivered to the calling process on that stack. Otherwise, the signal shall be delivered on the current stack.
41772		
41773		
41774 XSI	SA_RESETHAND	If set, the disposition of the signal shall be reset to SIG_DFL and the SA_SIGINFO flag shall be cleared on entry to the signal handler.
41775		
41776		<b>Note:</b> SIGILL and SIGTRAP cannot be automatically reset when delivered; the system silently enforces this restriction.
41777		
41778		Otherwise, the disposition of the signal shall not be modified on entry to the signal handler.
41779		
41780		In addition, if this flag is set, <i>sigaction()</i> behaves as if the SA_NODEFER flag were also set.
41781		
41782 XSI	SA_RESTART	This flag affects the behavior of interruptible functions; that is, those specified to fail with <i>errno</i> set to [EINTR]. If set, and a function specified as interruptible is interrupted by this signal, the function shall restart and shall not fail with [EINTR] unless otherwise specified. If the flag is not set, interruptible functions interrupted by this signal shall fail with <i>errno</i> set to [EINTR].
41783		
41784		
41785		
41786		
41787		
41788	SA_SIGINFO	If cleared and the signal is caught, the signal-catching function shall be entered as:
41789		
41790		<pre>void func(int signo);</pre>
41791		where <i>signo</i> is the only argument to the signal catching function. In this case, the application shall use the <i>sa_handler</i> member to describe the signal catching function and the application shall not modify the <i>sa_sigaction</i> member.
41792		
41793		
41794		
41795 XSI RTS		If SA_SIGINFO is set and the signal is caught, the signal-catching function shall be entered as:
41796		
41797		<pre>void func(int signo, siginfo_t *info, void *context);</pre>
41798		where two additional arguments are passed to the signal catching function. The second argument shall point to an object of type <b>siginfo_t</b> explaining the reason why the signal was generated; the third argument can be cast to a pointer to an object of type <b>ucontext_t</b> to refer to the receiving process' context that was interrupted when the signal was delivered. In this case, the application shall use the <i>sa_sigaction</i> member to describe the signal catching function and the application shall not modify the <i>sa_handler</i> member.
41799		
41800		
41801		
41802		
41803		
41804		
41805		
41806		The <i>si_signo</i> member contains the system-generated signal number.
41807 XSI		The <i>si_errno</i> member may contain implementation-defined additional error information; if non-zero, it contains an error number identifying the condition that caused the signal to be generated.
41808		
41809		

41810 XSI|RTS                   The *si\_code* member contains a code identifying the cause of the signal. |

41811 XSI                        If the value of *si\_code* is less than or equal to 0, then the signal was |  
41812                               generated by a process and *si\_pid* and *si\_uid*, respectively, indicate the  
41813                               process ID and the real user ID of the sender. The `<signal.h>` header  
41814                               description contains information about the signal specific contents of the  
41815                               elements of the **siginfo\_t** type.

41816 XSI                   SA\_NOCLDWAIT           If set, and *sig* equals SIGCHLD, child processes of the calling processes  
41817                               shall not be transformed into zombie processes when they terminate. If  
41818                               the calling process subsequently waits for its children, and the process  
41819                               has no unwaited-for children that were transformed into zombie  
41820                               processes, it shall block until all of its children terminate, and *wait()*,  
41821                               *waitid()*, and *waitpid()* shall fail and set *errno* to [ECHILD]. Otherwise,  
41822                               terminating child processes shall be transformed into zombie processes,  
41823                               unless SIGCHLD is set to SIG\_IGN.

41824 XSI                   SA\_NODEFER             If set and *sig* is caught, *sig* shall not be added to the process' signal mask  
41825                               on entry to the signal handler unless it is included in *sa\_mask*. Otherwise,  
41826                               *sig* shall always be added to the process' signal mask on entry to the  
41827                               signal handler.

41828                               When a signal is caught by a signal-catching function installed by *sigaction()*, a new signal mask  
41829                               is calculated and installed for the duration of the signal-catching function (or until a call to either  
41830                               *sigprocmask()* or *sigsuspend()* is made). This mask is formed by taking the union of the current  
41831 XSI                               signal mask and the value of the *sa\_mask* for the signal being delivered unless SA\_NODEFER or  
41832                               SA\_RESETHAND is set, and then including the signal being delivered. If and when the user's  
41833                               signal handler returns normally, the original signal mask is restored.

41834                               Once an action is installed for a specific signal, it shall remain installed until another action is  
41835 XSI                               explicitly requested (by another call to *sigaction()*), until the SA\_RESETHAND flag causes  
41836                               resetting of the handler, or until one of the *exec* functions is called.

41837                               If the previous action for *sig* had been established by *signal()*, the values of the fields returned in  
41838                               the structure pointed to by *oact* are unspecified, and in particular *oact->sa\_handler* is not  
41839                               necessarily the same value passed to *signal()*. However, if a pointer to the same structure or a  
41840                               copy thereof is passed to a subsequent call to *sigaction()* via the *act* argument, handling of the  
41841                               signal shall be as if the original call to *signal()* were repeated.

41842                               If *sigaction()* fails, no new signal handler is installed.

41843                               It is unspecified whether an attempt to set the action for a signal that cannot be caught or  
41844                               ignored to SIG\_DFL is ignored or causes an error to be returned with *errno* set to [EINVAL].

41845                               If SA\_SIGINFO is not set in *sa\_flags*, then the disposition of subsequent occurrences of *sig* when  
41846                               it is already pending is implementation-defined; the signal-catching function shall be invoked  
41847 RTS                               with a single argument. If the implementation supports the Realtime Signals Extension option,  
41848                               and if SA\_SIGINFO is set in *sa\_flags*, then subsequent occurrences of *sig* generated by *sigqueue()*  
41849                               or as a result of any signal-generating function that supports the specification of an application-  
41850                               defined value (when *sig* is already pending) shall be queued in FIFO order until delivered or  
41851                               accepted; the signal-catching function shall be invoked with three arguments. The application  
41852                               specified value is passed to the signal-catching function as the *si\_value* member of the **siginfo\_t**  
41853                               structure.

41854                               The result of the use of *sigaction()* and a *sigwait()* function concurrently within a process on the  
41855                               same signal is unspecified.

## 41856 RETURN VALUE

41857 Upon successful completion, *sigaction()* shall return 0; otherwise, -1 shall be returned, *errno* shall  
 41858 be set to indicate the error, and no new signal-catching function shall be installed.

## 41859 ERRORS

41860 The *sigaction()* function shall fail if:

41861 [EINVAL] The *sig* argument is not a valid signal number or an attempt is made to catch a  
 41862 signal that cannot be caught or ignore a signal that cannot be ignored.

41863 [ENOTSUP] The SA\_SIGINFO bit flag is set in the *sa\_flags* field of the **sigaction** structure,  
 41864 and the implementation does not support either the Realtime Signals  
 41865 Extension option, or the X/Open System Interfaces Extension option.

41866 The *sigaction()* function may fail if:

41867 [EINVAL] An attempt was made to set the action to SIG\_DFL for a signal that cannot be  
 41868 caught or ignored (or both).

## 41869 EXAMPLES

41870 None.

## 41871 APPLICATION USAGE

41872 The *sigaction()* function supersedes the *signal()* function, and should be used in preference. In  
 41873 particular, *sigaction()* and *signal()* should not be used in the same process to control the same  
 41874 signal. The behavior of reentrant functions, as defined in the DESCRIPTION, is as specified by  
 41875 this volume of IEEE Std 1003.1-200x, regardless of invocation from a signal-catching function.  
 41876 This is the only intended meaning of the statement that reentrant functions may be used in  
 41877 signal-catching functions without restrictions. Applications must still consider all effects of such  
 41878 functions on such things as data structures, files, and process state. In particular, application  
 41879 writers need to consider the restrictions on interactions when interrupting *sleep()* and  
 41880 interactions among multiple handles for a file description. The fact that any specific function is  
 41881 listed as reentrant does not necessarily mean that invocation of that function from a signal-  
 41882 catching function is recommended.

41883 In order to prevent errors arising from interrupting non-reentrant function calls, applications  
 41884 should protect calls to these functions either by blocking the appropriate signals or through the  
 41885 use of some programmatic semaphore (see *semget()*, *sem\_init()*, *sem\_open()*, and so on). Note in  
 41886 particular that even the “safe” functions may modify *errno*; the signal-catching function, if not  
 41887 executing as an independent thread, may want to save and restore its value. Naturally, the same  
 41888 principles apply to the reentrancy of application routines and asynchronous data access. Note  
 41889 that *longjmp()* and *siglongjmp()* are not in the list of reentrant functions. This is because the code  
 41890 executing after *longjmp()* and *siglongjmp()* can call any unsafe functions with the same danger as  
 41891 calling those unsafe functions directly from the signal handler. Applications that use *longjmp()*  
 41892 and *siglongjmp()* from within signal handlers require rigorous protection in order to be portable.  
 41893 Many of the other functions that are excluded from the list are traditionally implemented using  
 41894 either *malloc()* or *free()* functions or the standard I/O library, both of which traditionally use  
 41895 data structures in a non-reentrant manner. Since any combination of different functions using a  
 41896 common data structure can cause reentrancy problems, this volume of IEEE Std 1003.1-200x  
 41897 does not define the behavior when any unsafe function is called in a signal handler that  
 41898 interrupts an unsafe function.

41899 If the signal occurs other than as the result of calling *abort()*, *kill()*, or *raise()*, the behavior is  
 41900 undefined if the signal handler calls any function in the standard library other than one of the  
 41901 functions listed in the table above or refers to any object with static storage duration other than  
 41902 by assigning a value to a static storage duration variable of type **volatile sig\_atomic\_t**.  
 41903 Furthermore, if such a call fails, the value of *errno* is unspecified.

41904 Usually, the signal is executed on the stack that was in effect before the signal was delivered. An  
41905 alternate stack may be specified to receive a subset of the signals being caught.

41906 When the signal handler returns, the receiving process resumes execution at the point it was  
41907 interrupted unless the signal handler makes other arrangements. If *longjmp()* or *\_longjmp()*  
41908 is used to leave the signal handler, then the signal mask must be explicitly restored by the process.

41909 This volume of IEEE Std 1003.1-200x defines the third argument of a signal handling function  
41910 when SA\_SIGINFO is set as a **void** \* instead of a **ucontext\_t** \*, but without requiring type  
41911 checking. New applications should explicitly cast the third argument of the signal handling  
41912 function to **ucontext\_t** \*.

41913 The BSD optional four argument signal handling function is not supported by this volume of  
41914 IEEE Std 1003.1-200x. The BSD declaration would be:

```
41915 void handler(int sig, int code, struct sigcontext *scp,  
41916             char *addr);
```

41917 where *sig* is the signal number, *code* is additional information on certain signals, *scp* is a pointer  
41918 to the sigcontext structure, and *addr* is additional address information. Much the same  
41919 information is available in the objects pointed to by the second argument of the signal handler  
41920 specified when SA\_SIGINFO is set.

#### 41921 RATIONALE

41922 Although this volume of IEEE Std 1003.1-200x requires that signals that cannot be ignored shall  
41923 not be added to the signal mask when a signal-catching function is entered, there is no explicit  
41924 requirement that subsequent calls to *sigaction()* reflect this in the information returned in the *oact*  
41925 argument. In other words, if SIGKILL is included in the *sa\_mask* field of *act*, it is unspecified  
41926 whether or not a subsequent call to *sigaction()* returns with SIGKILL included in the *sa\_mask*  
41927 field of *oact*.

41928 The SA\_NOCLDSTOP flag, when supplied in the *act->sa\_flags* parameter, allows overloading  
41929 SIGCHLD with the System V semantics that each SIGCLD signal indicates a single terminated |  
41930 child. Most conforming applications that catch SIGCHLD are expected to install signal-catching |  
41931 functions that repeatedly call the *waitpid()* function with the WNOHANG flag set, acting on  
41932 each child for which status is returned, until *waitpid()* returns zero. If stopped children are not of  
41933 interest, the use of the SA\_NOCLDSTOP flag can prevent the overhead from invoking the  
41934 signal-catching routine when they stop.

41935 Some historical implementations also define other mechanisms for stopping processes, such as  
41936 the *ptrace()* function. These implementations usually do not generate a SIGCHLD signal when  
41937 processes stop due to this mechanism; however, that is beyond the scope of this volume of  
41938 IEEE Std 1003.1-200x.

41939 This volume of IEEE Std 1003.1-200x requires that calls to *sigaction()* that supply a NULL *act*  
41940 argument succeed, even in the case of signals that cannot be caught or ignored (that is, SIGKILL  
41941 or SIGSTOP). The System V *signal()* and BSD *sigvec()* functions return [EINVAL] in these cases  
41942 and, in this respect, their behavior varies from *sigaction()*.

41943 This volume of IEEE Std 1003.1-200x requires that *sigaction()* properly save and restore a signal  
41944 action set up by the ISO C standard *signal()* function. However, there is no guarantee that the  
41945 reverse is true, nor could there be given the greater amount of information conveyed by the  
41946 **sigaction** structure. Because of this, applications should avoid using both functions for the same  
41947 signal in the same process. Since this cannot always be avoided in case of general-purpose  
41948 library routines, they should always be implemented with *sigaction()*.

41949 It was intended that the *signal()* function should be implementable as a library routine using  
41950 *sigaction()*.

41951 The POSIX Realtime Extension extends the *sigaction()* function as specified by the POSIX.1-1990  
 41952 standard to allow the application to request on a per-signal basis via an additional signal action  
 41953 flag that the extra parameters, including the application-defined signal value, if any, be passed  
 41954 to the signal-catching function.

#### 41955 FUTURE DIRECTIONS

41956 None.

#### 41957 SEE ALSO

41958 Section 2.4 (on page 478), *bsd\_signal()*, *kill()*, *\_longjmp()*, *longjmp()*, *raise()*, *semget()*, *sem\_init()*,  
 41959 *sem\_open()*, *sigaddset()*, *sigaltstack()*, *sigdelset()*, *sigemptyset()*, *sigfillset()*, *sigismember()*, *signal()*,  
 41960 *sigprocmask()*, *sigsuspend()*, *wait()*, *waitid()*, *waitpid()*, the Base Definitions volume of  
 41961 IEEE Std 1003.1-200x, <**signal.h**>, <**ucontext.h**>

#### 41962 CHANGE HISTORY

41963 First released in Issue 3.

41964 Entry included for alignment with the POSIX.1-1988 standard.

#### 41965 Issue 5

41966 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and POSIX  
 41967 Threads Extension.

41968 In the DESCRIPTION, the second argument to *func* when SA\_SIGINFO is set is no longer  
 41969 permitted to be NULL, and the description of permitted **siginfo\_t** contents is expanded by  
 41970 reference to <**signal.h**>.

41971 Since the X/OPEN UNIX Extension functionality is now folded into the BASE, the [ENOTSUP]  
 41972 error is deleted.

#### 41973 Issue 6

41974 The Open Group Corrigendum U028/7 is applied. In the paragraph entitled “Signal Effects on  
 41975 Other Functions”, a reference to *sigpending()* is added.

41976 In the DESCRIPTION, the text “Signal Generation and Delivery”, “Signal Actions”, and “Signal  
 41977 Effects on Other Functions” are moved to a separate section of this volume of  
 41978 IEEE Std 1003.1-200x.

41979 Text describing functionality from the Realtime Signals option is marked.

41980 The following changes are made for alignment with the ISO POSIX-1: 1996 standard:

- 41981 • The [ENOTSUP] error condition is added.

41982 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

41983 The **restrict** keyword is added to the *sigaction()* prototype for alignment with the  
 41984 ISO/IEC 9899: 1999 standard.

41985 References to the *wait3()* function are removed.

41986 The SYNOPSIS is marked CX since the presence of this function in the <**signal.h**> header is an  
 41987 extension over the ISO C standard.

41988 **NAME**

41989 sigaddset — add a signal to a signal set

41990 **SYNOPSIS**41991 **CX** #include <signal.h>

41992 int sigaddset(sigset\_t \*set, int signo);

41993

41994 **DESCRIPTION**41995 The *sigaddset()* function adds the individual signal specified by the *signo* to the signal set pointed  
41996 to by *set*.41997 Applications shall call either *sigemptyset()* or *sigfillset()* at least once for each object of type  
41998 **sigset\_t** prior to any other use of that object. If such an object is not initialized in this way, but is  
41999 nonetheless supplied as an argument to any of *pthread\_sigmask()*, *sigaction()*, *sigaddset()*,  
42000 *sigdelset()*, *sigismember()*, *sigpending()*, *sigprocmask()*, *sigsuspend()*, *sigtimedwait()*, *sigwait()*, or  
42001 *sigwaitinfo()*, the results are undefined.42002 **RETURN VALUE**42003 Upon successful completion, *sigaddset()* shall return 0; otherwise, it shall return -1 and set *errno*  
42004 to indicate the error.42005 **ERRORS**42006 The *sigaddset()* function may fail if:42007 [EINVAL] The value of the *signo* argument is an invalid or unsupported signal number.42008 **EXAMPLES**

42009 None.

42010 **APPLICATION USAGE**

42011 None.

42012 **RATIONALE**

42013 None.

42014 **FUTURE DIRECTIONS**

42015 None.

42016 **SEE ALSO**42017 Section 2.4 (on page 478), *sigaction()*, *sigdelset()*, *sigemptyset()*, *sigfillset()*, *sigismember()*,  
42018 *sigpending()*, *sigprocmask()*, *sigsuspend()*, the Base Definitions volume of IEEE Std 1003.1-200x,  
42019 <**signal.h**>42020 **CHANGE HISTORY**

42021 First released in Issue 3.

42022 Entry included for alignment with the POSIX.1-1988 standard.

42023 **Issue 5**42024 The last paragraph of the DESCRIPTION was included as an APPLICATION USAGE note in  
42025 previous issues.42026 **Issue 6**

42027 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

42028 The SYNOPSIS is marked CX since the presence of this function in the <**signal.h**> header is an  
42029 extension over the ISO C standard.

42030 **NAME**

42031 sigaltstack — set and get signal alternate stack context

42032 **SYNOPSIS**42033 XSI 

```
#include <signal.h>
```

42034 

```
int sigaltstack(const stack_t *restrict ss, stack_t *restrict oss);
```

42035

42036 **DESCRIPTION**

42037 The *sigaltstack()* function allows a process to define and examine the state of an alternate stack  
 42038 for signal handlers. Signals that have been explicitly declared to execute on the alternate stack  
 42039 shall be delivered on the alternate stack.

42040 If *ss* is not a null pointer, it points to a **stack\_t** structure that specifies the alternate signal stack  
 42041 that shall take effect upon return from *sigaltstack()*. The *ss\_flags* member specifies the new stack  
 42042 state. If it is set to `SS_DISABLE`, the stack is disabled and *ss\_sp* and *ss\_size* are ignored.  
 42043 Otherwise, the stack shall be enabled, and the *ss\_sp* and *ss\_size* members specify the new address  
 42044 and size of the stack.

42045 The range of addresses starting at *ss\_sp* up to but not including *ss\_sp+ss\_size*, is available to the  
 42046 implementation for use as the stack. This function makes no assumptions regarding which end  
 42047 is the stack base and in which direction the stack grows as items are pushed.

42048 If *oss* is not a null pointer, on successful completion it shall point to a **stack\_t** structure that  
 42049 specifies the alternate signal stack that was in effect prior to the call to *sigaltstack()*. The *ss\_sp*  
 42050 and *ss\_size* members specify the address and size of that stack. The *ss\_flags* member specifies the  
 42051 stack's state, and may contain one of the following values:

42052 **SS\_ONSTACK** The process is currently executing on the alternate signal stack. Attempts to  
 42053 modify the alternate signal stack while the process is executing on it fail. This  
 42054 flag shall not be modified by processes.

42055 **SS\_DISABLE** The alternate signal stack is currently disabled.

42056 The value `SIGSTKSZ` is a system default specifying the number of bytes that would be used to  
 42057 cover the usual case when manually allocating an alternate stack area. The value `MINSIGSTKSZ`  
 42058 is defined to be the minimum stack size for a signal handler. In computing an alternate stack  
 42059 size, a program should add that amount to its stack requirements to allow for the system  
 42060 implementation overhead. The constants `SS_ONSTACK`, `SS_DISABLE`, `SIGSTKSZ`, and  
 42061 `MINSIGSTKSZ` are defined in `<signal.h>`.

42062 After a successful call to one of the *exec* functions, there are no alternate signal stacks in the new  
 42063 process image.

42064 In some implementations, a signal (whether or not indicated to execute on the alternate stack)  
 42065 shall always execute on the alternate stack if it is delivered while another signal is being caught  
 42066 using the alternate stack.

42067 Use of this function by library threads that are not bound to kernel-scheduled entities results in  
 42068 undefined behavior.

42069 **RETURN VALUE**

42070 Upon successful completion, *sigaltstack()* shall return 0; otherwise, it shall return `-1` and set *errno*  
 42071 to indicate the error.

42072 **ERRORS**

42073 The *sigaltstack()* function shall fail if:

42074 [EINVAL] The *ss* argument is not a null pointer, and the *ss\_flags* member pointed to by *ss*  
42075 contains flags other than *SS\_DISABLE*.

42076 [ENOMEM] The size of the alternate stack area is less than *MINSIGSTKSZ*.

42077 [EPERM] An attempt was made to modify an active stack.

42078 **EXAMPLES**42079 **Allocating Memory for an Alternate Stack**

42080 The following example illustrates a method for allocating memory for an alternate stack.

```
42081 #include <signal.h>
42082 ...
42083 if ((sigstk.ss_sp = malloc(SIGSTKSZ)) == NULL)
42084     /* Error return. */
42085     sigstk.ss_size = SIGSTKSZ;
42086     sigstk.ss_flags = 0;
42087     if (sigaltstack(&sigstk, (stack_t *)0) < 0)
42088         perror("sigaltstack");
```

42089 **APPLICATION USAGE**

42090 On some implementations, stack space is automatically extended as needed. On those  
42091 implementations, automatic extension is typically not available for an alternate stack. If the stack  
42092 overflows, the behavior is undefined.

42093 **RATIONALE**

42094 None.

42095 **FUTURE DIRECTIONS**

42096 None.

42097 **SEE ALSO**

42098 Section 2.4 (on page 478), *sigaction()*, *sigsetjmp()*, the Base Definitions volume of  
42099 IEEE Std 1003.1-200x, <signal.h>

42100 **CHANGE HISTORY**

42101 First released in Issue 4, Version 2.

42102 **Issue 5**

42103 Moved from X/OPEN UNIX extension to BASE.

42104 The last sentence of the DESCRIPTION was included as an APPLICATION USAGE note in  
42105 previous issues.

42106 **Issue 6**

42107 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

42108 The **restrict** keyword is added to the *sigaltstack()* prototype for alignment with the  
42109 ISO/IEC 9899:1999 standard.

42110 **NAME**

42111 sigdelset — delete a signal from a signal set

42112 **SYNOPSIS**

42113 CX #include &lt;signal.h&gt;

42114 int sigdelset(sigset\_t \*set, int signo);

42115

42116 **DESCRIPTION**42117 The *sigdelset()* function deletes the individual signal specified by *signo* from the signal set  
42118 pointed to by *set*.42119 Applications should call either *sigemptyset()* or *sigfillset()* at least once for each object of type  
42120 **sigset\_t** prior to any other use of that object. If such an object is not initialized in this way, but is  
42121 nonetheless supplied as an argument to any of *pthread\_sigmask()*, *sigaction()*, *sigaddset()*,  
42122 *sigdelset()*, *sigismember()*, *sigpending()*, *sigprocmask()*, *sigsuspend()*, *sigtimedwait()*, *sigwait()*, or  
42123 *sigwaitinfo()*, the results are undefined.42124 **RETURN VALUE**42125 Upon successful completion, *sigdelset()* shall return 0; otherwise, it shall return -1 and set *errno*  
42126 to indicate the error.42127 **ERRORS**42128 The *sigdelset()* function may fail if:42129 [EINVAL] The *signo* argument is not a valid signal number, or is an unsupported signal  
42130 number.42131 **EXAMPLES**

42132 None.

42133 **APPLICATION USAGE**

42134 None.

42135 **RATIONALE**

42136 None.

42137 **FUTURE DIRECTIONS**

42138 None.

42139 **SEE ALSO**42140 Section 2.4 (on page 478), *sigaction()*, *sigaddset()*, *sigemptyset()*, *sigfillset()*, *sigismember()*,  
42141 *sigpending()*, *sigprocmask()*, *sigsuspend()*, the Base Definitions volume of IEEE Std 1003.1-200x,  
42142 <signal.h>42143 **CHANGE HISTORY**

42144 First released in Issue 3.

42145 Entry included for alignment with the POSIX.1-1988 standard.

42146 **Issue 5**42147 The last paragraph of the DESCRIPTION was included as an APPLICATION USAGE note in  
42148 previous issues.42149 **Issue 6**42150 The SYNOPSIS is marked CX since the presence of this function in the <signal.h> header is an  
42151 extension over the ISO C standard.

42152 **NAME**

42153 sigemptyset — initialize and empty a signal set

42154 **SYNOPSIS**

42155 cx #include &lt;signal.h&gt;

42156 int sigemptyset(sigset\_t \*set);

42157

42158 **DESCRIPTION**42159 The *sigemptyset()* function initializes the signal set pointed to by *set*, such that all signals defined  
42160 in IEEE Std 1003.1-200x are excluded.42161 **RETURN VALUE**42162 Upon successful completion, *sigemptyset()* shall return 0; otherwise, it shall return -1 and set  
42163 *errno* to indicate the error.42164 **ERRORS**

42165 No errors are defined.

42166 **EXAMPLES**

42167 None.

42168 **APPLICATION USAGE**

42169 None.

42170 **RATIONALE**42171 The implementation of the *sigemptyset()* (or *sigfillset()*) function could quite trivially clear (or  
42172 set) all the bits in the signal set. Alternatively, it would be reasonable to initialize part of the  
42173 structure, such as a version field, to permit binary-compatibility between releases where the size  
42174 of the set varies. For such reasons, either *sigemptyset()* or *sigfillset()* must be called prior to any  
42175 other use of the signal set, even if such use is read-only (for example, as an argument to  
42176 *sigpending()*). This function is not intended for dynamic allocation.42177 The *sigfillset()* and *sigemptyset()* functions require that the resulting signal set include (or  
42178 exclude) all the signals defined in this volume of IEEE Std 1003.1-200x. Although it is outside the  
42179 scope of this volume of IEEE Std 1003.1-200x to place this requirement on signals that are  
42180 implemented as extensions, it is recommended that implementation-defined signals also be  
42181 affected by these functions. However, there may be a good reason for a particular signal not to  
42182 be affected. For example, blocking or ignoring an implementation-defined signal may have  
42183 undesirable side effects, whereas the default action for that signal is harmless. In such a case, it  
42184 would be preferable for such a signal to be excluded from the signal set returned by *sigfillset()*.42185 In early proposals there was no distinction between invalid and unsupported signals (the names  
42186 of optional signals that were not supported by an implementation were not defined by that  
42187 implementation). The [EINVAL] error was thus specified as a required error for invalid signals.  
42188 With that distinction, it is not necessary to require implementations of these functions to  
42189 determine whether an optional signal is actually supported, as that could have a significant  
42190 performance impact for little value. The error could have been required for invalid signals and  
42191 optional for unsupported signals, but this seemed unnecessarily complex. Thus, the error is  
42192 optional in both cases.42193 **FUTURE DIRECTIONS**

42194 None.

42195 **SEE ALSO**

42196 Section 2.4 (on page 478), *sigaction()*, *sigaddset()*, *sigdelset()*, *sigfillset()*, *sigismember()*,  
42197 *sigpending()*, *sigprocmask()*, *sigsuspend()*, the Base Definitions volume of IEEE Std 1003.1-200x,  
42198 **<signal.h>**

42199 **CHANGE HISTORY**

42200 First released in Issue 3.

42201 Entry included for alignment with the POSIX.1-1988 standard. |

42202 **Issue 6** |

42203 The SYNOPSIS is marked CX since the presence of this function in the **<signal.h>** header is an |  
42204 extension over the ISO C standard. |

42205 **NAME**

42206 sigfillset — initialize and fill a signal set

42207 **SYNOPSIS**

42208 CX #include &lt;signal.h&gt;

42209 int sigfillset(sigset\_t \*set);

42210

42211 **DESCRIPTION**42212 The *sigfillset()* function shall initialize the signal set pointed to by *set*, such that all signals  
42213 defined in this volume of IEEE Std 1003.1-200x are included.42214 **RETURN VALUE**42215 Upon successful completion, *sigfillset()* shall return 0; otherwise, it shall return -1 and set *errno*  
42216 to indicate the error.42217 **ERRORS**

42218 No errors are defined.

42219 **EXAMPLES**

42220 None.

42221 **APPLICATION USAGE**

42222 None.

42223 **RATIONALE**42224 Refer to *sigemptyset()* (on page 1848).42225 **FUTURE DIRECTIONS**

42226 None.

42227 **SEE ALSO**42228 Section 2.4 (on page 478), *sigaction()*, *sigaddset()*, *sigdelset()*, *sigemptyset()*, *sigismember()*,  
42229 *sigpending()*, *sigprocmask()*, *sigsuspend()*, the Base Definitions volume of IEEE Std 1003.1-200x,  
42230 <signal.h>42231 **CHANGE HISTORY**

42232 First released in Issue 3.

42233 Entry included for alignment with the POSIX.1-1988 standard.

42234 **Issue 6**42235 The SYNOPSIS is marked CX since the presence of this function in the <signal.h> header is an  
42236 extension over the ISO C standard.

42237 **NAME**

42238 sighold, sigignore, sigpause, sigrelse, sigset — signal management

42239 **SYNOPSIS**

```

42240 XSI #include <signal.h>
42241 int sighold(int sig);
42242 int sigignore(int sig);
42243 int sigpause(int sig);
42244 int sigrelse(int sig);
42245 void (*sigset(int sig, void (*disp)(int)))(int);
42246

```

42247 **DESCRIPTION**

42248 Use of any of these functions is unspecified in a multi-threaded process.

42249 The *sighold()*, *sigignore()*, *sigpause()*, *sigrelse()*, and *sigset()* functions provide simplified signal  
42250 management.

42251 The *sigset()* function shall modify signal dispositions. The *sig* argument specifies the signal, |  
42252 which may be any signal except SIGKILL and SIGSTOP. The *disp* argument specifies the signal's |  
42253 disposition, which may be SIG\_DFL, SIG\_IGN, or the address of a signal handler. If *sigset()* is |  
42254 used, and *disp* is the address of a signal handler, the system shall add *sig* to the calling process' |  
42255 signal mask before executing the signal handler; when the signal handler returns, the system |  
42256 shall restore the calling process' signal mask to its state prior to the delivery of the signal. In |  
42257 addition, if *sigset()* is used, and *disp* is equal to SIG\_HOLD, *sig* shall be added to the calling |  
42258 process' signal mask and *sig*'s disposition shall remain unchanged. If *sigset()* is used, and *disp* is |  
42259 not equal to SIG\_HOLD, *sig* shall be removed from the calling process' signal mask.

42260 The *sighold()* function shall add *sig* to the calling process' signal mask. |42261 The *sigrelse()* function shall remove *sig* from the calling process' signal mask. |42262 The *sigignore()* function shall set the disposition of *sig* to SIG\_IGN. |

42263 The *sigpause()* function shall remove *sig* from the calling process' signal mask and suspend the |  
42264 calling process until a signal is received. The *sigpause()* function shall restore the process' signal |  
42265 mask to its original state before returning. |

42266 If the action for the SIGCHLD signal is set to SIG\_IGN, child processes of the calling processes |  
42267 shall not be transformed into zombie processes when they terminate. If the calling process |  
42268 subsequently waits for its children, and the process has no unwaited-for children that were |  
42269 transformed into zombie processes, it shall block until all of its children terminate, and *wait()*, |  
42270 *waitid()*, and *waitpid()* shall fail and set *errno* to [ECHILD].

42271 **RETURN VALUE**

42272 Upon successful completion, *sigset()* shall return SIG\_HOLD if the signal had been blocked and |  
42273 the signal's previous disposition if it had not been blocked. Otherwise, SIG\_ERR shall be |  
42274 returned and *errno* set to indicate the error.

42275 The *sigpause()* function shall suspend execution of the thread until a signal is received, |  
42276 whereupon it shall return -1 and set *errno* to [EINTR].

42277 For all other functions, upon successful completion, 0 shall be returned. Otherwise, -1 shall be |  
42278 returned and *errno* set to indicate the error.

42279 **ERRORS**

42280 These functions shall fail if:

42281 [EINVAL] The *sig* argument is an illegal signal number.

42282 The *sigset()* and *sigignore()* functions shall fail if:

42283 [EINVAL] An attempt is made to catch a signal that cannot be caught, or to ignore a  
42284 signal that cannot be ignored.

42285 **EXAMPLES**

42286 None.

42287 **APPLICATION USAGE**

42288 The *sigaction()* function provides a more comprehensive and reliable mechanism for controlling  
42289 signals; new applications should use *sigaction()* rather than *sigset()*.

42290 The *sighold()* function, in conjunction with *sigrelse()* or *sigpause()*, may be used to establish  
42291 critical regions of code that require the delivery of a signal to be temporarily deferred.

42292 The *sigsuspend()* function should be used in preference to *sigpause()* for broader portability.

42293 **RATIONALE**

42294 None.

42295 **FUTURE DIRECTIONS**

42296 None.

42297 **SEE ALSO**

42298 Section 2.4 (on page 478), *exec*, *pause()*, *sigaction()*, *signal()*, *sigsuspend()*, *waitid()*, the Base  
42299 Definitions volume of IEEE Std 1003.1-200x, <**signal.h**>

42300 **CHANGE HISTORY**

42301 First released in Issue 4 Version 2.

42302 **Issue 5**

42303 Moved from X/OPEN UNIX extension to BASE.

42304 The DESCRIPTION is updated to indicate that the *sigpause()* function restores the process'  
42305 signal mask to its original state before returning.

42306 The RETURN VALUE section is updated to indicate that the *sigpause()* function suspends  
42307 execution of the process until a signal is received, whereupon it returns -1 and sets *errno* to  
42308 [EINTR].

42309 **Issue 6**

42310 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

42311 References to the *wait3()* function are removed.

42312 The XSI functions are split out into their own reference page.

42313 **NAME**

42314 sigignore — signal management

42315 **SYNOPSIS**42316 XSI `#include <signal.h>`42317 `int sigignore(int sig);`

42318

42319 **DESCRIPTION**42320 Refer to *sighold()*.

## 42321 NAME

42322 siginterrupt — allow signals to interrupt functions

## 42323 SYNOPSIS

42324 XSI #include &lt;signal.h&gt;

42325 int siginterrupt(int sig, int flag);

42326

## 42327 DESCRIPTION

42328 The *siginterrupt()* function shall change the restart behavior when a function is interrupted by |  
 42329 the specified signal. The function *siginterrupt(sig, flag)* has an effect as if implemented as:

```
42330 siginterrupt(int sig, int flag) {
42331     int ret;
42332     struct sigaction act;
42333
42334     (void) sigaction(sig, NULL, &act);
42335     if (flag)
42336         act.sa_flags &= ~SA_RESTART;
42337     else
42338         act.sa_flags |= SA_RESTART;
42339     ret = sigaction(sig, &act, NULL);
42340     return ret;
42341 }
```

## 42341 RETURN VALUE

42342 Upon successful completion, *siginterrupt()* shall return 0; otherwise, -1 shall be returned and  
 42343 *errno* set to indicate the error.

## 42344 ERRORS

42345 The *siginterrupt()* function shall fail if:42346 [EINVAL] The *sig* argument is not a valid signal number.

## 42347 EXAMPLES

42348 None.

## 42349 APPLICATION USAGE

42350 The *siginterrupt()* function supports programs written to historical system interfaces. A |  
 42351 conforming application, when being written or rewritten, should use *sigaction()* with the |  
 42352 SA\_RESTART flag instead of *siginterrupt()*.

## 42353 RATIONALE

42354 None.

## 42355 FUTURE DIRECTIONS

42356 None.

## 42357 SEE ALSO

42358 Section 2.4 (on page 478), *sigaction()*, the Base Definitions volume of IEEE Std 1003.1-200x,  
 42359 <signal.h>

## 42360 CHANGE HISTORY

42361 First released in Issue 4, Version 2.

42362 **Issue 5**

42363 Moved from X/OPEN UNIX extension to BASE.

42364 **NAME**

42365 sigismember — test for a signal in a signal set

42366 **SYNOPSIS**42367 **CX** #include <signal.h>

42368 int sigismember(const sigset\_t \*set, int signo);

42369

42370 **DESCRIPTION**42371 The *sigismember()* function shall test whether the signal specified by *signo* is a member of the set  
42372 pointed to by *set*.42373 Applications should call either *sigemptyset()* or *sigfillset()* at least once for each object of type  
42374 **sigset\_t** prior to any other use of that object. If such an object is not initialized in this way, but is  
42375 nonetheless supplied as an argument to any of *pthread\_sigmask()*, *sigaction()*, *sigaddset()*,  
42376 *sigdelset()*, *sigismember()*, *sigpending()*, *sigprocmask()*, *sigsuspend()*, *sigtimedwait()*, *sigwait()*, or  
42377 *sigwaitinfo()*, the results are undefined.42378 **RETURN VALUE**42379 Upon successful completion, *sigismember()* shall return 1 if the specified signal is a member of  
42380 the specified set, or 0 if it is not. Otherwise, it shall return -1 and set *errno* to indicate the error.42381 **ERRORS**42382 The *sigismember()* function may fail if:42383 [EINVAL] The *signo* argument is not a valid signal number, or is an unsupported signal  
42384 number.42385 **EXAMPLES**

42386 None.

42387 **APPLICATION USAGE**

42388 None.

42389 **RATIONALE**

42390 None.

42391 **FUTURE DIRECTIONS**

42392 None.

42393 **SEE ALSO**42394 Section 2.4 (on page 478), *sigaction()*, *sigaddset()*, *sigdelset()*, *sigfillset()*, *sigemptyset()*,  
42395 *sigpending()*, *sigprocmask()*, *sigsuspend()*, the Base Definitions volume of IEEE Std 1003.1-200x,  
42396 <signal.h>42397 **CHANGE HISTORY**

42398 First released in Issue 3.

42399 Entry included for alignment with the POSIX.1-1988 standard.

42400 **Issue 5**42401 The last paragraph of the DESCRIPTION was included as an APPLICATION USAGE note in  
42402 previous issues.42403 **Issue 6**42404 The SYNOPSIS is marked CX since the presence of this function in the <signal.h> header is an  
42405 extension over the ISO C standard.

42406 **NAME**

42407 siglongjmp — non-local goto with signal handling

42408 **SYNOPSIS**

42409 CX #include &lt;setjmp.h&gt;

42410 void siglongjmp(sigjmp\_buf env, int val);

42411

42412 **DESCRIPTION**42413 The *siglongjmp()* function shall be equivalent to the *longjmp()* function, except as follows:

- 42414 • References to *setjmp()* shall be equivalent to *sigsetjmp()*.
- 42415 • The *siglongjmp()* function shall restore the saved signal mask if and only if the *env* argument
- 42416 was initialized by a call to *sigsetjmp()* with a non-zero *savemask* argument.

42417 **RETURN VALUE**

42418 After *siglongjmp()* is completed, program execution shall continue as if the corresponding  
 42419 invocation of *sigsetjmp()* had just returned the value specified by *val*. The *siglongjmp()* function  
 42420 shall not cause *sigsetjmp()* to return 0; if *val* is 0, *sigsetjmp()* shall return the value 1.

42421 **ERRORS**

42422 No errors are defined.

42423 **EXAMPLES**

42424 None.

42425 **APPLICATION USAGE**

42426 The distinction between *setjmp()* or *longjmp()* and *sigsetjmp()* or *siglongjmp()* is only significant  
 42427 for programs which use *sigaction()*, *sigprocmask()*, or *sigsuspend()*.

42428 **RATIONALE**

42429 None.

42430 **FUTURE DIRECTIONS**

42431 None.

42432 **SEE ALSO**

42433 *longjmp()*, *setjmp()*, *sigprocmask()*, *sigsetjmp()*, *sigsuspend()*, the Base Definitions volume of  
 42434 IEEE Std 1003.1-200x, <setjmp.h>

42435 **CHANGE HISTORY**

42436 First released in Issue 3.

42437 Entry included for alignment with the ISO POSIX-1 standard.

42438 **Issue 5**

42439 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

42440 **Issue 6**42441 The DESCRIPTION is rewritten in terms of *longjmp()*.

42442 The SYNOPSIS is marked CX since the presence of this function in the <setjmp.h> header is an  
 42443 extension over the ISO C standard.

## 42444 NAME

42445 signal — signal management

## 42446 SYNOPSIS

42447 #include &lt;signal.h&gt;

42448 void (\*signal(int sig, void (\*func)(int)))(int);

## 42449 DESCRIPTION

42450 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 42451 conflict between the requirements described here and the ISO C standard is unintentional. This  
 42452 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

42453 CX Use of this function is unspecified in a multi-threaded process.

42454 The *signal()* function chooses one of three ways in which receipt of the signal number *sig* is to be  
 42455 subsequently handled. If the value of *func* is SIG\_DFL, default handling for that signal shall  
 42456 occur. If the value of *func* is SIG\_IGN, the signal shall be ignored. Otherwise, the application  
 42457 shall ensure that *func* points to a function to be called when that signal occurs. An invocation of  
 42458 such a function because of a signal, or (recursively) of any further functions called by that  
 42459 invocation (other than functions in the standard library), is called a “signal handler”.

42460 When a signal occurs, and *func* points to a function, it is implementation-defined whether the  
 42461 equivalent of a:

42462 `signal(sig, SIG_DFL);`

42463 is executed or the implementation prevents some implementation-defined set of signals (at least  
 42464 including *sig*) from occurring until the current signal handling has completed. (If the value of *sig*  
 42465 is SIGILL, the implementation may alternatively define that no action is taken.) Next the  
 42466 equivalent of:

42467 `(*func)(sig);`

42468 is executed. If and when the function returns, if the value of *sig* was SIGFPE, SIGILL, or  
 42469 SIGSEGV or any other implementation-defined value corresponding to a computational  
 42470 exception, the behavior is undefined. Otherwise, the program shall resume execution at the  
 42471 CX point it was interrupted. If the signal occurs as the result of calling the *abort()*, *raise()*, *kill()*,  
 42472 *pthread\_kill()*, or *sigqueue()* function, the signal handler shall not call the *raise()* function.

42473 CX If the signal occurs other than as the result of calling *abort()*, *raise()*, *kill()*, *pthread\_kill()*, or  
 42474 *sigqueue()*, the behavior is undefined if the signal handler refers to any object with static storage  
 42475 duration other than by assigning a value to an object declared as volatile **sig\_atomic\_t**, or if the  
 42476 signal handler calls any function in the standard library other than one of the functions listed in  
 42477 Section 2.4 (on page 478). Furthermore, if such a call fails, the value of *errno* is unspecified.

42478 At program start-up, the equivalent of:

42479 `signal(sig, SIG_IGN);`

42480 is executed for some signals, and the equivalent of:

42481 `signal(sig, SIG_DFL);`

42482 CX is executed for all other signals (see *exec*).

## 42483 RETURN VALUE

42484 If the request can be honored, *signal()* shall return the value of *func* for the most recent call to  
 42485 *signal()* for the specified signal *sig*. Otherwise, SIG\_ERR shall be returned and a positive value  
 42486 shall be stored in *errno*.

42487 **ERRORS**42488 The *signal()* function shall fail if:

42489 CX [EINVAL] The *sig* argument is not a valid signal number or an attempt is made to catch a  
 42490 signal that cannot be caught or ignore a signal that cannot be ignored.

42491 The *signal()* function may fail if:

42492 CX [EINVAL] An attempt was made to set the action to SIG\_DFL for a signal that cannot be  
 42493 caught or ignored (or both).

42494 **EXAMPLES**

42495 None.

42496 **APPLICATION USAGE**

42497 The *sigaction()* function provides a more comprehensive and reliable mechanism for controlling  
 42498 signals; new applications should use *sigaction()* rather than *signal()*.

42499 **RATIONALE**

42500 None.

42501 **FUTURE DIRECTIONS**

42502 None.

42503 **SEE ALSO**

42504 Section 2.4 (on page 478), *exec*, *pause()*, *sigaction()*, *sigsuspend()*, *waitid()*, the Base Definitions  
 42505 volume of IEEE Std 1003.1-200x, <**signal.h**>

42506 **CHANGE HISTORY**

42507 First released in Issue 1. Derived from Issue 1 of the SVID.

42508 **Issue 5**

42509 Moved from X/OPEN UNIX extension to BASE.

42510 The DESCRIPTION is updated to indicate that the *sigpause()* function restores the process'  
 42511 signal mask to its original state before returning.

42512 The RETURN VALUE section is updated to indicate that the *sigpause()* function suspends  
 42513 execution of the process until a signal is received, whereupon it returns -1 and sets *errno* to  
 42514 [EINTR].

42515 **Issue 6**

42516 Extensions beyond the ISO C standard are now marked.

42517 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

42518 The DESCRIPTION is updated for alignment with the ISO/IEC 9899:1999 standard.

42519 References to the *wait3()* function are removed.

42520 The *sighold()*, *sigignore()*, *sigrelse()*, and *sigset()* functions are split out onto their own reference  
 42521 page.

42522 **NAME**

42523 signbit — test sign

42524 **SYNOPSIS**

42525 #include &lt;math.h&gt;

42526 int signbit(real-floating x);

42527 **DESCRIPTION**

42528 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
42529 conflict between the requirements described here and the ISO C standard is unintentional. This  
42530 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

42531 The *signbit()* macro shall determine whether the sign of its argument value is negative. NaNs,  
42532 zeros, and infinities have a sign bit.

42533 **RETURN VALUE**

42534 The *signbit()* macro shall return a non-zero value if and only if the sign of its argument value is  
42535 negative.

42536 **ERRORS**

42537 No errors are defined.

42538 **EXAMPLES**

42539 None.

42540 **APPLICATION USAGE**

42541 None.

42542 **RATIONALE**

42543 None.

42544 **FUTURE DIRECTIONS**

42545 None.

42546 **SEE ALSO**

42547 *fpclassify()*, *isfinite()*, *isinf()*, *isnan()*, *isnormal()*, the Base Definitions volume of  
42548 IEEE Std 1003.1-200x, <math.h>

42549 **CHANGE HISTORY**

42550 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

42551 **NAME**

42552 sigpause — remove a signal from the signal mask and suspend the thread

42553 **SYNOPSIS**

42554 XSI #include <signal.h>

42555 int sigpause(int sig);

42556

42557 **DESCRIPTION**

42558 Refer to *sighold()*.

42559 **NAME**

42560 sigpending — examine pending signals

42561 **SYNOPSIS**42562 **CX** #include <signal.h>

42563 int sigpending(sigset\_t \*set);

42564

42565 **DESCRIPTION**

42566 The *sigpending()* function shall store, in the location referenced by the *set* argument, the set of  
42567 signals that are blocked from delivery to the calling thread and that are pending on the process  
42568 or the calling thread.

42569 **RETURN VALUE**

42570 Upon successful completion, *sigpending()* shall return 0; otherwise, -1 shall be returned and  
42571 *errno* set to indicate the error.

42572 **ERRORS**

42573 No errors are defined.

42574 **EXAMPLES**

42575 None.

42576 **APPLICATION USAGE**

42577 None.

42578 **RATIONALE**

42579 None.

42580 **FUTURE DIRECTIONS**

42581 None.

42582 **SEE ALSO**

42583 *sigaddset()*, *sigdelset()*, *sigemptyset()*, *sigfillset()*, *sigismember()*, *sigprocmask()*, the Base Definitions  
42584 volume of IEEE Std 1003.1-200x, <signal.h>

42585 **CHANGE HISTORY**

42586 First released in Issue 3.

42587 **Issue 5**

42588 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

42589 **Issue 6**

42590 The SYNOPSIS is marked CX since the presence of this function in the <signal.h> header is an  
42591 extension over the ISO C standard.

42592 **NAME**

42593 sigprocmask — examine and change blocked signals

42594 **SYNOPSIS**

42595 cx #include &lt;signal.h&gt; |

42596 int sigprocmask(int *how*, const sigset\_t \*restrict *set*,  
42597 sigset\_t \*restrict *oset*);

42598 |

42599 **DESCRIPTION**42600 Refer to *pthread\_sigmask()*.

## 42601 NAME

42602 sigqueue — queue a signal to a process (**REALTIME**)

## 42603 SYNOPSIS

42604 RTS #include &lt;signal.h&gt;

42605 int sigqueue(pid\_t pid, int signo, const union sigval value);

42606

## 42607 DESCRIPTION

42608 The *sigqueue()* function shall cause the signal specified by *signo* to be sent with the value  
 42609 specified by *value* to the process specified by *pid*. If *signo* is zero (the null signal), error checking  
 42610 is performed but no signal is actually sent. The null signal can be used to check the validity of  
 42611 *pid*.

42612 The conditions required for a process to have permission to queue a signal to another process  
 42613 are the same as for the *kill()* function.

42614 The *sigqueue()* function shall return immediately. If SA\_SIGINFO is set for *signo* and if the  
 42615 resources were available to queue the signal, the signal shall be queued and sent to the receiving  
 42616 process. If SA\_SIGINFO is not set for *signo*, then *signo* shall be sent at least once to the receiving  
 42617 process; it is unspecified whether *value* shall be sent to the receiving process as a result of this  
 42618 call.

42619 If the value of *pid* causes *signo* to be generated for the sending process, and if *signo* is not blocked  
 42620 for the calling thread and if no other thread has *signo* unblocked or is waiting in a *sigwait()*  
 42621 function for *signo*, either *signo* or at least the pending, unblocked signal shall be delivered to the  
 42622 calling thread before the *sigqueue()* function returns. Should any multiple pending signals in the  
 42623 range SIGRTMIN to SIGRTMAX be selected for delivery, it shall be the lowest numbered one.  
 42624 The selection order between realtime and non-realtime signals, or between multiple pending  
 42625 non-realtime signals, is unspecified.

## 42626 RETURN VALUE

42627 Upon successful completion, the specified signal shall have been queued, and the *sigqueue()*  
 42628 function shall return a value of zero. Otherwise, the function shall return a value of  $-1$  and set  
 42629 *errno* to indicate the error.

## 42630 ERRORS

42631 The *sigqueue()* function shall fail if:

42632 [EAGAIN] No resources available to queue the signal. The process has already queued  
 42633 SIGQUEUE\_MAX signals that are still pending at the receiver(s), or a system-  
 42634 wide resource limit has been exceeded.

42635 [EINVAL] The value of the *signo* argument is an invalid or unsupported signal number.

42636 [EPERM] The process does not have the appropriate privilege to send the signal to the  
 42637 receiving process.

42638 [ESRCH] The process *pid* does not exist.

42639 **EXAMPLES**

42640 None.

42641 **APPLICATION USAGE**

42642 None.

42643 **RATIONALE**

42644 The *sigqueue()* function allows an application to queue a realtime signal to itself or to another  
42645 process, specifying the application-defined value. This is common practice in realtime  
42646 applications on existing realtime systems. It was felt that specifying another function in the  
42647 *sig...* name space already carved out for signals was preferable to extending the interface to  
42648 *kill()*.

42649 Such a function became necessary when the put/get event function of the message queues was  
42650 removed. It should be noted that the *sigqueue()* function implies reduced performance in a  
42651 security-conscious implementation as the access permissions between the sender and receiver  
42652 have to be checked on each send when the *pid* is resolved into a target process. Such access  
42653 checks were necessary only at message queue open in the previous interface.

42654 The standard developers required that *sigqueue()* have the same semantics with respect to the  
42655 null signal as *kill()*, and that the same permission checking be used. But because of the difficulty  
42656 of implementing the “broadcast” semantic of *kill()* (for example, to process groups) and the  
42657 interaction with resource allocation, this semantic was not adopted. The *sigqueue()* function  
42658 queues a signal to a single process specified by the *pid* argument.

42659 The *sigqueue()* function can fail if the system has insufficient resources to queue the signal. An  
42660 explicit limit on the number of queued signals that a process could send was introduced. While  
42661 the limit is “per-sender”, this volume of IEEE Std 1003.1-200x does not specify that the resources  
42662 be part of the state of the sender. This would require either that the sender be maintained after  
42663 exit until all signals that it had sent to other processes were handled or that all such signals that  
42664 had not yet been acted upon be removed from the queue(s) of the receivers. This volume of  
42665 IEEE Std 1003.1-200x does not preclude this behavior, but an implementation that allocated  
42666 queuing resources from a system-wide pool (with per-sender limits) and that leaves queued  
42667 signals pending after the sender exits is also permitted.

42668 **FUTURE DIRECTIONS**

42669 None.

42670 **SEE ALSO**42671 Section 2.8.1 (on page 491), the Base Definitions volume of IEEE Std 1003.1-200x, <**signal.h**>42672 **CHANGE HISTORY**

42673 First released in Issue 5. Included for alignment with the POSIX Realtime Extension and the  
42674 POSIX Threads Extension.

42675 **Issue 6**42676 The *sigqueue()* function is marked as part of the Realtime Signals Extension option.

42677 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
42678 implementation does not support the Realtime Signals Extension option.

42679 **NAME**

42680 sigrelse — remove a signal from signal mask or modify signal disposition

42681 **SYNOPSIS**

42682 XSI #include <signal.h>

42683 int sigrelse(int sig);

42684

42685 **DESCRIPTION**

42686 Refer to *sighold()*.

42687 **NAME**

42688 sigset — signal management |

42689 **SYNOPSIS**

42690 #include &lt;signal.h&gt;

42691 XSI void (\*sigset(int sig, void (\*disp)(int)))(int);

42692

42693 **DESCRIPTION**42694 Refer to *sighold()*.

42695 **NAME**

42696 sigsetjmp — set jump point for a non-local goto

42697 **SYNOPSIS**42698 `CX #include <setjmp.h>`42699 `int sigsetjmp(sigjmp_buf env, int savemask);`

42700

42701 **DESCRIPTION**42702 The *sigsetjmp()* function shall be equivalent to the *setjmp()* function, except as follows:

- 42703 • References to *setjmp()* are equivalent to *sigsetjmp()*.
- 42704 • References to *longjmp()* are equivalent to *siglongjmp()*.
- 42705 • If the value of the *savemask* argument is not 0, *sigsetjmp()* shall also save the current signal mask of the calling thread as part of the calling environment.

42707 **RETURN VALUE**

42708 If the return is from a successful direct invocation, *sigsetjmp()* shall return 0. If the return is from  
 42709 a call to *siglongjmp()*, *sigsetjmp()* shall return a non-zero value.

42710 **ERRORS**

42711 No errors are defined.

42712 **EXAMPLES**

42713 None.

42714 **APPLICATION USAGE**

42715 The distinction between *setjmp()/longjmp()* and *sigsetjmp()/siglongjmp()* is only significant for  
 42716 programs which use *sigaction()*, *sigprocmask()*, or *sigsuspend()*.

42717 Note that since this function is defined in terms of *setjmp()*, if *savemask* is zero, it is unspecified  
 42718 whether the signal mask is saved.

42719 **RATIONALE**

42720 The ISO C standard specifies various restrictions on the usage of the *setjmp()* macro in order to  
 42721 permit implementors to recognize the name in the compiler and not implement an actual  
 42722 function. These same restrictions apply to the *sigsetjmp()* macro.

42723 There are processors that cannot easily support these calls, but this was not considered a  
 42724 sufficient reason to exclude them.

42725 4.2 BSD, 4.3 BSD, and XSI-conformant systems provide functions named *\_setjmp()* and  
 42726 *\_longjmp()* that, together with *setjmp()* and *longjmp()*, provide the same functionality as  
 42727 *sigsetjmp()* and *siglongjmp()*. On those systems, *setjmp()* and *longjmp()* save and restore signal  
 42728 masks, while *\_setjmp()* and *\_longjmp()* do not. On System V, Release 3 and in corresponding  
 42729 issues of the SVID, *setjmp()* and *longjmp()* are explicitly defined not to save and restore signal  
 42730 masks. In order to permit existing practice in both cases, the relation of *setjmp()* and *longjmp()* to  
 42731 signal masks is not specified, and a new set of functions is defined instead.

42732 The *longjmp()* and *siglongjmp()* functions operate as in the previous issue provided the matching  
 42733 *setjmp()* or *sigsetjmp()* has been performed in the same thread. Non-local jumps into contexts  
 42734 saved by other threads would be at best a questionable practice and were not considered worthy  
 42735 of standardization.

42736 **FUTURE DIRECTIONS**

42737 None.

42738 **SEE ALSO**42739 *siglongjmp()*, *signal()*, *sigprocmask()*, *sigsuspend()*, the Base Definitions volume of  
42740 IEEE Std 1003.1-200x, <**setjmp.h**>42741 **CHANGE HISTORY**

42742 First released in Issue 3.

42743 Entry included for alignment with the POSIX.1-1988 standard.

42744 **Issue 5**

42745 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

42746 **Issue 6**42747 The DESCRIPTION is reworded in terms of *setjmp()*. |42748 The SYNOPSIS is marked CX since the presence of this function in the <**setjmp.h**> header is an |  
42749 extension over the ISO C standard. |

42750 **NAME**

42751 sigsuspend — wait for a signal

42752 **SYNOPSIS**

42753 cx #include &lt;signal.h&gt;

42754 int sigsuspend(const sigset\_t \*sigmask);

42755

42756 **DESCRIPTION**

42757 The *sigsuspend()* function shall replace the current signal mask of the calling thread with the set  
42758 of signals pointed to by *sigmask* and then suspend the thread until delivery of a signal whose  
42759 action is either to execute a signal-catching function or to terminate the process. This shall not  
42760 cause any other signals that may have been pending on the process to become pending on the  
42761 thread.

42762 If the action is to terminate the process then *sigsuspend()* shall never return. If the action is to  
42763 execute a signal-catching function, then *sigsuspend()* shall return after the signal-catching  
42764 function returns, with the signal mask restored to the set that existed prior to the *sigsuspend()*  
42765 call.

42766 It is not possible to block signals that cannot be ignored. This is enforced by the system without  
42767 causing an error to be indicated.

42768 **RETURN VALUE**

42769 Since *sigsuspend()* suspends thread execution indefinitely, there is no successful completion  
42770 return value. If a return occurs, *-1* shall be returned and *errno* set to indicate the error.

42771 **ERRORS**42772 The *sigsuspend()* function shall fail if:

42773 [EINTR] A signal is caught by the calling process and control is returned from the  
42774 signal-catching function.

42775 **EXAMPLES**

42776 None.

42777 **APPLICATION USAGE**

42778 Normally, at the beginning of a critical code section, a specified set of signals is blocked using  
42779 the *sigprocmask()* function. When the thread has completed the critical section and needs to wait  
42780 for the previously blocked signal(s), it pauses by calling *sigsuspend()* with the mask that was  
42781 returned by the *sigprocmask()* call.

42782 **RATIONALE**

42783 None.

42784 **FUTURE DIRECTIONS**

42785 None.

42786 **SEE ALSO**

42787 Section 2.4 (on page 478), *pause()*, *sigaction()*, *sigaddset()*, *sigdelset()*, *sigemptyset()*, *sigfillset()*, the  
42788 Base Definitions volume of IEEE Std 1003.1-200x, <signal.h>

42789 **CHANGE HISTORY**

42790 First released in Issue 3.

42791 Entry included for alignment with the POSIX.1-1988 standard.

42792 **Issue 5**

42793 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

42794 **Issue 6**

42795 The text in the RETURN VALUE section has been changed from “suspends process execution”  
42796 to “suspends thread execution”. This reflects IEEE PASC Interpretation 1003.1c #40.

42797 Text in the APPLICATION USAGE section has been replaced. |

42798 The SYNOPSIS is marked CX since the presence of this function in the <signal.h> header is an |  
42799 extension over the ISO C standard. |

## 42800 NAME

42801 sigtimedwait, sigwaitinfo — wait for queued signals (**REALTIME**)

## 42802 SYNOPSIS

42803 RTS #include &lt;signal.h&gt;

```
42804 int sigtimedwait(const sigset_t *restrict set,
42805                 siginfo_t *restrict info,
42806                 const struct timespec *restrict timeout);
42807 int sigwaitinfo(const sigset_t *restrict set,
42808                 siginfo_t *restrict info);
42809
```

## 42810 DESCRIPTION

42811 The *sigtimedwait()* function shall be equivalent to *sigwaitinfo()* except that if none of the signals  
 42812 specified by *set* are pending, *sigtimedwait()* shall wait for the time interval specified in the  
 42813 **timespec** structure referenced by *timeout*. If the **timespec** structure pointed to by *timeout* is  
 42814 zero-valued and if none of the signals specified by *set* are pending, then *sigtimedwait()* shall  
 42815 MON return immediately with an error. If *timeout* is the NULL pointer, the behavior is unspecified. If  
 42816 the Monotonic Clock option is supported, the CLOCK\_MONOTONIC clock shall be used to  
 42817 measure the time interval specified by the *timeout* argument.

42818 The *sigwaitinfo()* function selects the pending signal from the set specified by *set*. Should any of  
 42819 multiple pending signals in the range SIGRTMIN to SIGRTMAX be selected, it shall be the  
 42820 lowest numbered one. The selection order between realtime and non-realtime signals, or  
 42821 between multiple pending non-realtime signals, is unspecified. If no signal in *set* is pending at  
 42822 the time of the call, the calling thread shall be suspended until one or more signals in *set* become  
 42823 pending or until it is interrupted by an unblocked, caught signal.

42824 The *sigwaitinfo()* function shall be equivalent to the *sigwait()* function if the *info* argument is  
 42825 NULL. If the *info* argument is non-NULL, the *sigwaitinfo()* function shall be equivalent to  
 42826 *sigwait()*, except that the selected signal number shall be stored in the *si\_signo* member, and the  
 42827 cause of the signal shall be stored in the *si\_code* member. If any value is queued to the selected  
 42828 signal, the first such queued value shall be dequeued and, if the *info* argument is non-NULL, the  
 42829 value shall be stored in the *si\_value* member of *info*. The system resource used to queue the  
 42830 signal shall be released and returned to the system for other use. If no value is queued, the  
 42831 content of the *si\_value* member is undefined. If no further signals are queued for the selected  
 42832 signal, the pending indication for that signal shall be reset.

## 42833 RETURN VALUE

42834 Upon successful completion (that is, one of the signals specified by *set* is pending or is  
 42835 generated) *sigwaitinfo()* and *sigtimedwait()* shall return the selected signal number. Otherwise,  
 42836 the function shall return a value of -1 and set *errno* to indicate the error.

## 42837 ERRORS

42838 The *sigtimedwait()* function shall fail if:42839 [EAGAIN] No signal specified by *set* was generated within the specified timeout period.42840 The *sigtimedwait()* and *sigwaitinfo()* functions may fail if:

42841 [EINTR] The wait was interrupted by an unblocked, caught signal. It shall be  
 42842 documented in system documentation whether this error causes these  
 42843 functions to fail.

42844 The *sigtimedwait()* function may also fail if:

42845 [EINVAL] The *timeout* argument specified a *tv\_nsec* value less than zero or greater than  
42846 or equal to 1 000 million.

42847 An implementation only checks for this error if no signal is pending in *set* and it is necessary to  
42848 wait.

#### 42849 EXAMPLES

42850 None.

#### 42851 APPLICATION USAGE

42852 The *sigtimedwait()* function times out and returns an [EAGAIN] error. Application writers  
42853 should note that this is inconsistent with other functions such as *pthread\_cond\_timedwait()* that  
42854 return [ETIMEDOUT].

#### 42855 RATIONALE

42856 Existing programming practice on realtime systems uses the ability to pause waiting for a  
42857 selected set of events and handle the first event that occurs in-line instead of in a signal-handling  
42858 function. This allows applications to be written in an event-directed style similar to a state  
42859 machine. This style of programming is useful for largescale transaction processing in which the  
42860 overall throughput of an application and the ability to clearly track states are more important  
42861 than the ability to minimize the response time of individual event handling.

42862 It is possible to construct a signal-waiting macro function out of the realtime signal function  
42863 mechanism defined in this volume of IEEE Std 1003.1-200x. However, such a macro has to  
42864 include the definition of a generalized handler for all signals to be waited on. A significant  
42865 portion of the overhead of handler processing can be avoided if the signal-waiting function is  
42866 provided by the kernel. This volume of IEEE Std 1003.1-200x therefore provides two signal-  
42867 waiting functions—one that waits indefinitely and one with a timeout—as part of the overall  
42868 realtime signal function specification.

42869 The specification of a function with a timeout allows an application to be written that can be  
42870 broken out of a wait after a set period of time if no event has occurred. It was argued that setting  
42871 a timer event before the wait and recognizing the timer event in the wait would also implement  
42872 the same functionality, but at a lower performance level. Because of the performance  
42873 degradation associated with the user-level specification of a timer event and the subsequent  
42874 cancelation of that timer event after the wait completes for a valid event, and the complexity  
42875 associated with handling potential race conditions associated with the user-level method, the  
42876 separate function has been included.

42877 Note that the semantics of the *sigwaitinfo()* function are nearly identical to that of the *sigwait()*  
42878 function defined by this volume of IEEE Std 1003.1-200x. The only difference is that *sigwaitinfo()*  
42879 returns the queued signal value in the *value* argument. The return of the queued value is  
42880 required so that applications can differentiate between multiple events queued to the same  
42881 signal number.

42882 The two distinct functions are being maintained because some implementations may choose to  
42883 implement the POSIX Threads Extension functions and not implement the queued signals  
42884 extensions. Note, though, that *sigwaitinfo()* does not return the queued value if the *value*  
42885 argument is NULL, so the POSIX Threads Extension *sigwait()* function can be implemented as a  
42886 macro on *sigwaitinfo()*.

42887 The *sigtimedwait()* function was separated from the *sigwaitinfo()* function to address concerns  
42888 regarding the overloading of the *timeout* pointer to indicate indefinite wait (no timeout), timed  
42889 wait, and immediate return, and concerns regarding consistency with other functions where the  
42890 conditional and timed waits were separate functions from the pure blocking function. The  
42891 semantics of *sigtimedwait()* are specified such that *sigwaitinfo()* could be implemented as a  
42892 macro with a NULL pointer for *timeout*.

42893 The *sigwait* functions provide a synchronous mechanism for threads to wait for asynchronously  
 42894 generated signals. One important question was how many threads that are suspended in a call to  
 42895 a *sigwait()* function for a signal should return from the call when the signal is sent. Four choices  
 42896 were considered:

- 42897 1. Return an error for multiple simultaneous calls to *sigwait* functions for the same signal.
- 42898 2. One or more threads return.
- 42899 3. All waiting threads return.
- 42900 4. Exactly one thread returns.

42901 Prohibiting multiple calls to *sigwait()* for the same signal was felt to be overly restrictive. The  
 42902 “one or more” behavior made implementation of conforming packages easy at the expense of  
 42903 forcing POSIX threads clients to protect against multiple simultaneous calls to *sigwait()* in  
 42904 application code in order to achieve predictable behavior. There was concern that the “all  
 42905 waiting threads” behavior would result in “signal broadcast storms”, consuming excessive CPU  
 42906 resources by replicating the signals in the general case. Furthermore, no convincing examples  
 42907 could be presented that delivery to all was either simpler or more powerful than delivery to one.

42908 Thus, the consensus was that exactly one thread that was suspended in a call to a *sigwait*  
 42909 function for a signal should return when that signal occurs. This is not an onerous restriction as:

- 42910 • A multi-way signal wait can be built from the single-way wait.
- 42911 • Signals should only be handled by application-level code, as library routines cannot guess  
 42912 what the application wants to do with signals generated for the entire process.
- 42913 • Applications can thus arrange for a single thread to wait for any given signal and call any  
 42914 needed routines upon its arrival.

42915 In an application that is using signals for interprocess communication, signal processing is  
 42916 typically done in one place. Alternatively, if the signal is being caught so that process cleanup  
 42917 can be done, the signal handler thread can call separate process cleanup routines for each  
 42918 portion of the application. Since the application main line started each portion of the application,  
 42919 it is at the right abstraction level to tell each portion of the application to clean up.

42920 Certainly, there exist programming styles where it is logical to consider waiting for a single  
 42921 signal in multiple threads. A simple *sigwait\_multiple()* routine can be constructed to achieve this  
 42922 goal. A possible implementation would be to have each *sigwait\_multiple()* caller registered as  
 42923 having expressed interest in a set of signals. The caller then waits on a thread-specific condition  
 42924 variable. A single server thread calls a *sigwait()* function on the union of all registered signals.  
 42925 When the *sigwait()* function returns, the appropriate state is set and condition variables are  
 42926 broadcast. New *sigwait\_multiple()* callers may cause the pending *sigwait()* call to be canceled  
 42927 and reissued in order to update the set of signals being waited for.

#### 42928 FUTURE DIRECTIONS

42929 None.

#### 42930 SEE ALSO

42931 Section 2.8.1 (on page 491), *pause()*, *pthread\_sigmask()*, *sigaction()*, *sigpending()*, *sigsuspend()*,  
 42932 *sigwait()*, the Base Definitions volume of IEEE Std 1003.1-200x, <[signal.h](#)>, <[time.h](#)>

#### 42933 CHANGE HISTORY

42934 First released in Issue 5. Included for alignment with the POSIX Realtime Extension and the  
 42935 POSIX Threads Extension.

42936 **Issue 6**

- 42937 These functions are marked as part of the Realtime Signals Extension option.
- 42938 The Open Group Corrigendum U035/3 is applied. The SYNOPSIS of the *sigwaitinfo()* function  
42939 has been corrected so that the second argument is of type **siginfo\_t\***.
- 42940 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
42941 implementation does not support the Realtime Signals Extension option.
- 42942 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that the  
42943 CLOCK\_MONOTONIC clock, if supported, is used to measure timeout intervals.
- 42944 The **restrict** keyword is added to the *sigtimedwait()* and *sigwaitinfo()* prototypes for alignment  
42945 with the ISO/IEC 9899:1999 standard.

42946 **NAME**

42947 sigwait — wait for queued signals

42948 **SYNOPSIS**42949 `CX` #include <signal.h>42950 `int sigwait(const sigset_t *restrict set, int *restrict sig);`

42951

42952 **DESCRIPTION**

42953 The *sigwait()* function shall select a pending signal from *set*, atomically clear it from the system's  
42954 set of pending signals, and return that signal number in the location referenced by *sig*. If prior to  
42955 the call to *sigwait()* there are multiple pending instances of a single signal number, it is  
42956 implementation-defined whether upon successful return there are any remaining pending  
42957 `RTS` signals for that signal number. If the implementation supports queued signals and there are  
42958 multiple signals queued for the signal number selected, the first such queued signal shall cause a  
42959 return from *sigwait()* and the remainder shall remain queued. If no signal in *set* is pending at the  
42960 time of the call, the thread shall be suspended until one or more becomes pending. The signals  
42961 defined by *set* shall have been blocked at the time of the call to *sigwait()*; otherwise, the behavior  
42962 is undefined. The effect of *sigwait()* on the signal actions for the signals in *set* is unspecified.

42963 If more than one thread is using *sigwait()* to wait for the same signal, no more than one of these  
42964 threads shall return from *sigwait()* with the signal number. Which thread returns from *sigwait()*  
42965 if more than a single thread is waiting is unspecified.

42966 `RTS` Should any of the multiple pending signals in the range SIGRTMIN to SIGRTMAX be selected, it  
42967 shall be the lowest numbered one. The selection order between realtime and non-realtime  
42968 signals, or between multiple pending non-realtime signals, is unspecified.

42969 **RETURN VALUE**

42970 Upon successful completion, *sigwait()* shall store the signal number of the received signal at the  
42971 location referenced by *sig* and return zero. Otherwise, an error number shall be returned to  
42972 indicate the error.

42973 **ERRORS**42974 The *sigwait()* function may fail if:42975 [EINVAL] The *set* argument contains an invalid or unsupported signal number.42976 **EXAMPLES**

42977 None.

42978 **APPLICATION USAGE**

42979 None.

42980 **RATIONALE**

42981 To provide a convenient way for a thread to wait for a signal, this volume of  
42982 IEEE Std 1003.1-200x provides the *sigwait()* function. For most cases where a thread has to wait  
42983 for a signal, the *sigwait()* function should be quite convenient, efficient, and adequate.

42984 However, requests were made for a lower-level primitive than *sigwait()* and for semaphores that  
42985 could be used by threads. After some consideration, threads were allowed to use semaphores  
42986 and *sem\_post()* was defined to be async-signal and async-cancel-safe.

42987 In summary, when it is necessary for code run in response to an asynchronous signal to notify a  
42988 thread, *sigwait()* should be used to handle the signal. Alternatively, if the implementation  
42989 provides semaphores, they also can be used, either following *sigwait()* or from within a signal  
42990 handling routine previously registered with *sigaction()*.

42991 **FUTURE DIRECTIONS**

42992 None.

42993 **SEE ALSO**

42994 Section 2.4 (on page 478), Section 2.8.1 (on page 491), *pause()*, *pthread\_sigmask()*, *sigaction()*,  
42995 *sigpending()*, *sigsuspend()*, *sigwaitinfo()*, the Base Definitions volume of IEEE Std 1003.1-200x,  
42996 <**signal.h**>, <**time.h**>

42997 **CHANGE HISTORY**

42998 First released in Issue 5. Included for alignment with the POSIX Realtime Extension and the  
42999 POSIX Threads Extension.

43000 **Issue 6**

43001 The RATIONALE section is added.

43002 The **restrict** keyword is added to the *sigwait()* prototype for alignment with the  
43003 ISO/IEC 9899:1999 standard.

43004 **NAME**

43005 sigwaitinfo — wait for queued signals (**REALTIME**)

43006 **SYNOPSIS**

43007 RTS `#include <signal.h>`

43008 `int sigwaitinfo(const sigset_t *restrict set, siginfo_t *restrict info);`

43009

43010 **DESCRIPTION**

43011 Refer to *sigtimedwait()*.

43012 **NAME**

43013 sin, sinf, sinl — sine function

43014 **SYNOPSIS**

43015 #include &lt;math.h&gt;

43016 double sin(double x);

43017 float sinf(float x);

43018 long double sinl(long double x);

43019 **DESCRIPTION**

43020 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 43021 conflict between the requirements described here and the ISO C standard is unintentional. This  
 43022 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

43023 These functions shall compute the sine of their argument *x*, measured in radians.

43024 An application wishing to check for error situations should set *errno* to zero and call  
 43025 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 43026 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 43027 zero, an error has occurred.

43028 **RETURN VALUE**43029 Upon successful completion, these functions shall return the sine of *x*.43030 **MX** If *x* is NaN, a NaN shall be returned.43031 If *x* is  $\pm 0$ , *x* shall be returned.43032 If *x* is subnormal, a range error may occur and *x* should be returned.

43033 If *x* is  $\pm\text{Inf}$ , a domain error shall occur, and either a NaN (if supported), or an implementation-  
 43034 defined value shall be returned.

43035 **ERRORS**

43036 These functions shall fail if:

43037 **MX** **Domain Error** The *x* argument is  $\pm\text{Inf}$ .

43038 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero, |  
 43039 then *errno* shall be set to [EDOM]. If the integer expression (math\_errhandling |  
 43040 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception |  
 43041 shall be raised. |

43042 These functions may fail if:

43043 **MX** **Range Error** The value of *x* is subnormal

43044 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero, |  
 43045 then *errno* shall be set to [ERANGE]. If the integer expression |  
 43046 (math\_errhandling & MATH\_ERREXCEPT) is non-zero, then the underflow |  
 43047 floating-point exception shall be raised. |

43048 **EXAMPLES**43049 **Taking the Sine of a 45-Degree Angle**

```
43050 #include <math.h>
43051 ...
43052 double radians = 45.0 * M_PI / 180;
43053 double result;
43054 ...
43055 result = sin(radians);
```

43056 **APPLICATION USAGE**

43057 These functions may lose accuracy when their argument is near a multiple of  $\pi$  or is far from 0.0.

43058 On error, the expressions (math\_errhandling & MATH\_ERRNO) and (math\_errhandling &  
43059 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

43060 **RATIONALE**

43061 None.

43062 **FUTURE DIRECTIONS**

43063 None.

43064 **SEE ALSO**

43065 *asin()*, *feclearexcept()*, *fetestexcept()*, *isnan()*, the Base Definitions volume of IEEE Std 1003.1-200x, |  
43066 Section 4.18, Treatment of Error Conditions for Mathematical Functions, <math.h> |

43067 **CHANGE HISTORY**

43068 First released in Issue 1. Derived from Issue 1 of the SVID.

43069 **Issue 5**

43070 The last two paragraphs of the DESCRIPTION were included as APPLICATION USAGE notes  
43071 in previous issues.

43072 **Issue 6**

43073 The *sinf()* and *sinl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

43074 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
43075 revised to align with the ISO/IEC 9899:1999 standard.

43076 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
43077 marked.

43078 **NAME**

43079       sinf — sine function

43080 **SYNOPSIS**

43081       #include <math.h>

43082       float sinf(float x);

43083 **DESCRIPTION**

43084       Refer to *sin()*.

43085 **NAME**

43086       sinh, sinhf, sinhl — hyperbolic sine function

43087 **SYNOPSIS**

43088       #include &lt;math.h&gt;

43089       double sinh(double x);

43090       float sinhf(float x);

43091       long double sinhl(long double x);

43092 **DESCRIPTION**

43093 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
 43094 conflict between the requirements described here and the ISO C standard is unintentional. This  
 43095 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

43096       These functions shall compute the hyperbolic sine of their argument *x*.

43097       An application wishing to check for error situations should set *errno* to zero and call  
 43098 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 43099 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 43100 zero, an error has occurred.

43101 **RETURN VALUE**43102       Upon successful completion, these functions shall return the hyperbolic sine of *x*.

43103       If the result would cause an overflow, a range error shall occur and  $\pm$ HUGE\_VAL,  
 43104  $\pm$ HUGE\_VALF, and  $\pm$ HUGE\_VALL (with the same sign as *x*) shall be returned as appropriate for  
 43105 the type of the function.

43106 **MX**       If *x* is NaN, a NaN shall be returned.43107       If *x* is  $\pm 0$ , or  $\pm$ Inf, *x* shall be returned.43108       If *x* is subnormal, a range error may occur and *x* should be returned.43109 **ERRORS**

43110       These functions shall fail if:

43111       Range Error       The result would cause an overflow.

43112               If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero, |  
 43113 then *errno* shall be set to [ERANGE]. If the integer expression |  
 43114 (math\_errhandling & MATH\_ERREXCEPT) is non-zero, then the overflow |  
 43115 floating-point exception shall be raised. |

43116       These functions may fail if:

43117 **MX**       Range Error       The value *x* is subnormal.

43118               If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero, |  
 43119 then *errno* shall be set to [ERANGE]. If the integer expression |  
 43120 (math\_errhandling & MATH\_ERREXCEPT) is non-zero, then the underflow |  
 43121 floating-point exception shall be raised. |

43122 **EXAMPLES**

43123 None.

43124 **APPLICATION USAGE**

43125 On error, the expressions (math\_errhandling & MATH\_ERRNO) and (math\_errhandling &  
43126 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

43127 **RATIONALE**

43128 None.

43129 **FUTURE DIRECTIONS**

43130 None.

43131 **SEE ALSO**

43132 *asinh()*, *cosh()*, *feclearexcept()*, *fetestexcept()*, *isnan()*, *tanh()*, the Base Definitions volume of |  
43133 IEEE Std 1003.1-200x, Section 4.18, Treatment of Error Conditions for Mathematical Functions, |  
43134 <math.h>

43135 **CHANGE HISTORY**

43136 First released in Issue 1. Derived from Issue 1 of the SVID.

43137 **Issue 5**

43138 The DESCRIPTION is updated to indicate how an application should check for an error. This  
43139 text was previously published in the APPLICATION USAGE section.

43140 **Issue 6**43141 The *sinhf()* and *sinhl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

43142 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
43143 revised to align with the ISO/IEC 9899:1999 standard.

43144 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
43145 marked.

43146 **NAME**

43147       sinl — sine function

43148 **SYNOPSIS**

43149       #include <math.h>

43150       long double sinl(long double x);

43151 **DESCRIPTION**

43152       Refer to *sin()*.

43153 **NAME**

43154 sleep — suspend execution for an interval of time

43155 **SYNOPSIS**

43156 #include &lt;unistd.h&gt;

43157 unsigned sleep(unsigned *seconds*);43158 **DESCRIPTION**

43159 The *sleep()* function shall cause the calling thread to be suspended from execution until either  
 43160 the number of realtime seconds specified by the argument *seconds* has elapsed or a signal is  
 43161 delivered to the calling thread and its action is to invoke a signal-catching function or to  
 43162 terminate the process. The suspension time may be longer than requested due to the scheduling  
 43163 of other activity by the system.

43164 If a SIGALRM signal is generated for the calling process during execution of *sleep()* and if the  
 43165 SIGALRM signal is being ignored or blocked from delivery, it is unspecified whether *sleep()*  
 43166 returns when the SIGALRM signal is scheduled. If the signal is being blocked, it is also  
 43167 unspecified whether it remains pending after *sleep()* returns or it is discarded.

43168 If a SIGALRM signal is generated for the calling process during execution of *sleep()*, except as a  
 43169 result of a prior call to *alarm()*, and if the SIGALRM signal is not being ignored or blocked from  
 43170 delivery, it is unspecified whether that signal has any effect other than causing *sleep()* to return.

43171 If a signal-catching function interrupts *sleep()* and examines or changes either the time a  
 43172 SIGALRM is scheduled to be generated, the action associated with the SIGALRM signal, or  
 43173 whether the SIGALRM signal is blocked from delivery, the results are unspecified.

43174 If a signal-catching function interrupts *sleep()* and calls *siglongjmp()* or *longjmp()* to restore an  
 43175 environment saved prior to the *sleep()* call, the action associated with the SIGALRM signal and  
 43176 the time at which a SIGALRM signal is scheduled to be generated are unspecified. It is also  
 43177 unspecified whether the SIGALRM signal is blocked, unless the process' signal mask is restored  
 43178 as part of the environment.

43179 XSI Interactions between *sleep()* and any of *setitimer()*, *ualarm()*, or *usleep()* are unspecified.

43180 **RETURN VALUE**

43181 If *sleep()* returns because the requested time has elapsed, the value returned shall be 0. If *sleep()*  
 43182 returns due to delivery of a signal, the return value shall be the “unslept” amount (the requested  
 43183 time minus the time actually slept) in seconds.

43184 **ERRORS**

43185 No errors are defined.

43186 **EXAMPLES**

43187 None.

43188 **APPLICATION USAGE**

43189 None.

43190 **RATIONALE**

43191 There are two general approaches to the implementation of the *sleep()* function. One is to use the  
 43192 *alarm()* function to schedule a SIGALRM signal and then suspend the process waiting for that  
 43193 signal. The other is to implement an independent facility. This volume of IEEE Std 1003.1-200x  
 43194 permits either approach.

43195 In order to comply with the requirement that no primitive shall change a process attribute unless  
 43196 explicitly described by this volume of IEEE Std 1003.1-200x, an implementation using SIGALRM  
 43197 must carefully take into account any SIGALRM signal scheduled by previous *alarm()* calls, the

43198 action previously established for SIGALRM, and whether SIGALRM was blocked. If a SIGALRM  
43199 has been scheduled before the *sleep()* would ordinarily complete, the *sleep()* must be shortened  
43200 to that time and a SIGALRM generated (possibly simulated by direct invocation of the signal-  
43201 catching function) before *sleep()* returns. If a SIGALRM has been scheduled after the *sleep()*  
43202 would ordinarily complete, it must be rescheduled for the same time before *sleep()* returns. The  
43203 action and blocking for SIGALRM must be saved and restored.

43204 Historical implementations often implement the SIGALRM-based version using *alarm()* and  
43205 *pause()*. One such implementation is prone to infinite hangups, as described in *pause()*. Another  
43206 such implementation uses the C-language *setjmp()* and *longjmp()* functions to avoid that  
43207 window. That implementation introduces a different problem: when the SIGALRM signal  
43208 interrupts a signal-catching function installed by the user to catch a different signal, the  
43209 *longjmp()* aborts that signal-catching function. An implementation based on *sigprocmask()*,  
43210 *alarm()*, and *sigsuspend()* can avoid these problems.

43211 Despite all reasonable care, there are several very subtle, but detectable and unavoidable,  
43212 differences between the two types of implementations. These are the cases mentioned in this  
43213 volume of IEEE Std 1003.1-200x where some other activity relating to SIGALRM takes place, and  
43214 the results are stated to be unspecified. All of these cases are sufficiently unusual as not to be of  
43215 concern to most applications.

43216 See also the discussion of the term *realtime* in *alarm()*.

43217 Since *sleep()* can be implemented using *alarm()*, the discussion about alarms occurring early  
43218 under *alarm()* applies to *sleep()* as well.

43219 Application writers should note that the type of the argument *seconds* and the return value of  
43220 *sleep()* is **unsigned**. That means that a Strictly Conforming POSIX System Interfaces Application  
43221 cannot pass a value greater than the minimum guaranteed value for {UINT\_MAX}, which the  
43222 ISO C standard sets as 65 535, and any application passing a larger value is restricting its  
43223 portability. A different type was considered, but historical implementations, including those  
43224 with a 16-bit **int** type, consistently use either **unsigned** or **int**.

43225 Scheduling delays may cause the process to return from the *sleep()* function significantly after  
43226 the requested time. In such cases, the return value should be set to zero, since the formula  
43227 (requested time minus the time actually spent) yields a negative number and *sleep()* returns an  
43228 **unsigned**.

#### 43229 FUTURE DIRECTIONS

43230 None.

#### 43231 SEE ALSO

43232 *alarm()*, *getitimer()*, *nanosleep()*, *pause()*, *sigaction()*, *sigsetjmp()*, *ualarm()*, *usleep()*, the Base  
43233 Definitions volume of IEEE Std 1003.1-200x, <**unistd.h**>

#### 43234 CHANGE HISTORY

43235 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 43236 Issue 5

43237 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

43238 **NAME**

43239            snprintf — print formatted output

43240 **SYNOPSIS**

43241            #include &lt;stdio.h&gt;

43242            int snprintf(char \*restrict *s*, size\_t *n*,43243                const char \*restrict *format*, ...);43244 **DESCRIPTION**43245            Refer to *fprintf()*.

43246 **NAME**

43247 socket — create an endpoint for communication

43248 **SYNOPSIS**

43249 #include &lt;sys/socket.h&gt;

43250 int socket(int *domain*, int *type*, int *protocol*);43251 **DESCRIPTION**43252 The *socket()* function shall create an unbound socket in a communications domain, and return a  
43253 file descriptor that can be used in later function calls that operate on sockets.43254 The *socket()* function takes the following arguments:43255 *domain* Specifies the communications domain in which a socket is to be created.43256 *type* Specifies the type of socket to be created.43257 *protocol* Specifies a particular protocol to be used with the socket. Specifying a *protocol*  
43258 of 0 causes *socket()* to use an unspecified default protocol appropriate for the  
43259 requested socket type.43260 The *domain* argument specifies the address family used in the communications domain. The  
43261 address families supported by the system are implementation-defined.43262 Symbolic constants that can be used for the domain argument are defined in the <sys/socket.h>  
43263 header.43264 The *type* argument specifies the socket type, which determines the semantics of communication  
43265 over the socket. The following socket types are defined; implementations may specify additional  
43266 socket types:43267 SOCK\_STREAM Provides sequenced, reliable, bidirectional, connection-mode byte  
43268 streams, and may provide a transmission mechanism for out-of-band  
43269 data.43270 SOCK\_DGRAM Provides datagrams, which are connectionless-mode, unreliable messages  
43271 of fixed maximum length.43272 SOCK\_SEQPACKET Provides sequenced, reliable, bidirectional, connection-mode  
43273 transmission path for records. A record can be sent using one or more  
43274 output operations and received using one or more input operations, but a  
43275 single operation never transfers part of more than one record. Record  
43276 boundaries are visible to the receiver via the MSG\_EOR flag.43277 If the *protocol* argument is non-zero, it shall specify a protocol that is supported by the address  
43278 family. If the *protocol* argument is zero, the default protocol for this address family and type shall  
43279 be used. The protocols supported by the system are implementation-defined.43280 The process may need to have appropriate privileges to use the *socket()* function or to create  
43281 some sockets.43282 **RETURN VALUE**43283 Upon successful completion, *socket()* shall return a non-negative integer, the socket file  
43284 descriptor. Otherwise, a value of -1 shall be returned and *errno* set to indicate the error.43285 **ERRORS**43286 The *socket()* function shall fail if:

43287 [EAFNOSUPPORT]

43288 The implementation does not support the specified address family.

- 43289 [EMFILE] No more file descriptors are available for this process.
- 43290 [ENFILE] No more file descriptors are available for the system.
- 43291 [EPROTONOSUPPORT]  
 43292 The protocol is not supported by the address family, or the protocol is not  
 43293 supported by the implementation.
- 43294 [EPROTOTYPE] The socket type is not supported by the protocol.
- 43295 The *socket()* function may fail if:
- 43296 [EACCES] The process does not have appropriate privileges.
- 43297 [ENOBUFS] Insufficient resources were available in the system to perform the operation.
- 43298 [ENOMEM] Insufficient memory was available to fulfill the request.
- 43299 **EXAMPLES**
- 43300 None.
- 43301 **APPLICATION USAGE**
- 43302 The documentation for specific address families specifies which protocols each address family  
 43303 supports. The documentation for specific protocols specifies which socket types each protocol  
 43304 supports.
- 43305 The application can determine whether an address family is supported by trying to create a  
 43306 socket with *domain* set to the protocol in question.
- 43307 **RATIONALE**
- 43308 None.
- 43309 **FUTURE DIRECTIONS**
- 43310 None.
- 43311 **SEE ALSO**
- 43312 *accept()*, *bind()*, *connect()*, *getsockname()*, *getsockopt()*, *listen()*, *recv()*, *recvfrom()*, *recvmsg()*,  
 43313 *send()*, *sendmsg()*, *setsockopt()*, *shutdown()*, *socketpair()*, the Base Definitions volume of  
 43314 IEEE Std 1003.1-200x, <*netinet/in.h*>, <*sys/socket.h*>
- 43315 **CHANGE HISTORY**
- 43316 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

43317 **NAME**

43318 socketpair — create a pair of connected sockets

43319 **SYNOPSIS**

43320 #include &lt;sys/socket.h&gt;

43321 int socketpair(int domain, int type, int protocol,  
43322 int socket\_vector[2]);43323 **DESCRIPTION**43324 The *socketpair()* function shall create an unbound pair of connected sockets in a specified *domain*,  
43325 of a specified *type*, under the protocol optionally specified by the *protocol* argument. The two  
43326 sockets shall be identical. The file descriptors used in referencing the created sockets shall be  
43327 returned in *socket\_vector*[0] and *socket\_vector*[1].43328 The *socketpair()* function takes the following arguments:

43329	<i>domain</i>	Specifies the communications domain in which the sockets are to be created.
43330	<i>type</i>	Specifies the type of sockets to be created.
43331	<i>protocol</i>	Specifies a particular protocol to be used with the sockets. Specifying a
43332		<i>protocol</i> of 0 causes <i>socketpair()</i> to use an unspecified default protocol
43333		appropriate for the requested socket type.
43334	<i>socket_vector</i>	Specifies a 2-integer array to hold the file descriptors of the created socket
43335		pair.

43336 The *type* argument specifies the socket type, which determines the semantics of communications  
43337 over the socket. The following socket types are defined; implementations may specify additional  
43338 socket types:

43339	SOCK_STREAM	Provides sequenced, reliable, bidirectional, connection-mode byte
43340		streams, and may provide a transmission mechanism for out-of-band
43341		data.
43342	SOCK_DGRAM	Provides datagrams, which are connectionless-mode, unreliable messages
43343		of fixed maximum length.
43344	SOCK_SEQPACKET	Provides sequenced, reliable, bidirectional, connection-mode
43345		transmission paths for records. A record can be sent using one or more
43346		output operations and received using one or more input operations, but a
43347		single operation never transfers part of more than one record. Record
43348		boundaries are visible to the receiver via the MSG_EOR flag.

43349 If the *protocol* argument is non-zero, it shall specify a protocol that is supported by the address  
43350 family. If the *protocol* argument is zero, the default protocol for this address family and type shall  
43351 be used. The protocols supported by the system are implementation-defined.43352 The process may need to have appropriate privileges to use the *socketpair()* function or to create  
43353 some sockets.43354 **RETURN VALUE**43355 Upon successful completion, this function shall return 0; otherwise,  $-1$  shall be returned and  
43356 *errno* set to indicate the error.43357 **ERRORS**43358 The *socketpair()* function shall fail if:

43359 [EAFNOSUPPORT]

43360 The implementation does not support the specified address family.

- 43361 [EMFILE] No more file descriptors are available for this process.
- 43362 [ENFILE] No more file descriptors are available for the system.
- 43363 [EOPNOTSUPP] The specified protocol does not permit creation of socket pairs.
- 43364 [EPROTONOSUPPORT]  
43365 The protocol is not supported by the address family, or the protocol is not  
43366 supported by the implementation.
- 43367 [EPROTOTYPE] The socket type is not supported by the protocol.
- 43368 The *socketpair()* function may fail if:
- 43369 [EACCES] The process does not have appropriate privileges.
- 43370 [ENOBUFS] Insufficient resources were available in the system to perform the operation.
- 43371 [ENOMEM] Insufficient memory was available to fulfill the request.

**43372 EXAMPLES**

43373 None.

**43374 APPLICATION USAGE**

43375 The documentation for specific address families specifies which protocols each address family  
43376 supports. The documentation for specific protocols specifies which socket types each protocol  
43377 supports.

43378 The *socketpair()* function is used primarily with UNIX domain sockets and need not be  
43379 supported for other domains.

**43380 RATIONALE**

43381 None.

**43382 FUTURE DIRECTIONS**

43383 None.

**43384 SEE ALSO**

43385 *socket()*, the Base Definitions volume of IEEE Std 1003.1-200x, <sys/socket.h>

**43386 CHANGE HISTORY**

43387 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

43388 **NAME**

43389       printf — print formatted output

43390 **SYNOPSIS**

43391       #include <stdio.h>

43392       int printf(char \*restrict *s*, const char \*restrict *format*, ...);

43393 **DESCRIPTION**

43394       Refer to *fprintf()*.

43395 **NAME**

43396 sqrt, sqrtf, sqrtl — square root function

43397 **SYNOPSIS**

43398 #include &lt;math.h&gt;

43399 double sqrt(double x);

43400 float sqrtf(float x);

43401 long double sqrtl(long double x);

43402 **DESCRIPTION**

43403 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 43404 conflict between the requirements described here and the ISO C standard is unintentional. This  
 43405 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

43406 These functions shall compute the square root of their argument  $x$ ,  $\sqrt{x}$ .

43407 An application wishing to check for error situations should set *errno* to zero and call  
 43408 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 43409 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 43410 zero, an error has occurred.

43411 **RETURN VALUE**43412 Upon successful completion, these functions shall return the square root of  $x$ .

43413 **MX** For finite values of  $x < -0$ , a domain error shall occur, and either a NaN (if supported), or an  
 43414 implementation-defined value shall be returned.

43415 **MX** If  $x$  is NaN, a NaN shall be returned.

43416 If  $x$  is  $\pm 0$ , or  $+\text{Inf}$ ,  $x$  shall be returned.

43417 If  $x$  is  $-\text{Inf}$ , a domain error shall occur, and either a NaN (if supported), or an implementation-  
 43418 defined value shall be returned.

43419 **ERRORS**

43420 These functions shall fail if:

43421 **MX** Domain Error The finite value of  $x$  is  $< -0$ , or  $x$  is  $-\text{Inf}$ .

43422 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero, |  
 43423 then *errno* shall be set to [EDOM]. If the integer expression (math\_errhandling |  
 43424 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception |  
 43425 shall be raised. |

43426 **EXAMPLES**43427 **Taking the Square Root of 9.0**

43428 #include &lt;math.h&gt;

43429 ...

43430 double x = 9.0;

43431 double result;

43432 ...

43433 result = sqrt(x);

43434 **APPLICATION USAGE**

43435           On error, the expressions (`math_errhandling & MATH_ERRNO`) and (`math_errhandling &`  
43436           `MATH_ERREXCEPT`) are independent of each other, but at least one of them must be non-zero.

43437 **RATIONALE**

43438           None.

43439 **FUTURE DIRECTIONS**

43440           None.

43441 **SEE ALSO**

43442           *feclearexcept()*, *fetetestexcept()*, *isnan()*, the Base Definitions volume of IEEE Std 1003.1-200x, |  
43443           Section 4.18, Treatment of Error Conditions for Mathematical Functions, `<math.h>`, `<stdio.h>` |

43444 **CHANGE HISTORY**

43445           First released in Issue 1. Derived from Issue 1 of the SVID.

43446 **Issue 5**

43447           The DESCRIPTION is updated to indicate how an application should check for an error. This  
43448           text was previously published in the APPLICATION USAGE section.

43449 **Issue 6**

43450           The *sqrtf()* and *sqrtl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

43451           The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
43452           revised to align with the ISO/IEC 9899:1999 standard.

43453           IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
43454           marked.

43455 **NAME**

43456           srand — pseudo-random number generator

43457 **SYNOPSIS**

43458           #include <stdlib.h>

43459           void srand(unsigned *seed*);

43460 **DESCRIPTION**

43461           Refer to *rand()*.

43462 **NAME**

43463           srand48 — seed uniformly distributed double-precision pseudo-random number generator

43464 **SYNOPSIS**

43465 XSI        #include &lt;stdlib.h&gt;

43466            void srand48(long *seedval*);

43467

43468 **DESCRIPTION**43469            Refer to *drand48()*.

43470 **NAME**

43471           srandom — seed pseudo-random number generator

43472 **SYNOPSIS**

43473 xSI       #include &lt;stdlib.h&gt;

43474           void srandom(unsigned *seed*);

43475

43476 **DESCRIPTION**43477           Refer to *initstate()*.

43478 **NAME**43479        `sscanf` — convert formatted input43480 **SYNOPSIS**43481        `#include <stdio.h>`43482        `int sscanf(const char *restrict s, const char *restrict format, ...);`43483 **DESCRIPTION**43484        Refer to *fscanf()*.

43485 **NAME**

43486       stat — get file status

43487 **SYNOPSIS**

43488       #include &lt;sys/stat.h&gt;

43489       int stat(const char \*restrict path, struct stat \*restrict buf);

43490 **DESCRIPTION**

43491       The *stat()* function shall obtain information about the named file and write it to the area pointed  
 43492       to by the *buf* argument. The *path* argument points to a pathname naming a file. Read, write, or  
 43493       execute permission of the named file is not required. An implementation that provides  
 43494       additional or alternate file access control mechanisms may, under implementation-defined  
 43495       conditions, cause *stat()* to fail. In particular, the system may deny the existence of the file  
 43496       specified by *path*.

43497       If the named file is a symbolic link, the *stat()* function shall continue pathname resolution using  
 43498       the contents of the symbolic link, and shall return information pertaining to the resulting file if  
 43499       the file exists.

43500       The *buf* argument is a pointer to a **stat** structure, as defined in the <sys/stat.h> header, into  
 43501       which information is placed concerning the file.

43502       The *stat()* function shall update any time-related fields (as described in the Base Definitions  
 43503       volume of IEEE Std 1003.1-200x, Section 4.7, File Times Update), before writing into the **stat**  
 43504       structure.

43505       The structure members *st\_mode*, *st\_ino*, *st\_dev*, *st\_uid*, *st\_gid*, *st\_atime*, *st\_ctime*, and *st\_mtime*  
 43506       shall have meaningful values for all file types defined in this volume of IEEE Std 1003.1-200x.  
 43507       The value of the member *st\_nlink* shall be set to the number of links to the file.

43508 **RETURN VALUE**

43509       Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to  
 43510       indicate the error.

43511 **ERRORS**43512       The *stat()* function shall fail if:

43513       [EACCES]       Search permission is denied for a component of the path prefix.

43514       [EIO]           An error occurred while reading from the file system.

43515       [ELOOP]        A loop exists in symbolic links encountered during resolution of the *path*  
 43516       argument.

43517       [ENAMETOOLONG]

43518       The length of the *path* argument exceeds {PATH\_MAX} or a pathname  
 43519       component is longer than {NAME\_MAX}.43520       [ENOENT]       A component of *path* does not name an existing file or *path* is an empty string.

43521       [ENOTDIR]      A component of the path prefix is not a directory.

43522       [EOVERFLOW]    The file size in bytes or the number of blocks allocated to the file or the file  
 43523       serial number cannot be represented correctly in the structure pointed to by  
 43524       *buf*.43525       The *stat()* function may fail if:43526       [ELOOP]        More than {SYMLOOP\_MAX} symbolic links were encountered during  
 43527       resolution of the *path* argument.

43528 [ENAMETOOLONG]  
 43529 As a result of encountering a symbolic link in resolution of the *path* argument, |  
 43530 the length of the substituted pathname string exceeded {PATH\_MAX}. |

43531 [EOVERFLOW] A value to be stored would overflow one of the members of the **stat** structure.

43532 **EXAMPLES**43533 **Obtaining File Status Information**

43534 The following example shows how to obtain file status information for a file named  
 43535 **/home/cnd/mod1**. The structure variable *buffer* is defined for the **stat** structure.

```
43536 #include <sys/types.h>
43537 #include <sys/stat.h>
43538 #include <fcntl.h>

43539 struct stat buffer;
43540 int      status;
43541 ...
43542 status = stat("/home/cnd/mod1", &buffer);
```

43543 **Getting Directory Information**

43544 The following example fragment gets status information for each entry in a directory. The call to  
 43545 the *stat()* function stores file information in the **stat** structure pointed to by *statbuf*. The lines  
 43546 that follow the *stat()* call format the fields in the **stat** structure for presentation to the user of the  
 43547 program.

```
43548 #include <sys/types.h>
43549 #include <sys/stat.h>
43550 #include <dirent.h>
43551 #include <pwd.h>
43552 #include <grp.h>
43553 #include <time.h>
43554 #include <locale.h>
43555 #include <langinfo.h>
43556 #include <stdio.h>
43557 #include <stdint.h>

43558 struct dirent *dp;
43559 struct stat   statbuf;
43560 struct passwd *pwd;
43561 struct group  *grp;
43562 struct tm     *tm;
43563 char          datestring[256];
43564 ...
43565 /* Loop through directory entries */
43566 while ((dp = readdir(dir)) != NULL) {
43567     /* Get entry's information. */
43568     if (stat(dp->d_name, &statbuf) == -1)
43569         continue;

43570     /* Print out type, permissions, and number of links. */
43571     printf("%10.10s", sperm (statbuf.st_mode));
43572     printf("%4d", statbuf.st_nlink);
```

```

43573     /* Print out owners name if it is found using getpwuid(). */
43574     if ((pwd = getpwuid(statbuf.st_uid)) != NULL)
43575         printf(" %-8.8s", pwd->pw_name);
43576     else
43577         printf(" %-8d", statbuf.st_uid);
43578     /* Print out group name if it's found using getgrgid(). */
43579     if ((grp = getgrgid(statbuf.st_gid)) != NULL)
43580         printf(" %-8.8s", grp->gr_name);
43581     else
43582         printf(" %-8d", statbuf.st_gid);
43583     /* Print size of file. */
43584     printf(" %9jd", (intmax_t)statbuf.st_size);
43585     tm = localtime(&statbuf.st_mtime);
43586     /* Get localized date string. */
43587     strftime(datestring, sizeof(datestring), nl_langinfo(D_T_FMT), tm);
43588     printf(" %s %s\n", datestring, dp->d_name);
43589 }

```

#### 43590 APPLICATION USAGE

43591 None.

#### 43592 RATIONALE

43593 The intent of the paragraph describing “additional or alternate file access control mechanisms”  
 43594 is to allow a secure implementation where a process with a label that does not dominate the  
 43595 file’s label cannot perform a *stat()* function. This is not related to read permission; a process with  
 43596 a label that dominates the file’s label does not need read permission. An implementation that  
 43597 supports write-up operations could fail *lstat()* function calls even though it has a valid file  
 43598 descriptor open for writing.

#### 43599 FUTURE DIRECTIONS

43600 None.

#### 43601 SEE ALSO

43602 *lstat()*, *lstat()*, *readlink()*, *symlink()*, the Base Definitions volume of IEEE Std 1003.1-200x,  
 43603 <sys/stat.h>, <sys/types.h>

#### 43604 CHANGE HISTORY

43605 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 43606 Issue 5

43607 Large File Summit extensions are added.

#### 43608 Issue 6

43609 In the SYNOPSIS, the optional include of the <sys/types.h> header is removed.

43610 The following new requirements on POSIX implementations derive from alignment with the  
 43611 Single UNIX Specification:

- 43612 • The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was  
 43613 required for conforming implementations of previous POSIX specifications, it was not  
 43614 required for UNIX applications.
- 43615 • The [EIO] mandatory error condition is added.

- 43616 • The [ELOOP] mandatory error condition is added.
  - 43617 • The [EOVERFLOW] mandatory error condition is added. This change is to support large
  - 43618 files.
  - 43619 • The [ENAMETOOLONG] and the second [EOVERFLOW] optional error conditions are
  - 43620 added.
- 43621 The following changes were made to align with the IEEE P1003.1a draft standard:
- 43622 • Details are added regarding the treatment of symbolic links.
  - 43623 • The [ELOOP] optional error condition is added.
- 43624 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.
- 43625 The **restrict** keyword is added to the *stat()* prototype for alignment with the ISO/IEC 9899:1999
- 43626 standard.

43627 **NAME**

43628           statvfs — get file system information

43629 **SYNOPSIS**

43630 xSI        #include &lt;sys/statvfs.h&gt;

43631            int statvfs(const char \*restrict *path*, struct statvfs \*restrict *buf*);

43632

43633 **DESCRIPTION**43634           Refer to *fstatvfs()*.

43635 **NAME**

43636 stderr, stdin, stdout — standard I/O streams

43637 **SYNOPSIS**

43638 #include <stdio.h>

43639 extern FILE \*stderr, \*stdin, \*stdout;

43640 **DESCRIPTION**

43641 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
 43642 conflict between the requirements described here and the ISO C standard is unintentional. This  
 43643 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

43644 A file with associated buffering is called a *stream* and is declared to be a pointer to a defined type  
 43645 **FILE**. The *fopen()* function shall create certain descriptive data for a stream and return a pointer  
 43646 to designate the stream in all further transactions. Normally, there are three open streams with  
 43647 constant pointers declared in the <stdio.h> header and associated with the standard open files.

43648 At program start-up, three streams shall be predefined and need not be opened explicitly:  
 43649 *standard input* (for reading conventional input), *standard output* (for writing conventional output),  
 43650 and *standard error* (for writing diagnostic output). When opened, the standard error stream is not  
 43651 fully buffered; the standard input and standard output streams are fully buffered if and only if  
 43652 the stream can be determined not to refer to an interactive device.

43653 cx The following symbolic values in <unistd.h> define the file descriptors that shall be associated  
 43654 with the C-language *stdin*, *stdout*, and *stderr* when the application is started:

43655 STDIN\_FILENO Standard input value, *stdin*. Its value is 0.

43656 STDOUT\_FILENO Standard output value, *stdout*. Its value is 1.

43657 STDERR\_FILENO Standard error value, *stderr*. Its value is 2.

43658 The *stderr* stream is expected to be open for reading and writing.

43659 **RETURN VALUE**

43660 None.

43661 **ERRORS**

43662 No errors are defined.

43663 **EXAMPLES**

43664 None.

43665 **APPLICATION USAGE**

43666 None.

43667 **RATIONALE**

43668 None.

43669 **FUTURE DIRECTIONS**

43670 None.

43671 **SEE ALSO**

43672 *fclose()*, *feof()*, *ferror()*, *fileno()*, *fopen()*, *fread()*, *fseek()*, *getc()*, *gets()*, *popen()*, *printf()*, *putc()*,  
 43673 *puts()*, *read()*, *scanf()*, *setbuf()*, *setvbuf()*, *tmpfile()*, *ungetc()*, *vprintf()*, the Base Definitions  
 43674 volume of IEEE Std 1003.1-200x, <stdio.h>, <unistd.h>

43675 **CHANGE HISTORY**

43676 First released in Issue 1.

43677 **Issue 6**

43678 Extensions beyond the ISO C standard are now marked.

43679 A note that *stderr* is expected to be open for reading and writing is added to the DESCRIPTION.

43680 **NAME**

43681 strcasecmp, strncasecmp — case-insensitive string comparisons

43682 **SYNOPSIS**43683 XSI `#include <strings.h>`43684 `int strcasecmp(const char *s1, const char *s2);`43685 `int strncasecmp(const char *s1, const char *s2, size_t n);`

43686

43687 **DESCRIPTION**

43688 The *strcasecmp()* function shall compare, while ignoring differences in case, the string pointed to  
43689 by *s1* to the string pointed to by *s2*. The *strncasecmp()* function shall compare, while ignoring  
43690 differences in case, not more than *n* bytes from the string pointed to by *s1* to the string pointed to  
43691 by *s2*.

43692 In the POSIX locale, *strcasecmp()* and *strncasecmp()* shall behave as if the strings had been |  
43693 converted to lowercase and then a byte comparison performed. The results are unspecified in |  
43694 other locales. |

43695 **RETURN VALUE**

43696 Upon completion, *strcasecmp()* shall return an integer greater than, equal to, or less than 0, if the  
43697 string pointed to by *s1* is, ignoring case, greater than, equal to, or less than the string pointed to  
43698 by *s2*, respectively.

43699 Upon successful completion, *strncasecmp()* shall return an integer greater than, equal to, or less  
43700 than 0, if the possibly null-terminated array pointed to by *s1* is, ignoring case, greater than, equal  
43701 to, or less than the possibly null-terminated array pointed to by *s2*, respectively.

43702 **ERRORS**

43703 No errors are defined.

43704 **EXAMPLES**

43705 None.

43706 **APPLICATION USAGE**

43707 None.

43708 **RATIONALE**

43709 None.

43710 **FUTURE DIRECTIONS**

43711 None.

43712 **SEE ALSO**

43713 The Base Definitions volume of IEEE Std 1003.1-200x, &lt;strings.h&gt;

43714 **CHANGE HISTORY**

43715 First released in Issue 4, Version 2.

43716 **Issue 5**

43717 Moved from X/OPEN UNIX extension to BASE.

43718 **NAME**

43719           strcat — concatenate two strings

43720 **SYNOPSIS**

43721           #include &lt;string.h&gt;

43722           char \*strcat(char \*restrict *s1*, const char \*restrict *s2*);43723 **DESCRIPTION**

43724 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
43725 conflict between the requirements described here and the ISO C standard is unintentional. This  
43726 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

43727       The *strcat()* function shall append a copy of the string pointed to by *s2* (including the  
43728 terminating null byte) to the end of the string pointed to by *s1*. The initial byte of *s2* overwrites  
43729 the null byte at the end of *s1*. If copying takes place between objects that overlap, the behavior is  
43730 undefined.

43731 **RETURN VALUE**43732       The *strcat()* function shall return *s1*; no return value is reserved to indicate an error.43733 **ERRORS**

43734       No errors are defined.

43735 **EXAMPLES**

43736       None.

43737 **APPLICATION USAGE**

43738       This issue is aligned with the ISO C standard; this does not affect compatibility with XPG3  
43739 applications. Reliable error detection by this function was never guaranteed.

43740 **RATIONALE**

43741       None.

43742 **FUTURE DIRECTIONS**

43743       None.

43744 **SEE ALSO**43745       *strncat()*, the Base Definitions volume of IEEE Std 1003.1-200x, <string.h>43746 **CHANGE HISTORY**

43747       First released in Issue 1. Derived from Issue 1 of the SVID.

43748 **Issue 6**43749       The *strcat()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

43750 **NAME**

43751           strchr — string scanning operation

43752 **SYNOPSIS**

43753           #include &lt;string.h&gt;

43754           char \*strchr(const char \*s, int c);

43755 **DESCRIPTION**43756 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
43757       conflict between the requirements described here and the ISO C standard is unintentional. This  
43758       volume of IEEE Std 1003.1-200x defers to the ISO C standard.43759 **CX**       The *strchr()* function shall locate the first occurrence of *c* (converted to an **unsigned char**) in the  
43760       string pointed to by *s*. The terminating null byte is considered to be part of the string.43761 **RETURN VALUE**43762           Upon completion, *strchr()* shall return a pointer to the byte, or a null pointer if the byte was not  
43763           found.43764 **ERRORS**

43765           No errors are defined.

43766 **EXAMPLES**

43767           None.

43768 **APPLICATION USAGE**

43769           None.

43770 **RATIONALE**

43771           None.

43772 **FUTURE DIRECTIONS**

43773           None.

43774 **SEE ALSO**43775           *strchr()*, the Base Definitions volume of IEEE Std 1003.1-200x, <string.h>43776 **CHANGE HISTORY**

43777           First released in Issue 1. Derived from Issue 1 of the SVID.

43778 **Issue 6**

43779           Extensions beyond the ISO C standard are now marked.

43780 **NAME**

43781 strcmp — compare two strings

43782 **SYNOPSIS**

43783 #include &lt;string.h&gt;

43784 int strcmp(const char \*s1, const char \*s2);

43785 **DESCRIPTION**

43786 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
 43787 conflict between the requirements described here and the ISO C standard is unintentional. This  
 43788 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

43789 The *strcmp()* function shall compare the string pointed to by *s1* to the string pointed to by *s2*.

43790 The sign of a non-zero return value shall be determined by the sign of the difference between the  
 43791 values of the first pair of bytes (both interpreted as type **unsigned char**) that differ in the strings  
 43792 being compared.

43793 **RETURN VALUE**

43794 Upon completion, *strcmp()* shall return an integer greater than, equal to, or less than 0, if the  
 43795 string pointed to by *s1* is greater than, equal to, or less than the string pointed to by *s2*,  
 43796 respectively.

43797 **ERRORS**

43798 No errors are defined.

43799 **EXAMPLES**43800 **Checking a Password Entry**

43801 The following example compares the information read from standard input to the value of the  
 43802 name of the user entry. If the *strcmp()* function returns 0 (indicating a match), a further check  
 43803 will be made to see if the user entered the proper old password. The *crypt()* function shall  
 43804 encrypt the old password entered by the user, using the value of the encrypted password in the  
 43805 **passwd** structure as the salt. If this value matches the value of the encrypted **passwd** in the  
 43806 structure, the entered password *oldpasswd* is the correct user's password. Finally, the program  
 43807 encrypts the new password so that it can store the information in the **passwd** structure.

```

43808 #include <string.h>
43809 #include <unistd.h>
43810 #include <stdio.h>
43811 ...
43812 int valid_change;
43813 struct passwd *p;
43814 char user[100];
43815 char oldpasswd[100];
43816 char newpasswd[100];
43817 char savepasswd[100];
43818 ...
43819 if (strcmp(p->pw_name, user) == 0) {
43820     if (strcmp(p->pw_passwd, crypt(oldpasswd, p->pw_passwd)) == 0) {
43821         strcpy(savepasswd, crypt(newpasswd, user));
43822         p->pw_passwd = savepasswd;
43823         valid_change = 1;
43824     }
43825     else {
```

```
43826             fprintf(stderr, "Old password is not valid\n");
43827         }
43828     }
43829     ...
```

**43830 APPLICATION USAGE**

43831 None.

**43832 RATIONALE**

43833 None.

**43834 FUTURE DIRECTIONS**

43835 None.

**43836 SEE ALSO**

43837 *strncmp()*, the Base Definitions volume of IEEE Std 1003.1-200x, <**string.h**>

**43838 CHANGE HISTORY**

43839 First released in Issue 1. Derived from Issue 1 of the SVID.

**43840 Issue 6**

43841 Extensions beyond the ISO C standard are now marked.

43842 **NAME**

43843 strcoll — string comparison using collating information

43844 **SYNOPSIS**

43845 #include &lt;string.h&gt;

43846 int strcoll(const char \*s1, const char \*s2);

43847 **DESCRIPTION**

43848 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 43849 conflict between the requirements described here and the ISO C standard is unintentional. This  
 43850 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

43851 The *strcoll()* function shall compare the string pointed to by *s1* to the string pointed to by *s2*,  
 43852 both interpreted as appropriate to the *LC\_COLLATE* category of the current locale.

43853 CX The *strcoll()* function shall not change the setting of *errno* if successful.

43854 Since no return value is reserved to indicate an error, an application wishing to check for error  
 43855 situations should set *errno* to 0, then call *strcoll()*, then check *errno*.

43856 **RETURN VALUE**

43857 Upon successful completion, *strcoll()* shall return an integer greater than, equal to, or less than 0,  
 43858 according to whether the string pointed to by *s1* is greater than, equal to, or less than the string  
 43859 CX pointed to by *s2* when both are interpreted as appropriate to the current locale. On error,  
 43860 *strcoll()* may set *errno*, but no return value is reserved to indicate an error.

43861 **ERRORS**43862 The *strcoll()* function may fail if:

43863 CX [EINVAL] The *s1* or *s2* arguments contain characters outside the domain of the collating  
 43864 sequence.

43865 **EXAMPLES**43866 **Comparing Nodes**

43867 The following example uses an application-defined function, *node\_compare()*, to compare two  
 43868 nodes based on an alphabetical ordering of the *string* field.

```
43869 #include <string.h>
43870 ...
43871 struct node { /* These are stored in the table. */
43872     char *string;
43873     int length;
43874 };
43875 ...
43876 int node_compare(const void *node1, const void *node2)
43877 {
43878     return strcoll(((const struct node *)node1)->string,
43879                 ((const struct node *)node2)->string);
43880 }
43881 ...
```

43882 **APPLICATION USAGE**43883 The *strxfrm()* and *strcmp()* functions should be used for sorting large lists.

43884 **RATIONALE**

43885           None.

43886 **FUTURE DIRECTIONS**

43887           None.

43888 **SEE ALSO**43889           *strcmp()*, *strxfrm()*, the Base Definitions volume of IEEE Std 1003.1-200x, <**string.h**>43890 **CHANGE HISTORY**

43891           First released in Issue 3.

43892 **Issue 5**43893           The DESCRIPTION is updated to indicate that *errno* does not be changed if the function is  
43894           successful.43895 **Issue 6**

43896           Extensions beyond the ISO C standard are now marked.

43897           The following new requirements on POSIX implementations derive from alignment with the  
43898           Single UNIX Specification:43899           

- The [EINVAL] optional error condition is added.

43900           An example is added. |

43901 **NAME**

43902           strcpy — copy a string

43903 **SYNOPSIS**

43904           #include &lt;string.h&gt;

43905           char \*strcpy(char \*restrict *s1*, const char \*restrict *s2*);43906 **DESCRIPTION**

43907 *cx*       The functionality described on this reference page is aligned with the ISO C standard. Any  
 43908 conflict between the requirements described here and the ISO C standard is unintentional. This  
 43909 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

43910       The *strcpy()* function shall copy the string pointed to by *s2* (including the terminating null byte)  
 43911 into the array pointed to by *s1*. If copying takes place between objects that overlap, the behavior  
 43912 is undefined.

43913 **RETURN VALUE**43914       The *strcpy()* function shall return *s1*; no return value is reserved to indicate an error.43915 **ERRORS**

43916       No errors are defined.

43917 **EXAMPLES**43918           **Initializing a String**43919       The following example copies the string "-----" into the *permstring* variable.

```
43920       #include <string.h>
43921       ...
43922       static char permstring[11];
43923       ...
43924       strcpy(permstring, "-----");
43925       ...
```

43926           **Storing a Key and Data**

43927       The following example allocates space for a key using *malloc()* then uses *strcpy()* to place the  
 43928 key there. Then it allocates space for data using *malloc()*, and uses *strcpy()* to place data there.  
 43929 (The user-defined function *dbfree()* frees memory previously allocated to an array of type **struct**  
 43930 **element** \*.)

```
43931       #include <string.h>
43932       #include <stdlib.h>
43933       #include <stdio.h>
43934       ...
43935       /* Structure used to read data and store it. */
43936       struct element {
43937           char *key;
43938           char *data;
43939       };
43940       struct element *tbl, *curtbl;
43941       char *key, *data;
43942       int count;
43943       ...
43944       void dbfree(struct element *, int);
```

```
43945     ...
43946     if ((curtbl->key = malloc(strlen(key) + 1)) == NULL) {
43947         perror("malloc"); dbfree(tbl, count); return NULL;
43948     }
43949     strcpy(curtbl->key, key);
43950
43951     if ((curtbl->data = malloc(strlen(data) + 1)) == NULL) {
43952         perror("malloc"); free(curtbl->key); dbfree(tbl, count); return NULL;
43953     }
43954     strcpy(curtbl->data, data);
43955     ...
```

**43955 APPLICATION USAGE**

43956 Character movement is performed differently in different implementations. Thus, overlapping  
43957 moves may yield surprises.

43958 This issue is aligned with the ISO C standard; this does not affect compatibility with XPG3  
43959 applications. Reliable error detection by this function was never guaranteed.

**43960 RATIONALE**

43961 None.

**43962 FUTURE DIRECTIONS**

43963 None.

**43964 SEE ALSO**

43965 *strncpy()*, the Base Definitions volume of IEEE Std 1003.1-200x, <**string.h**>

**43966 CHANGE HISTORY**

43967 First released in Issue 1. Derived from Issue 1 of the SVID.

**43968 Issue 6**

43969 The *strcpy()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

43970 **NAME**

43971 strcspn — get length of a complementary substring

43972 **SYNOPSIS**

43973 #include &lt;string.h&gt;

43974 size\_t strcspn(const char \*s1, const char \*s2);

43975 **DESCRIPTION**43976 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
43977 conflict between the requirements described here and the ISO C standard is unintentional. This  
43978 volume of IEEE Std 1003.1-200x defers to the ISO C standard.43979 The *strcspn()* function shall compute the length (in bytes) of the maximum initial segment of the  
43980 string pointed to by *s1* which consists entirely of bytes *not* from the string pointed to by *s2*.43981 **RETURN VALUE**43982 The *strcspn()* function shall return the length of the computed segment of the string pointed to  
43983 by *s1*; no return value is reserved to indicate an error.43984 **ERRORS**

43985 No errors are defined.

43986 **EXAMPLES**

43987 None.

43988 **APPLICATION USAGE**

43989 None.

43990 **RATIONALE**

43991 None.

43992 **FUTURE DIRECTIONS**

43993 None.

43994 **SEE ALSO**43995 *strspn()*, the Base Definitions volume of IEEE Std 1003.1-200x, <**string.h**>43996 **CHANGE HISTORY**

43997 First released in Issue 1. Derived from Issue 1 of the SVID.

43998 **Issue 5**43999 The RETURN VALUE section is updated to indicated that *strcspn()* returns the length of *s1*, and  
44000 not *s1* itself as was previously stated.44001 **Issue 6**44002 The Open Group Corrigendum U030/1 is applied. The text of the RETURN VALUE section is  
44003 updated to indicate that the computed segment length is returned, not the *s1* length.

44004 **NAME**

44005        **strdup** — duplicate a string

44006 **SYNOPSIS**

44007 XSI        #include <string.h>

44008        char \*strdup(const char \*s1);

44009

44010 **DESCRIPTION**

44011        The *strdup()* function shall return a pointer to a new string, which is a duplicate of the string pointed to by *s1*. The returned pointer can be passed to *free()*. A null pointer is returned if the new string cannot be created.

44014 **RETURN VALUE**

44015        The *strdup()* function shall return a pointer to a new string on success. Otherwise, it shall return a null pointer and set *errno* to indicate the error.

44017 **ERRORS**

44018        The *strdup()* function may fail if:

44019        [ENOMEM]       Storage space available is insufficient.

44020 **EXAMPLES**

44021        None.

44022 **APPLICATION USAGE**

44023        None.

44024 **RATIONALE**

44025        None.

44026 **FUTURE DIRECTIONS**

44027        None.

44028 **SEE ALSO**

44029        *free()*, *malloc()*, the Base Definitions volume of IEEE Std 1003.1-200x, <string.h>

44030 **CHANGE HISTORY**

44031        First released in Issue 4, Version 2.

44032 **Issue 5**

44033        Moved from X/OPEN UNIX extension to BASE.

44034 **NAME**

44035            strerror, strerror\_r — get error message string

44036 **SYNOPSIS**

44037            #include &lt;string.h&gt;

44038            char \*strerror(int *errnum*);44039 TSF        int strerror\_r(int *errnum*, char \**strerrbuf*, size\_t *buflen*);

44040

44041 **DESCRIPTION**44042 CX        For *strerror()*: The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-200x defers to the ISO C standard.44045            The *strerror()* function shall map the error number in *errnum* to a locale-dependent error message string and shall return a pointer to it. Typically, the values for *errnum* come from *errno*, but *strerror()* shall map any value of type **int** to a message.44048            The string pointed to shall not be modified by the application, but may be overwritten by a subsequent call to *strerror()* or *perror()*.44049 CX        The contents of the error message strings returned by *strerror()* should be determined by the setting of the *LC\_MESSAGES* category in the current locale.44050 CX        The implementation shall behave as if no function defined in this volume of IEEE Std 1003.1-200x calls *strerror()*.44051            The *strerror()* function shall not change the setting of *errno* if successful.44052            Since no return value is reserved to indicate an error, an application wishing to check for error situations should set *errno* to 0, then call *strerror()*, then check *errno*.44053            The *strerror()* function need not be reentrant. A function that is not required to be reentrant is not required to be thread-safe.44054 CX        The *strerror\_r()* function shall map the error number in *errnum* to a locale-dependent error message string and shall return the string in the buffer pointed to by *strerrbuf*, with length *buflen*.

44055

44056 TSF        The *strerror\_r()* function shall map the error number in *errnum* to a locale-dependent error message string and shall return the string in the buffer pointed to by *strerrbuf*, with length *buflen*.44057            Upon successful completion, *strerror\_r()* shall return 0. Otherwise, an error number shall be returned to indicate the error.44058            Upon successful completion, *strerror\_r()* shall return 0. Otherwise, an error number shall be returned to indicate the error.44059 TSF        The *strerror\_r()* function shall map the error number in *errnum* to a locale-dependent error message string and shall return the string in the buffer pointed to by *strerrbuf*, with length *buflen*.44060            The *strerror\_r()* function shall map the error number in *errnum* to a locale-dependent error message string and shall return the string in the buffer pointed to by *strerrbuf*, with length *buflen*.44061            The *strerror\_r()* function shall map the error number in *errnum* to a locale-dependent error message string and shall return the string in the buffer pointed to by *strerrbuf*, with length *buflen*.44062            The *strerror\_r()* function shall map the error number in *errnum* to a locale-dependent error message string and shall return the string in the buffer pointed to by *strerrbuf*, with length *buflen*.44063            The *strerror\_r()* function shall map the error number in *errnum* to a locale-dependent error message string and shall return the string in the buffer pointed to by *strerrbuf*, with length *buflen*.44064            The *strerror\_r()* function shall map the error number in *errnum* to a locale-dependent error message string and shall return the string in the buffer pointed to by *strerrbuf*, with length *buflen*.

44065

44073 **EXAMPLES**

44074 None.

44075 **APPLICATION USAGE**

44076 None.

44077 **RATIONALE**

44078 None.

44079 **FUTURE DIRECTIONS**

44080 None.

44081 **SEE ALSO**44082 *perror()*, the Base Definitions volume of IEEE Std 1003.1-200x, <**string.h**>44083 **CHANGE HISTORY**

44084 First released in Issue 3.

44085 **Issue 5**44086 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.

44087 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

44088 **Issue 6**

44089 Extensions beyond the ISO C standard are now marked.

44090 The following new requirements on POSIX implementations derive from alignment with the  
44091 Single UNIX Specification:

- 44092
- In the RETURN VALUE section, the fact that *errno* may be set is added.
  - The [EINVAL] optional error condition is added.

44094 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

44095 The *strerror\_r()* function is added in response to IEEE PASC Interpretation 1003.1c #39.44096 The *strerror\_r()* function is marked as part of the Thread-Safe Functions option.

44097 **NAME**

44098 strfmon — convert monetary value to a string

44099 **SYNOPSIS**

```
44100 xsi #include <monetary.h>
44101
44101 ssize_t strfmon(char *restrict s, size_t maxsize,
44102               const char *restrict format, ...);
44103
```

44104 **DESCRIPTION**

44105 The *strfmon()* function shall place characters into the array pointed to by *s* as controlled by the  
 44106 string pointed to by *format*. No more than *maxsize* bytes are placed into the array.

44107 The format is a character string, beginning and ending in its initial state, if any, that contains two  
 44108 types of objects: *plain characters*, which are simply copied to the output stream, and *conversion*  
 44109 *specifications*, each of which shall result in the fetching of zero or more arguments which are  
 44110 converted and formatted. The results are undefined if there are insufficient arguments for the  
 44111 format. If the format is exhausted while arguments remain, the excess arguments are simply  
 44112 ignored.

44113 The application shall ensure that a conversion specification consists of the following sequence:

- 44114 • A '%' character
- 44115 • Optional flags
- 44116 • Optional field width
- 44117 • Optional left precision
- 44118 • Optional right precision
- 44119 • A required conversion specifier character that determines the conversion to be performed

44120 **Flags**

44121 One or more of the following optional flags can be specified to control the conversion:

- 44122 =*f* An '=' followed by a single character *f* which is used as the numeric fill character. In  
 44123 order to work with precision or width counts, the fill character shall be a single byte  
 44124 character; if not, the behavior is undefined. The default numeric fill character is the  
 44125 <space>. This flag does not affect field width filling which always uses the <space>.  
 44126 This flag is ignored unless a left precision (see below) is specified.
- 44127 ^ Do not format the currency amount with grouping characters. The default is to insert  
 44128 the grouping characters if defined for the current locale.
- 44129 + or ( Specify the style of representing positive and negative currency amounts. Only one of  
 44130 '+' or '(' may be specified. If '+' is specified, the locale's equivalent of '+' and '-'  
 44131 are used (for example, in the U.S., the empty string if positive and '-' if negative). If  
 44132 '(' is specified, negative amounts are enclosed within parentheses. If neither flag is  
 44133 specified, the '+' style is used.
- 44134 ! Suppress the currency symbol from the output conversion.
- 44135 - Specify the alignment. If this flag is present the result of the conversion is left-justified  
 44136 (padded to the right) rather than right-justified. This flag shall be ignored unless a field  
 44137 width (see below) is specified.

44138 **Field Width**

44139 *w* A decimal digit string *w* specifying a minimum field width in bytes in which the result  
 44140 of the conversion is right-justified (or left-justified if the flag '-' is specified). The  
 44141 default is 0.

44142 **Left Precision**

44143 *#n* A '#' followed by a decimal digit string *n* specifying a maximum number of digits  
 44144 expected to be formatted to the left of the radix character. This option can be used to  
 44145 keep the formatted output from multiple calls to the *strfmon()* function aligned in the  
 44146 same columns. It can also be used to fill unused positions with a special character as in  
 44147 "\$\*\*\*123.45". This option causes an amount to be formatted as if it has the number  
 44148 of digits specified by *n*. If more than *n* digit positions are required, this conversion  
 44149 specification is ignored. Digit positions in excess of those actually required are filled  
 44150 with the numeric fill character (see the *=f* flag above).

44151 If grouping has not been suppressed with the '^' flag, and it is defined for the current  
 44152 locale, grouping separators are inserted before the fill characters (if any) are added.  
 44153 Grouping separators are not applied to fill characters even if the fill character is a digit.

44154 To ensure alignment, any characters appearing before or after the number in the  
 44155 formatted output such as currency or sign symbols are padded as necessary with  
 44156 <space>s to make their positive and negative formats an equal length.

44157 **Right Precision**

44158 *.p* A period followed by a decimal digit string *p* specifying the number of digits after the  
 44159 radix character. If the value of the right precision *p* is 0, no radix character appears. If a  
 44160 right precision is not included, a default specified by the current locale is used. The  
 44161 amount being formatted is rounded to the specified number of digits prior to  
 44162 formatting.

44163 **Conversion Specifier Characters**

44164 The conversion specifier characters and their meanings are:

44165 *i* The **double** argument is formatted according to the locale's international currency |  
 44166 format (for example, in the U.S.: USD 1,234.56). If the argument is ±Inf or NaN, the |  
 44167 result of the conversion is unspecified. |

44168 *n* The **double** argument is formatted according to the locale's national currency format |  
 44169 (for example, in the U.S.: \$1,234.56). If the argument is ±Inf or NaN, the result of the |  
 44170 conversion is unspecified. |

44171 *%* Convert to a '%'; no argument is converted. The entire conversion specification shall  
 44172 be %%.

44173 **Locale Information**

44174 The *LC\_MONETARY* category of the program's locale affects the behavior of this function  
 44175 including the monetary radix character (which may be different from the numeric radix  
 44176 character affected by the *LC\_NUMERIC* category), the grouping separator, the currency  
 44177 symbols, and formats. The international currency symbol should be conformant with the  
 44178 ISO 4217:1995 standard.

44179 If the value of *maxsize* is greater than {SSIZE\_MAX}, the result is implementation-defined.

44180 **RETURN VALUE**

44181 If the total number of resulting bytes including the terminating null byte is not more than  
44182 *maxsize*, *strfmon()* shall return the number of bytes placed into the array pointed to by *s*, not  
44183 including the terminating null byte. Otherwise, -1 shall be returned, the contents of the array are |  
44184 unspecified, and *errno* shall be set to indicate the error. |

44185 **ERRORS**

44186 The *strfmon()* function shall fail if:

44187 [E2BIG] Conversion stopped due to lack of space in the buffer.

44188 **EXAMPLES**

44189 Given a locale for the U.S. and the values 123.45, -123.45, and 3456.781:

44190

44191

44192

44193

44194

44195

44196

44197

44198

44199

44200

44201

44202

44203

44204

44205

44206

44207

44208

44209

44210

44211

44212

44213

44214

44215

44216

44217

44218

44219

44220

44221

44222

44223

44224

44225

44226

44227

Conversion Specification	Output	Comments
%n	\$123.45 -\$123.45 \$3,456.78	Default formatting.
%11n	\$123.45 -\$123.45 \$3,456.78	Right align within an 11 character field.
%#5n	\$ 123.45 -\$ 123.45 \$ 3,456.78	Aligned columns for values up to 99,999.
%=*#5n	\$***123.45 -\$***123.45 \$*3,456.78	Specify a fill character.
%=0#5n	\$000123.45 -\$000123.45 \$03,456.78	Fill characters do not use grouping even if the fill character is a digit.
%^#5n	\$ 123.45 -\$ 123.45 \$ 3456.78	Disable the grouping separator.
%^#5.0n	\$ 123 -\$ 123 \$ 3457	Round off to whole units.
%^#5.4n	\$ 123.4500 -\$ 123.4500 \$ 3456.7810	Increase the precision.
%(#5n	\$ 123.45 (\$ 123.45) \$ 3,456.78	Use an alternative pos/neg style.
%(!#5n	123.45 ( 123.45) 3,456.78	Disable the currency symbol.
%-14#5.4n	\$ 123.4500 -\$ 123.4500 \$3,456.7810	Left-justify the output.
%14#5.4n	\$ 123.4500 -\$ 123.4500 \$3,456.7810	Corresponding right-justified output.

44228 **APPLICATION USAGE**

44229 None.

44230 **RATIONALE**

44231 None.

44232 **FUTURE DIRECTIONS**

44233           Lowercase conversion characters are reserved for future standards use and uppercase for  
44234           implementation-defined use.

44235 **SEE ALSO**

44236           *localeconv()*, the Base Definitions volume of IEEE Std 1003.1-200x, <**monetary.h**>

44237 **CHANGE HISTORY**

44238           First released in Issue 4.

44239 **Issue 5**

44240           Moved from ENHANCED I18N to BASE.

44241           The [ENOSYS] error is removed.

44242           A sentence is added to the DESCRIPTION warning about values of *maxsize* that are greater than

44243           {SSIZE\_MAX}.

44244 **Issue 6**

44245           The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

44246           The **restrict** keyword is added to the *strfmon()* prototype for alignment with the

44247           ISO/IEC 9899:1999 standard.

## 44248 NAME

44249 strftime — convert date and time to a string

## 44250 SYNOPSIS

44251 #include &lt;time.h&gt;

```
44252     size_t strftime(char *restrict s, size_t maxsize,
44253                   const char *restrict format, const struct tm *restrict timeptr);
```

## 44254 DESCRIPTION

44255 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
 44256 conflict between the requirements described here and the ISO C standard is unintentional. This  
 44257 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

44258 The *strftime()* function shall place bytes into the array pointed to by *s* as controlled by the string  
 44259 pointed to by *format*. The format is a character string, beginning and ending in its initial shift  
 44260 state, if any. The *format* string consists of zero or more conversion specifications and ordinary  
 44261 characters. A conversion specification consists of a '%' character, possibly followed by an E or O  
 44262 modifier, and a terminating conversion specifier character that determines the conversion  
 44263 specification's behavior. All ordinary characters (including the terminating null byte) are copied  
 44264 unchanged into the array. If copying takes place between objects that overlap, the behavior is  
 44265 undefined. No more than *maxsize* bytes are placed into the array. Each conversion specifier is  
 44266 replaced by appropriate characters as described in the following list. The appropriate characters  
 44267 are determined using the *LC\_TIME* category of the current locale and by the values of zero or  
 44268 more members of the broken-down time structure pointed to by *timeptr*, as specified in brackets  
 44269 in the description. If any of the specified values are outside the normal range, the characters  
 44270 stored are unspecified.

44271 cx Local timezone information is used as though *strftime()* called *tzset()*.

44272 The following conversion specifications are supported:

44273	%a	Replaced by the locale's abbreviated weekday name. [ <i>tm_wday</i> ]
44274	%A	Replaced by the locale's full weekday name. [ <i>tm_wday</i> ]
44275	%b	Replaced by the locale's abbreviated month name. [ <i>tm_mon</i> ]
44276	%B	Replaced by the locale's full month name. [ <i>tm_mon</i> ]
44277	%c	Replaced by the locale's appropriate date and time representation. (See the Base 44278 Definitions volume of IEEE Std 1003.1-200x, < <b>time.h</b> >.)
44279	%C	Replaced by the year divided by 100 and truncated to an integer, as a decimal number 44280 [00,99]. [ <i>tm_year</i> ]
44281	%d	Replaced by the day of the month as a decimal number [01,31]. [ <i>tm_mday</i> ]
44282	%D	Equivalent to %m/%d/%y. [ <i>tm_mon</i> , <i>tm_mday</i> , <i>tm_year</i> ]
44283	%e	Replaced by the day of the month as a decimal number [1,31]; a single digit is preceded 44284 by a space. [ <i>tm_mday</i> ]
44285	%F	Equivalent to %Y-%m-%d (the ISO 8601:2000 standard date format). [ <i>tm_year</i> , <i>tm_mon</i> , 44286 <i>tm_mday</i> ]
44287	%g	Replaced by the last 2 digits of the week-based year (see below) as a decimal number 44288 [00,99]. [ <i>tm_year</i> , <i>tm_wday</i> , <i>tm_yday</i> ]
44289	%G	Replaced by the week-based year (see below) as a decimal number (for example, 1977). 44290 [ <i>tm_year</i> , <i>tm_wday</i> , <i>tm_yday</i> ]

44291	%h	Equivalent to %b. [ <i>tm_mon</i> ]	
44292	%H	Replaced by the hour (24-hour clock) as a decimal number [00,23]. [ <i>tm_hour</i> ]	
44293	%I	Replaced by the hour (12-hour clock) as a decimal number [01,12]. [ <i>tm_hour</i> ]	
44294	%j	Replaced by the day of the year as a decimal number [001,366]. [ <i>tm_yday</i> ]	
44295	%m	Replaced by the month as a decimal number [01,12]. [ <i>tm_mon</i> ]	
44296	%M	Replaced by the minute as a decimal number [00,59]. [ <i>tm_min</i> ]	
44297	%n	Replaced by a <newline>.	
44298	%p	Replaced by the locale's equivalent of either a.m. or p.m. [ <i>tm_hour</i> ]	
44299 CX	%r	Replaced by the time in a.m. and p.m. notation; in the POSIX locale this shall be	
44300		equivalent to %I:%M:%S %p. [ <i>tm_hour</i> , <i>tm_min</i> , <i>tm_sec</i> ]	
44301	%R	Replaced by the time in 24 hour notation (%H:%M). [ <i>tm_hour</i> , <i>tm_min</i> ]	
44302	%S	Replaced by the second as a decimal number [00,60]. [ <i>tm_sec</i> ]	
44303	%t	Replaced by a <tab>.	
44304	%T	Replaced by the time (%H:%M:%S). [ <i>tm_hour</i> , <i>tm_min</i> , <i>tm_sec</i> ]	
44305	%u	Replaced by the weekday as a decimal number [1,7], with 1 representing Monday.	
44306		[ <i>tm_wday</i> ]	
44307	%U	Replaced by the week number of the year as a decimal number [00,53]. The first	
44308		Sunday of January is the first day of week 1; days in the new year before this are in	
44309		week 0. [ <i>tm_year</i> , <i>tm_wday</i> , <i>tm_yday</i> ]	
44310	%V	Replaced by the week number of the year (Monday as the first day of the week) as a	
44311		decimal number [01,53]. If the week containing 1 January has four or more days in the	
44312		new year, then it is considered week 1. Otherwise, it is the last week of the previous	
44313		year, and the next week is week 1. Both January 4th and the first Thursday of January	
44314		are always in week 1. [ <i>tm_year</i> , <i>tm_wday</i> , <i>tm_yday</i> ]	
44315	%w	Replaced by the weekday as a decimal number [0,6], with 0 representing Sunday.	
44316		[ <i>tm_wday</i> ]	
44317	%W	Replaced by the week number of the year as a decimal number [00,53]. The first	
44318		Monday of January is the first day of week 1; days in the new year before this are in	
44319		week 0. [ <i>tm_year</i> , <i>tm_wday</i> , <i>tm_yday</i> ]	
44320	%x	Replaced by the locale's appropriate date representation. (See the Base Definitions	
44321		volume of IEEE Std 1003.1-200x, <time.h>.)	
44322	%X	Replaced by the locale's appropriate time representation. (See the Base Definitions	
44323		volume of IEEE Std 1003.1-200x, <time.h>.)	
44324	%y	Replaced by the last two digits of the year as a decimal number [00,99]. [ <i>tm_year</i> ]	
44325	%Y	Replaced by the year as a decimal number (for example, 1997). [ <i>tm_year</i> ]	
44326	%z	Replaced by the offset from UTC in the ISO 8601:2000 standard format (+hhmm or	
44327		-hhmm), or by no characters if no timezone is determinable. For example, "-0430"	
44328 CX		means 4 hours 30 minutes behind UTC (west of Greenwich). If <i>tm_isdst</i> is zero, the	
44329		standard time offset is used. If <i>tm_isdst</i> is greater than zero, the daylight savings time	
44330		offset is used. If <i>tm_isdst</i> is negative, no characters are returned. [ <i>tm_isdst</i> ]	

44331	%Z	Replaced by the timezone name or abbreviation, or by no bytes if no timezone information exists. [ <i>tm_isdst</i> ]	
44332			
44333	%%	Replaced by %.	
44334		If a conversion specification does not correspond to any of the above, the behavior is undefined.	
44335		<b>Modified Conversion Specifiers</b>	
44336		Some conversion specifiers can be modified by the <b>E</b> or <b>O</b> modifier characters to indicate that an alternative format or specification should be used rather than the one normally used by the unmodified conversion specifier. If the alternative format or specification does not exist for the current locale, (see ERA in the Base Definitions volume of IEEE Std 1003.1-200x, Section 7.3.5, LC_TIME) the behavior shall be as if the unmodified conversion specification were used.	
44337			
44338			
44339			
44340			
44341	%Ec	Replaced by the locale's alternative appropriate date and time representation.	
44342	%EC	Replaced by the name of the base year (period) in the locale's alternative representation.	
44343			
44344	%Ex	Replaced by the locale's alternative date representation.	
44345	%EX	Replaced by the locale's alternative time representation.	
44346	%Ey	Replaced by the offset from %EC (year only) in the locale's alternative representation.	
44347	%EY	Replaced by the full alternative year representation.	
44348	%Od	Replaced by the day of the month, using the locale's alternative numeric symbols, filled as needed with leading zeros if there is any alternative symbol for zero; otherwise, with leading spaces.	
44349			
44350			
44351	%Oe	Replaced by the day of the month, using the locale's alternative numeric symbols, filled as needed with leading spaces.	
44352			
44353	%OH	Replaced by the hour (24-hour clock) using the locale's alternative numeric symbols.	
44354	%OI	Replaced by the hour (12-hour clock) using the locale's alternative numeric symbols.	
44355	%Om	Replaced by the month using the locale's alternative numeric symbols.	
44356	%OM	Replaced by the minutes using the locale's alternative numeric symbols.	
44357	%OS	Replaced by the seconds using the locale's alternative numeric symbols.	
44358	%Ou	Replaced by the weekday as a number in the locale's alternative representation (Monday=1).	
44359			
44360	%OU	Replaced by the week number of the year (Sunday as the first day of the week, rules corresponding to %U) using the locale's alternative numeric symbols.	
44361			
44362	%OV	Replaced by the week number of the year (Monday as the first day of the week, rules corresponding to %V) using the locale's alternative numeric symbols.	
44363			
44364	%Ow	Replaced by the number of the weekday (Sunday=0) using the locale's alternative numeric symbols.	
44365			
44366	%OW	Replaced by the week number of the year (Monday as the first day of the week) using the locale's alternative numeric symbols.	
44367			
44368	%Oy	Replaced by the year (offset from %C) using the locale's alternative numeric symbols.	
44369	%g, %G, and %V	give values according to the ISO 8601:2000 standard week-based year. In this system, weeks begin on a Monday and week 1 of the year is the week that includes January 4th,	
44370			

44371 which is also the week that includes the first Thursday of the year, and is also the first week that  
 44372 contains at least four days in the year. If the first Monday of January is the 2nd, 3rd, or 4th, the  
 44373 preceding days are part of the last week of the preceding year; thus, for Saturday 2nd January  
 44374 1999, %G is replaced by 1998 and %V is replaced by 53. If December 29th, 30th, or 31st is a  
 44375 Monday, it and any following days are part of week 1 of the following year. Thus, for Tuesday  
 44376 30th December 1997, %G is replaced by 1998 and %V is replaced by 01.

44377 If a conversion specifier is not one of the above, the behavior is undefined. |

#### 44378 RETURN VALUE

44379 If the total number of resulting bytes including the terminating null byte is not more than  
 44380 *maxsize*, *strptime()* shall return the number of bytes placed into the array pointed to by *s*, not  
 44381 including the terminating null byte. Otherwise, 0 shall be returned and the contents of the array  
 44382 are unspecified. |

#### 44383 ERRORS

44384 No errors are defined.

#### 44385 EXAMPLES

##### 44386 Getting a Localized Date String

44387 The following example first sets the locale to the user's default. The locale information will be  
 44388 used in the *nl\_langinfo()* and *strptime()* functions. The *nl\_langinfo()* function returns the localized  
 44389 date string which specifies how the date is laid out. The *strptime()* function takes this information  
 44390 and, using the **tm** structure for values, places the date and time information into *datestring*.

```
44391 #include <time.h>
44392 #include <locale.h>
44393 #include <langinfo.h>
44394 ...
44395 struct tm *tm;
44396 char datestring[256];
44397 ...
44398 setlocale (LC_ALL, "");
44399 ...
44400 strptime (datestring, sizeof(datestring), nl_langinfo (D_T_FMT), tm);
44401 ...
```

#### 44402 APPLICATION USAGE

44403 The range of values for %S is [00,60] rather than [00,59] to allow for the occasional leap second.

44404 Some of the conversion specifications are duplicates of others. They are included for |  
 44405 compatibility with *nl\_cxtime()* and *nl\_ascxtime()*, which were published in Issue 2. |

44406 Applications should use %Y (4-digit years) in preference to %y (2-digit years).

44407 In the C locale, the E and O modifiers are ignored and the replacement strings for the following  
 44408 specifiers are:

44409	%a	The first three characters of %A.
44410	%A	One of Sunday, Monday, ..., Saturday.
44411	%b	The first three characters of %B.
44412	%B	One of January, February, ..., December.
44413	%c	Equivalent to %a %b %e %T %Y.

44414	%p	One of AM or PM.	
44415	%r	Equivalent to %I:%M:%S %p.	
44416	%x	Equivalent to %m/%d/%Y.	
44417	%X	Equivalent to %T.	
44418	%Z	Implementation-defined.	
44419	<b>RATIONALE</b>		
44420		None.	
44421	<b>FUTURE DIRECTIONS</b>		
44422		None.	
44423	<b>SEE ALSO</b>		
44424		<i>asctime()</i> , <i>clock()</i> , <i>ctime()</i> , <i>difftime()</i> , <i>getdate()</i> , <i>gmtime()</i> , <i>localtime()</i> , <i>mktime()</i> , <i>strptime()</i> , <i>time()</i> ,	
44425		<i>tzset()</i> , <i>utime()</i> , the Base Definitions volume of IEEE Std 1003.1-200x, < <b>time.h</b> >	
44426	<b>CHANGE HISTORY</b>		
44427		First released in Issue 3.	
44428	<b>Issue 5</b>		
44429		The description of %OV is changed to be consistent with %v and defines Monday as the first day	
44430		of the week.	
44431		The description of %Oy is clarified.	
44432	<b>Issue 6</b>		
44433		Extensions beyond the ISO C standard are now marked.	
44434		The Open Group Corrigendum U033/8 is applied. The %V conversion specifier is changed from	
44435		“Otherwise, it is week 53 of the previous year, and the next week is week 1” to “Otherwise, it is	
44436		the last week of the previous year, and the next week is week 1”.	
44437		The following new requirements on POSIX implementations derive from alignment with the	
44438		Single UNIX Specification:	
44439		<ul style="list-style-type: none"> <li>• The %C, %D, %e, %h, %n, %r, %R, %t, and %T conversion specifiers are added.</li> </ul>	
44440		<ul style="list-style-type: none"> <li>• The modified conversion specifiers are added for consistency with the ISO POSIX-2 standard</li> </ul>	
44441		<i>date</i> utility.	
44442		The following changes are made for alignment with the ISO/IEC 9899: 1999 standard:	
44443		<ul style="list-style-type: none"> <li>• The <i>strptime()</i> prototype is updated.</li> </ul>	
44444		<ul style="list-style-type: none"> <li>• The DESCRIPTION is extensively revised.</li> </ul>	
44445		<ul style="list-style-type: none"> <li>• The %z conversion specifier is added.</li> </ul>	
44446		A new example is added.	

44447 **NAME**

44448            strlen — get string length

44449 **SYNOPSIS**

44450            #include &lt;string.h&gt;

44451            size\_t strlen(const char \*s);

44452 **DESCRIPTION**

44453 cx        The functionality described on this reference page is aligned with the ISO C standard. Any  
 44454        conflict between the requirements described here and the ISO C standard is unintentional. This  
 44455        volume of IEEE Std 1003.1-200x defers to the ISO C standard.

44456        The *strlen()* function shall compute the number of bytes in the string to which *s* points, not  
 44457        including the terminating null byte.

44458 **RETURN VALUE**

44459        The *strlen()* function shall return the length of *s*; no return value shall be reserved to indicate an  
 44460        error.

44461 **ERRORS**

44462        No errors are defined.

44463 **EXAMPLES**44464            **Getting String Lengths**

44465        The following example sets the maximum length of *key* and *data* by using *strlen()* to get the  
 44466        lengths of those strings.

```

44467        #include <string.h>
44468        ...
44469        struct element {
44470            char *key;
44471            char *data;
44472        };
44473        ...
44474        char *key, *data;
44475        int len;

44476        *keylength = *datalength = 0;
44477        ...
44478        if ((len = strlen(key)) > *keylength)
44479            *keylength = len;
44480        if ((len = strlen(data)) > *datalength)
44481            *datalength = len;
44482        ...
```

44483 **APPLICATION USAGE**

44484        None.

44485 **RATIONALE**

44486        None.

44487 **FUTURE DIRECTIONS**

44488        None.

44489 **SEE ALSO**44490           The Base Definitions volume of IEEE Std 1003.1-200x, <**string.h**>44491 **CHANGE HISTORY**

44492           First released in Issue 1. Derived from Issue 1 of the SVID.

44493 **Issue 5**44494           The RETURN VALUE section is updated to indicate that *strlen()* returns the length of *s*, and not  
44495           *s* itself as was previously stated.

44496 **NAME**

44497       strncasecmp — case-insensitive string comparison

44498 **SYNOPSIS**

44499 XSI       #include &lt;strings.h&gt;

44500       int strncasecmp(const char \*s1, const char \*s2, size\_t n);

44501

44502 **DESCRIPTION**44503       Refer to *strcasecmp()*.

44504 **NAME**

44505           strncat — concatenate a string with part of another

44506 **SYNOPSIS**

44507           #include <string.h>

44508           char \*strncat(char \*restrict *s1*, const char \*restrict *s2*, size\_t *n*);

44509 **DESCRIPTION**

44510 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
44511 conflict between the requirements described here and the ISO C standard is unintentional. This  
44512 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

44513       The *strncat()* function shall append not more than *n* bytes (a null byte and bytes that follow it  
44514 are not appended) from the array pointed to by *s2* to the end of the string pointed to by *s1*. The  
44515 initial byte of *s2* overwrites the null byte at the end of *s1*. A terminating null byte is always  
44516 appended to the result. If copying takes place between objects that overlap, the behavior is  
44517 undefined.

44518 **RETURN VALUE**

44519       The *strncat()* function shall return *s1*; no return value shall be reserved to indicate an error.

44520 **ERRORS**

44521       No errors are defined.

44522 **EXAMPLES**

44523       None.

44524 **APPLICATION USAGE**

44525       None.

44526 **RATIONALE**

44527       None.

44528 **FUTURE DIRECTIONS**

44529       None.

44530 **SEE ALSO**

44531       *strcat()*, the Base Definitions volume of IEEE Std 1003.1-200x, <**string.h**>

44532 **CHANGE HISTORY**

44533       First released in Issue 1. Derived from Issue 1 of the SVID.

44534 **Issue 6**

44535       The *strncat()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

44536 **NAME**

44537 strncmp — compare part of two strings

44538 **SYNOPSIS**

44539 #include &lt;string.h&gt;

44540 int strncmp(const char \*s1, const char \*s2, size\_t n);

44541 **DESCRIPTION**

44542 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
44543 conflict between the requirements described here and the ISO C standard is unintentional. This  
44544 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

44545 The *strncmp()* function shall compare not more than *n* bytes (bytes that follow a null byte are not  
44546 compared) from the array pointed to by *s1* to the array pointed to by *s2*.

44547 The sign of a non-zero return value is determined by the sign of the difference between the  
44548 values of the first pair of bytes (both interpreted as type **unsigned char**) that differ in the strings  
44549 being compared.

44550 **RETURN VALUE**

44551 Upon successful completion, *strncmp()* shall return an integer greater than, equal to, or less than  
44552 0, if the possibly null-terminated array pointed to by *s1* is greater than, equal to, or less than the  
44553 possibly null-terminated array pointed to by *s2* respectively.

44554 **ERRORS**

44555 No errors are defined.

44556 **EXAMPLES**

44557 None.

44558 **APPLICATION USAGE**

44559 None.

44560 **RATIONALE**

44561 None.

44562 **FUTURE DIRECTIONS**

44563 None.

44564 **SEE ALSO**44565 *strcmp()*, the Base Definitions volume of IEEE Std 1003.1-200x, <**string.h**>44566 **CHANGE HISTORY**

44567 First released in Issue 1. Derived from Issue 1 of the SVID.

44568 **Issue 6**

44569 Extensions beyond the ISO C standard are now marked.

44570 **NAME**

44571       strncpy — copy part of a string

44572 **SYNOPSIS**

44573       #include <string.h>

44574       char \*strncpy(char \*restrict *s1*, const char \*restrict *s2*, size\_t *n*);

44575 **DESCRIPTION**

44576 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
44577 conflict between the requirements described here and the ISO C standard is unintentional. This  
44578 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

44579       The *strncpy()* function shall copy not more than *n* bytes (bytes that follow a null byte are not  
44580 copied) from the array pointed to by *s2* to the array pointed to by *s1*. If copying takes place  
44581 between objects that overlap, the behavior is undefined.

44582       If the array pointed to by *s2* is a string that is shorter than *n* bytes, null bytes shall be appended  
44583 to the copy in the array pointed to by *s1*, until *n* bytes in all are written.

44584 **RETURN VALUE**

44585       The *strncpy()* function shall return *s1*; no return value is reserved to indicate an error.

44586 **ERRORS**

44587       No errors are defined.

44588 **EXAMPLES**

44589       None.

44590 **APPLICATION USAGE**

44591       Character movement is performed differently in different implementations. Thus, overlapping  
44592 moves may yield surprises.

44593       If there is no null byte in the first *n* bytes of the array pointed to by *s2*, the result is not null-  
44594 terminated.

44595 **RATIONALE**

44596       None.

44597 **FUTURE DIRECTIONS**

44598       None.

44599 **SEE ALSO**

44600       *strcpy()*, the Base Definitions volume of IEEE Std 1003.1-200x, <**string.h**>

44601 **CHANGE HISTORY**

44602       First released in Issue 1. Derived from Issue 1 of the SVID.

44603 **Issue 6**

44604       The *strncpy()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

44605 **NAME**

44606 strpbrk — scan string for byte

44607 **SYNOPSIS**

44608 #include &lt;string.h&gt;

44609 char \*strpbrk(const char \*s1, const char \*s2);

44610 **DESCRIPTION**

44611 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
44612 conflict between the requirements described here and the ISO C standard is unintentional. This  
44613 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

44614 The *strpbrk()* function shall locate the first occurrence in the string pointed to by *s1* of any byte  
44615 from the string pointed to by *s2*.

44616 **RETURN VALUE**

44617 Upon successful completion, *strpbrk()* shall return a pointer to the byte or a null pointer if no  
44618 byte from *s2* occurs in *s1*.

44619 **ERRORS**

44620 No errors are defined.

44621 **EXAMPLES**

44622 None.

44623 **APPLICATION USAGE**

44624 None.

44625 **RATIONALE**

44626 None.

44627 **FUTURE DIRECTIONS**

44628 None.

44629 **SEE ALSO**44630 *strchr()*, *strchr()*, the Base Definitions volume of IEEE Std 1003.1-200x, <string.h>44631 **CHANGE HISTORY**

44632 First released in Issue 1. Derived from Issue 1 of the SVID.

## 44633 NAME

44634 strptime — date and time conversion

## 44635 SYNOPSIS

44636 XSI #include &lt;time.h&gt;

```
44637 char *strptime(const char *restrict buf, const char *restrict format,
44638               struct tm *restrict tm);
44639
```

## 44640 DESCRIPTION

44641 The *strptime()* function shall convert the character string pointed to by *buf* to values which are  
 44642 stored in the **tm** structure pointed to by *tm*, using the format specified by *format*.

44643 The *format* is composed of zero or more directives. Each directive is composed of one of the  
 44644 following: one or more white-space characters (as specified by *isspace()*); an ordinary character  
 44645 (neither '%' nor a white-space character); or a conversion specification. Each conversion  
 44646 specification is composed of a '%' character followed by a conversion character which specifies  
 44647 the replacement required. The application shall ensure that there is white-space or other non-  
 44648 alphanumeric characters between any two conversion specifications. The following conversion  
 44649 specifications are supported:

44650	%a	The day of the week, using the locale's weekday names; either the abbreviated or full name may be specified.	
44651			
44652	%A	Equivalent to %a.	
44653	%b	The month, using the locale's month names; either the abbreviated or full name may be specified.	
44654			
44655	%B	Equivalent to %b.	
44656	%c	Replaced by the locale's appropriate date and time representation.	
44657	%C	The century number [0,99]; leading zeros are permitted but not required.	
44658	%d	The day of the month [1,31]; leading zeros are permitted but not required.	
44659	%D	The date as %m/%d/%y.	
44660	%e	Equivalent to %d.	
44661	%h	Equivalent to %b.	
44662	%H	The hour (24-hour clock) [0,23]; leading zeros are permitted but not required.	
44663	%I	The hour (12-hour clock) [1,12]; leading zeros are permitted but not required.	
44664	%j	The day number of the year [1,366]; leading zeros are permitted but not required.	
44665	%m	The month number [1,12]; leading zeros are permitted but not required.	
44666	%M	The minute [0,59]; leading zeros are permitted but not required.	
44667	%n	Any white space.	
44668	%p	The locale's equivalent of a.m or p.m.	
44669	%r	12-hour clock time using the AM/PM notation if <b>t_fmt_ampm</b> is not an empty string in the LC_TIME portion of the current locale; in the POSIX locale, this shall be equivalent	
44670		to %I:%M:%S %p.	
44671			
44672	%R	The time as %H:%M.	

44673	%S	The seconds [0,60]; leading zeros are permitted but not required.
44674	%t	Any white space.
44675	%T	The time as %H:%M:%S.
44676	%U	The week number of the year (Sunday as the first day of the week) as a decimal number [00,53]; leading zeros are permitted but not required.
44678	%w	The weekday as a decimal number [0,6], with 0 representing Sunday; leading zeros are permitted but not required.
44680	%W	The week number of the year (Monday as the first day of the week) as a decimal number [00,53]; leading zeros are permitted but not required.
44682	%x	The date, using the locale's date format.
44683	%X	The time, using the locale's time format.
44684	%y	The year within century. When a century is not otherwise specified, values in the range [69,99] shall refer to years 1969 to 1999 inclusive, and values in the range [00,68] shall refer to years 2000 to 2068 inclusive; leading zeros shall be permitted but shall not be required.
44688	<b>Note:</b>	It is expected that in a future version of IEEE Std 1003.1-200x the default century inferred from a 2-digit year will change. (This would apply to all commands accepting a 2-digit year as input.)
44691	%Y	The year, including the century (for example, 1988).
44692	%%	Replaced by %.

#### 44693 **Modified Conversion Specifiers**

44694		Some conversion specifiers can be modified by the <b>E</b> and <b>O</b> modifier characters to indicate that an alternative format or specification should be used rather than the one normally used by the unmodified conversion specifier. If the alternative format or specification does not exist in the current locale, the behavior shall be as if the unmodified conversion specification were used.
44698	%Ec	The locale's alternative appropriate date and time representation.
44699	%EC	The name of the base year (period) in the locale's alternative representation.
44700	%Ex	The locale's alternative date representation.
44701	%EX	The locale's alternative time representation.
44702	%Ey	The offset from %EC (year only) in the locale's alternative representation.
44703	%EY	The full alternative year representation.
44704	%Od	The day of the month using the locale's alternative numeric symbols; leading zeros are permitted but not required.
44706	%Oe	Equivalent to %Od.
44707	%OH	The hour (24-hour clock) using the locale's alternative numeric symbols.
44708	%OI	The hour (12-hour clock) using the locale's alternative numeric symbols.
44709	%Om	The month using the locale's alternative numeric symbols.
44710	%OM	The minutes using the locale's alternative numeric symbols.

44711	%OS	The seconds using the locale's alternative numeric symbols.
44712	%OU	The week number of the year (Sunday as the first day of the week) using the locale's alternative numeric symbols.
44713		
44714	%Ow	The number of the weekday (Sunday=0) using the locale's alternative numeric symbols.
44715	%OW	The week number of the year (Monday as the first day of the week) using the locale's alternative numeric symbols.
44716		
44717	%Oy	The year (offset from %C) using the locale's alternative numeric symbols.
44718		A conversion specification composed of white-space characters is executed by scanning input up to the first character that is not white-space (which remains unscanned), or until no more characters can be scanned.
44719		
44720		
44721		A conversion specification that is an ordinary character is executed by scanning the next character from the buffer. If the character scanned from the buffer differs from the one comprising the directive, the directive fails, and the differing and subsequent characters remain unscanned.
44722		
44723		
44724		
44725		A series of conversion specifications composed of %n, %t, white-space characters, or any combination is executed by scanning up to the first character that is not white space (which remains unscanned), or until no more characters can be scanned.
44726		
44727		
44728		Any other conversion specification is executed by scanning characters until a character matching the next directive is scanned, or until no more characters can be scanned. These characters, except the one matching the next directive, are then compared to the locale values associated with the conversion specifier. If a match is found, values for the appropriate <b>tm</b> structure members are set to values corresponding to the locale information. Case is ignored when matching items in <i>buf</i> such as month or weekday names. If no match is found, <i>strptime()</i> fails and no more characters are scanned.
44729		
44730		
44731		
44732		
44733		
44734		
44735	<b>RETURN VALUE</b>	
44736		Upon successful completion, <i>strptime()</i> shall return a pointer to the character following the last character parsed. Otherwise, a null pointer shall be returned.
44737		
44738	<b>ERRORS</b>	
44739		No errors are defined.
44740	<b>EXAMPLES</b>	
44741		None.
44742	<b>APPLICATION USAGE</b>	
44743		Several "equivalent to" formats and the special processing of white-space characters are provided in order to ease the use of identical <i>format</i> strings for <i>strptime()</i> and <i>strptime()</i> .
44744		
44745		Applications should use %Y (4-digit years) in preference to %y (2-digit years).
44746		It is unspecified whether multiple calls to <i>strptime()</i> using the same <b>tm</b> structure will update the current contents of the structure or overwrite all contents of the structure. Conforming applications should make a single call to <i>strptime()</i> with a format and all data needed to completely specify the date and time being converted.
44747		
44748		
44749		
44750	<b>RATIONALE</b>	
44751		None.

44752 **FUTURE DIRECTIONS**

44753           The *strptime()* function is expected to be mandatory in the next version of this volume of  
44754           IEEE Std 1003.1-200x.

44755 **SEE ALSO**

44756           *scanf()*, *strptime()*, *time()*, the Base Definitions volume of IEEE Std 1003.1-200x, <**time.h**>

44757 **CHANGE HISTORY**

44758           First released in Issue 4.

44759 **Issue 5**

44760           Moved from ENHANCED I18N to BASE.

44761           The [ENOSYS] error is removed.

44762           The exact meaning of the %y and %Oy specifiers are clarified in the DESCRIPTION.

44763 **Issue 6**

44764           The Open Group Corrigendum U033/5 is applied. The %r specifier description is reworded.

44765           The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

44766           The **restrict** keyword is added to the *strptime()* prototype for alignment with the  
44767           ISO/IEC 9899:1999 standard.

44768           The Open Group Corrigendum U047/2 is applied.

44769           The DESCRIPTION is updated to use the terms “conversion specifier” and “conversion  
44770           specification” for consistency with *strptime()*.

44771 **NAME**

44772       strchr — string scanning operation

44773 **SYNOPSIS**

44774       #include &lt;string.h&gt;

44775       char \*strchr(const char \*s, int c);

44776 **DESCRIPTION**

44777 CX       The functionality described on this reference page is aligned with the ISO C standard. Any  
44778       conflict between the requirements described here and the ISO C standard is unintentional. This  
44779       volume of IEEE Std 1003.1-200x defers to the ISO C standard.

44780 CX       The *strchr()* function shall locate the last occurrence of *c* (converted to an **unsigned char**) in the  
44781       string pointed to by *s*. The terminating null byte is considered to be part of the string.

44782 **RETURN VALUE**

44783       Upon successful completion, *strchr()* shall return a pointer to the byte or a null pointer if *c* does  
44784       not occur in the string.

44785 **ERRORS**

44786       No errors are defined.

44787 **EXAMPLES**44788       **Finding the Base Name of a File**

44789       The following example uses *strchr()* to get a pointer to the base name of a file. The *strchr()*  
44790       function searches backwards through the name of the file to find the last '/' character in *name*.  
44791       This pointer (plus one) will point to the base name of the file.

```
44792       #include <string.h>
44793       ...
44794       const char *name;
44795       char *basename;
44796       ...
44797       basename = strchr(name, '/') + 1;
44798       ...
```

44799 **APPLICATION USAGE**

44800       None.

44801 **RATIONALE**

44802       None.

44803 **FUTURE DIRECTIONS**

44804       None.

44805 **SEE ALSO**44806       *strchr()*, the Base Definitions volume of IEEE Std 1003.1-200x, <string.h>44807 **CHANGE HISTORY**

44808       First released in Issue 1. Derived from Issue 1 of the SVID.

44809 **NAME**

44810 strspn — get length of a substring

44811 **SYNOPSIS**

44812 #include &lt;string.h&gt;

44813 size\_t strspn(const char \*s1, const char \*s2);

44814 **DESCRIPTION**

44815 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
44816 conflict between the requirements described here and the ISO C standard is unintentional. This  
44817 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

44818 The *strspn()* function shall compute the length (in bytes) of the maximum initial segment of the  
44819 string pointed to by *s1* which consists entirely of bytes from the string pointed to by *s2*.

44820 **RETURN VALUE**

44821 The *strspn()* function shall return the length of *s1*; no return value is reserved to indicate an  
44822 error.

44823 **ERRORS**

44824 No errors are defined.

44825 **EXAMPLES**

44826 None.

44827 **APPLICATION USAGE**

44828 None.

44829 **RATIONALE**

44830 None.

44831 **FUTURE DIRECTIONS**

44832 None.

44833 **SEE ALSO**44834 *strcspn()*, the Base Definitions volume of IEEE Std 1003.1-200x, <string.h>44835 **CHANGE HISTORY**

44836 First released in Issue 1. Derived from Issue 1 of the SVID.

44837 **Issue 5**

44838 The RETURN VALUE section is updated to indicate that *strspn()* returns the length of *s*, and not  
44839 *s* itself as was previously stated.

44840 **NAME**44841        `strstr` — find a substring44842 **SYNOPSIS**44843        `#include <string.h>`44844        `char *strstr(const char *s1, const char *s2);`44845 **DESCRIPTION**

44846 **CX**        The functionality described on this reference page is aligned with the ISO C standard. Any  
44847        conflict between the requirements described here and the ISO C standard is unintentional. This  
44848        volume of IEEE Std 1003.1-200x defers to the ISO C standard.

44849        The `strstr()` function shall locate the first occurrence in the string pointed to by `s1` of the  
44850        sequence of bytes (excluding the terminating null byte) in the string pointed to by `s2`.

44851 **RETURN VALUE**

44852        Upon successful completion, `strstr()` shall return a pointer to the located string or a null pointer  
44853        if the string is not found.

44854        If `s2` points to a string with zero length, the function shall return `s1`.

44855 **ERRORS**

44856        No errors are defined.

44857 **EXAMPLES**

44858        None.

44859 **APPLICATION USAGE**

44860        None.

44861 **RATIONALE**

44862        None.

44863 **FUTURE DIRECTIONS**

44864        None.

44865 **SEE ALSO**

44866        `strchr()`, the Base Definitions volume of IEEE Std 1003.1-200x, `<string.h>`

44867 **CHANGE HISTORY**

44868        First released in Issue 3.

44869        Entry included for alignment with the ANSI C standard.

## 44870 NAME

44871 strtod, strtodf, strtold — convert string to a double-precision number

## 44872 SYNOPSIS

44873 #include &lt;stdlib.h&gt;

44874 double strtod(const char \*restrict *nptr*, char \*\*restrict *endp*);44875 float strtodf(const char \*restrict *nptr*, char \*\*restrict *endp*);44876 long double strtold(const char \*restrict *nptr*, char \*\*restrict *endp*);

## 44877 DESCRIPTION

44878 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
 44879 conflict between the requirements described here and the ISO C standard is unintentional. This  
 44880 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

44881 These functions shall convert the initial portion of the string pointed to by *nptr* to **double**, **float**,  
 44882 and **long double** representation, respectively. First, they decompose the input string into three  
 44883 parts:

- 44884 1. An initial, possibly empty, sequence of white-space characters (as specified by *isspace()*)
- 44885 2. A subject sequence interpreted as a floating-point constant or representing infinity or NaN
- 44886 3. A final string of one or more unrecognized characters, including the terminating null byte  
 44887 of the input string

44888 Then they shall attempt to convert the subject sequence to a floating-point number, and return  
 44889 the result.

44890 The expected form of the subject sequence is an optional plus or minus sign, then one of the  
 44891 following:

- 44892 • A non-empty sequence of decimal digits optionally containing a radix character, then an  
 44893 optional exponent part
- 44894 • A 0x or 0X, then a non-empty sequence of hexadecimal digits optionally containing a radix  
 44895 character, then an optional binary exponent part
- 44896 • One of INF or INFINITY, ignoring case
- 44897 • One of NAN or NAN(*n-char-sequence<sub>opt</sub>*), ignoring case in the NAN part, where:

```
44898 n-char-sequence:
44899     digit
44900     nondigit
44901     n-char-sequence digit
44902     n-char-sequence nondigit
```

44903 The subject sequence is defined as the longest initial subsequence of the input string, starting  
 44904 with the first non-white-space character, that is of the expected form. The subject sequence  
 44905 contains no characters if the input string is not of the expected form.

44906 If the subject sequence has the expected form for a floating-point number, the sequence of  
 44907 characters starting with the first digit or the decimal-point character (whichever occurs first)  
 44908 shall be interpreted as a floating constant of the C language, except that the radix character shall  
 44909 be used in place of a period, and that if neither an exponent part nor a radix character appears in  
 44910 a decimal floating-point number, or if a binary exponent part does not appear in a hexadecimal  
 44911 floating-point number, an exponent part of the appropriate type with value zero is assumed to  
 44912 follow the last digit in the string. If the subject sequence begins with a minus sign, the sequence  
 44913 shall be interpreted as negated. A character sequence INF or INFINITY shall be interpreted as an

44914 infinity, if representable in the return type, else as if it were a floating constant that is too large |  
 44915 for the range of the return type. A character sequence NAN or NAN(*n-char-sequence<sub>opt</sub>*) shall be |  
 44916 interpreted as a quiet NaN, if supported in the return type, else as if it were a subject sequence |  
 44917 part that does not have the expected form; the meaning of the *n-char* sequences is |  
 44918 implementation-defined. A pointer to the final string is stored in the object pointed to by *endptr*, |  
 44919 provided that *endptr* is not a null pointer.

44920 If the subject sequence has the hexadecimal form and FLT\_RADIX is a power of 2, the value |  
 44921 resulting from the conversion is correctly rounded.

44922 CX The radix character is defined in the program's locale (category *LC\_NUMERIC*). In the POSIX |  
 44923 locale, or in a locale where the radix character is not defined, the radix character shall default to a |  
 44924 period ('.').

44925 CX In other than the C or POSIX locales, other implementation-defined subject sequences may be |  
 44926 accepted.

44927 If the subject sequence is empty or does not have the expected form, no conversion shall be |  
 44928 performed; the value of *str* is stored in the object pointed to by *endptr*, provided that *endptr* is not |  
 44929 a null pointer.

44930 CX The *strtod()* function shall not change the setting of *errno* if successful.

44931 Since 0 is returned on error and is also a valid return on success, an application wishing to check |  
 44932 for error situations should set *errno* to 0, then call *strtod()*, *strtof()*, or *strtold()*, then check *errno*.

#### 44933 RETURN VALUE

44934 Upon successful completion, these functions shall return the converted value. If no conversion |  
 44935 could be performed, 0 shall be returned, and *errno* may be set to [EINVAL].

44936 If the correct value is outside the range of representable values, HUGE\_VAL, HUGE\_VALF, or |  
 44937 HUGE\_VALL shall be returned (according to the sign of the value), and *errno* shall be set to |  
 44938 [ERANGE].

44939 If the correct value would cause an underflow, a value whose magnitude is no greater than the |  
 44940 smallest normalized positive number in the return type shall be returned and *errno* set to |  
 44941 [ERANGE].

#### 44942 ERRORS

44943 These functions shall fail if:

44944 CX [ERANGE] The value to be returned would cause overflow or underflow.

44945 These functions may fail if:

44946 CX [EINVAL] No conversion could be performed.

#### 44947 EXAMPLES

44948 None.

#### 44949 APPLICATION USAGE

44950 If the subject sequence has the hexadecimal form and FLT\_RADIX is not a power of 2, and the |  
 44951 result is not exactly representable, the result should be one of the two numbers in the |  
 44952 appropriate internal format that are adjacent to the hexadecimal floating source value, with the |  
 44953 extra stipulation that the error should have a correct sign for the current rounding direction. |

44954 If the subject sequence has the decimal form and at most DECIMAL\_DIG (defined in <float.h>) |  
 44955 significant digits, the result should be correctly rounded. If the subject sequence *D* has the |  
 44956 decimal form and more than DECIMAL\_DIG significant digits, consider the two bounding, |  
 44957 adjacent decimal strings *L* and *U*, both having DECIMAL\_DIG significant digits, such that the

44958 values of  $L$ ,  $D$ , and  $U$  satisfy  $L \leq D \leq U$ . The result should be one of the (equal or adjacent)  
 44959 values that would be obtained by correctly rounding  $L$  and  $U$  according to the current rounding  
 44960 direction, with the extra stipulation that the error with respect to  $D$  should have a correct sign  
 44961 for the current rounding direction.

44962 The changes to `strtod()` introduced by the ISO/IEC 9899:1999 standard can alter the behavior of  
 44963 well-formed applications complying with the ISO/IEC 9899:1990 standard and thus earlier  
 44964 versions of IEEE Std 1003.1-200x. One such example would be:

```

44965 int
44966 what_kind_of_number (char *s)
44967 {
44968     char *endp;
44969     double d;
44970     long l;

44971     d = strtod(s, &endp);
44972     if (s != endp && *endp == '\0')
44973         printf("It's a float with value %g\n", d);
44974     else
44975     {
44976         l = strtol(s, &endp, 0);
44977         if (s != endp && *endp == '\0')
44978             printf("It's an integer with value %ld\n", l);
44979         else
44980             return 1;
44981     }
44982     return 0;
44983 }

```

44984 If the function is called with:

```
44985 what_kind_of_number ("0x10")
```

44986 an ISO/IEC 9899:1990 standard-compliant library will result in the function printing:

```
44987 It's an integer with value 16
```

44988 With the ISO/IEC 9899:1999 standard, the result is:

```
44989 It's a float with value 16
```

44990 The change in behavior is due to the inclusion of floating-point numbers in hexadecimal  
 44991 notation without requiring that either a decimal point or the binary exponent be present.

#### 44992 RATIONALE

44993 None.

#### 44994 FUTURE DIRECTIONS

44995 None.

#### 44996 SEE ALSO

44997 `isspace()`, `localeconv()`, `scanf()`, `setlocale()`, `strtol()`, the Base Definitions volume of  
 44998 IEEE Std 1003.1-200x, `<float.h>`, `<stdlib.h>`, the Base Definitions volume of  
 44999 IEEE Std 1003.1-200x, Chapter 7, Locale

45000 **CHANGE HISTORY**

45001 First released in Issue 1. Derived from Issue 1 of the SVID.

45002 **Issue 5**

45003 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.

45004 **Issue 6**

45005 Extensions beyond the ISO C standard are now marked.

45006 The following new requirements on POSIX implementations derive from alignment with the  
45007 Single UNIX Specification:

45008 • In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is  
45009 added if no conversion could be performed.

45010 The following changes are made for alignment with the ISO/IEC 9899: 1999 standard:

45011 • The *strtod()* function is updated.

45012 • The *strtof()* and *strtold()* functions are added.

45013 • The DESCRIPTION is extensively revised.

45014 ISO/IEC 9899: 1999 standard, Technical Corrigendum No. 1 is incorporated.

45015 **NAME**45016        `strtod` — convert string to a double-precision number45017 **SYNOPSIS**45018        `#include <stdlib.h>`45019        `float strtod(const char *restrict nptr, char **restrict endptr);`45020 **DESCRIPTION**45021        Refer to `strtod()`.

45022 **NAME**

45023 strtoimax, strtoumax — convert string to integer type

45024 **SYNOPSIS**

45025 #include <inttypes.h>

45026 intmax\_t strtoimax(const char \*restrict nptr, char \*\*restrict endptr,  
45027 int base);

45028 uintmax\_t strtoumax(const char \*restrict nptr, char \*\*restrict endptr,  
45029 int base);

45030 **DESCRIPTION**

45031 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
45032 conflict between the requirements described here and the ISO C standard is unintentional. This  
45033 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

45034 These functions shall be equivalent to the *strtol()*, *strtoll()*, *strtoul()*, and *strtoull()* functions,  
45035 except that the initial portion of the string shall be converted to **intmax\_t** and **uintmax\_t**  
45036 representation, respectively.

45037 **RETURN VALUE**

45038 These functions shall return the converted value, if any.

45039 If no conversion could be performed, zero shall be returned.

45040 If the correct value is outside the range of representable values, {INTMAX\_MAX},  
45041 {INTMAX\_MIN}, or {UINTMAX\_MAX} shall be returned (according to the return type and sign  
45042 of the value, if any), and *errno* shall be set to [ERANGE].

45043 **ERRORS**

45044 These functions shall fail if:

45045 [ERANGE] The value to be returned is not representable.

45046 These functions may fail if:

45047 [EINVAL] The value of *base* is not supported.

45048 **EXAMPLES**

45049 None.

45050 **APPLICATION USAGE**

45051 None.

45052 **RATIONALE**

45053 None.

45054 **FUTURE DIRECTIONS**

45055 None.

45056 **SEE ALSO**

45057 *strtol()*, *strtoul()*, the Base Definitions volume of IEEE Std 1003.1-200x, <inttypes.h>

45058 **CHANGE HISTORY**

45059 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

## 45060 NAME

45061 strtok, strtok\_r — split string into tokens

## 45062 SYNOPSIS

45063 #include &lt;string.h&gt;

45064 char \*strtok(char \*restrict s1, const char \*restrict s2);

45065 TSF char \*strtok\_r(char \*restrict s, const char \*restrict sep,

45066 char \*\*restrict lasts);

45067

## 45068 DESCRIPTION

45069 CX For *strtok()*: The functionality described on this reference page is aligned with the ISO C  
 45070 standard. Any conflict between the requirements described here and the ISO C standard is  
 45071 unintentional. This volume of IEEE Std 1003.1-200x defers to the ISO C standard.

45072 A sequence of calls to *strtok()* breaks the string pointed to by *s1* into a sequence of tokens, each  
 45073 of which is delimited by a byte from the string pointed to by *s2*. The first call in the sequence has  
 45074 *s1* as its first argument, and is followed by calls with a null pointer as their first argument. The  
 45075 separator string pointed to by *s2* may be different from call to call.

45076 The first call in the sequence searches the string pointed to by *s1* for the first byte that is *not*  
 45077 contained in the current separator string pointed to by *s2*. If no such byte is found, then there  
 45078 are no tokens in the string pointed to by *s1* and *strtok()* shall return a null pointer. If such a byte  
 45079 is found, it is the start of the first token.

45080 The *strtok()* function then searches from there for a byte that *is* contained in the current  
 45081 separator string. If no such byte is found, the current token extends to the end of the string  
 45082 pointed to by *s1*, and subsequent searches for a token shall return a null pointer. If such a byte is  
 45083 found, it is overwritten by a null byte, which terminates the current token. The *strtok()* function  
 45084 saves a pointer to the following byte, from which the next search for a token shall start.

45085 Each subsequent call, with a null pointer as the value of the first argument, starts searching from  
 45086 the saved pointer and behaves as described above.

45087 The implementation shall behave as if no function defined in this volume of  
 45088 IEEE Std 1003.1-200x calls *strtok()*.

45089 CX The *strtok()* function need not be reentrant. A function that is not required to be reentrant is not  
 45090 required to be thread-safe.

45091 TSF The *strtok\_r()* function considers the null-terminated string *s* as a sequence of zero or more text  
 45092 tokens separated by spans of one or more characters from the separator string *sep*. The  
 45093 argument *lasts* points to a user-provided pointer which points to stored information necessary  
 45094 for *strtok\_r()* to continue scanning the same string.

45095 In the first call to *strtok\_r()*, *s* points to a null-terminated string, *sep* to a null-terminated string of  
 45096 separator characters, and the value pointed to by *lasts* is ignored. The *strtok\_r()* function shall  
 45097 return a pointer to the first character of the first token, write a null character into *s* immediately  
 45098 following the returned token, and update the pointer to which *lasts* points.

45099 In subsequent calls, *s* is a NULL pointer and *lasts* shall be unchanged from the previous call so  
 45100 that subsequent calls shall move through the string *s*, returning successive tokens until no  
 45101 tokens remain. The separator string *sep* may be different from call to call. When no token  
 45102 remains in *s*, a NULL pointer shall be returned.

45103 **RETURN VALUE**

45104 Upon successful completion, *strtok()* shall return a pointer to the first byte of a token. Otherwise,  
45105 if there is no token, *strtok()* shall return a null pointer.

45106 TSF The *strtok\_r()* function shall return a pointer to the token found, or a NULL pointer when no  
45107 token is found.

45108 **ERRORS**

45109 No errors are defined.

45110 **EXAMPLES**45111 **Searching for Word Separators**

45112 The following example searches for tokens separated by space characters.

```
45113 #include <string.h>
45114 ...
45115 char *token;
45116 char *line = "LINE TO BE SEPARATED";
45117 char *search = " ";

45118 /* Token will point to "LINE". */
45119 token = strtok(line, search);

45120 /* Token will point to "TO". */
45121 token = strtok(NULL, search);
```

45122 **Breaking a Line**

45123 The following example uses *strtok()* to break a line into two character strings separated by any  
45124 combination of <space>s, <tab>s, or <newline>s.

```
45125 #include <string.h>
45126 ...
45127 struct element {
45128     char *key;
45129     char *data;
45130 };
45131 ...
45132 char line[LINE_MAX];
45133 char *key, *data;
45134 ...
45135 key = strtok(line, " \n");
45136 data = strtok(NULL, " \n");
45137 ...
```

45138 **APPLICATION USAGE**

45139 The *strtok\_r()* function is thread-safe and stores its state in a user-supplied buffer instead of  
45140 possibly using a static data area that may be overwritten by an unrelated call from another  
45141 thread.

45142 **RATIONALE**

45143 The *strtok()* function searches for a separator string within a larger string. It returns a pointer to  
45144 the last substring between separator strings. This function uses static storage to keep track of  
45145 the current string position between calls. The new function, *strtok\_r()*, takes an additional  
45146 argument, *lasts*, to keep track of the current position in the string.

45147 **FUTURE DIRECTIONS**

45148 None.

45149 **SEE ALSO**45150 The Base Definitions volume of IEEE Std 1003.1-200x, <**string.h**>45151 **CHANGE HISTORY**

45152 First released in Issue 1. Derived from Issue 1 of the SVID.

45153 **Issue 5**45154 The *strtok\_r()* function is included for alignment with the POSIX Threads Extension.45155 A note indicating that the *strtok()* function need not be reentrant is added to the DESCRIPTION.45156 **Issue 6**

45157 Extensions beyond the ISO C standard are now marked.

45158 The *strtok\_r()* function is marked as part of the Thread-Safe Functions option.

45159 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

45160 The APPLICATION USAGE section is updated to include a note on the thread-safe function and

45161 its avoidance of possibly using a static data area.

45162 The **restrict** keyword is added to the *strtok()* and *strtok\_r()* prototypes for alignment with the

45163 ISO/IEC 9899:1999 standard.

## 45164 NAME

45165 strtol, strtoll — convert string to a long integer

## 45166 SYNOPSIS

45167 #include &lt;stdlib.h&gt;

45168 long strtol(const char \*restrict *str*, char \*\*restrict *endptr*, int *base*);45169 long long strtoll(const char \*restrict *str*, char \*\*restrict *endptr*,45170 int *base*)

## 45171 DESCRIPTION

45172 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
 45173 conflict between the requirements described here and the ISO C standard is unintentional. This  
 45174 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

45175 These functions shall convert the initial portion of the string pointed to by *str* to a type **long** and  
 45176 **long long** representation, respectively. First, they decompose the input string into three parts:

- 45177 1. An initial, possibly empty, sequence of white-space characters (as specified by *isspace()*)
- 45178 2. A subject sequence interpreted as an integer represented in some radix determined by the  
 45179 value of *base*
- 45180 3. A final string of one or more unrecognized characters, including the terminating null byte  
 45181 of the input string.

45182 Then they shall attempt to convert the subject sequence to an integer, and return the result. |

45183 If the value of *base* is 0, the expected form of the subject sequence is that of a decimal constant,  
 45184 octal constant, or hexadecimal constant, any of which may be preceded by a '+' or '-' sign. A  
 45185 decimal constant begins with a non-zero digit, and consists of a sequence of decimal digits. An  
 45186 octal constant consists of the prefix '0' optionally followed by a sequence of the digits '0' to  
 45187 '7' only. A hexadecimal constant consists of the prefix 0x or 0X followed by a sequence of the  
 45188 decimal digits and letters 'a' (or 'A') to 'f' (or 'F') with values 10 to 15 respectively.

45189 If the value of *base* is between 2 and 36, the expected form of the subject sequence is a sequence  
 45190 of letters and digits representing an integer with the radix specified by *base*, optionally preceded  
 45191 by a '+' or '-' sign. The letters from 'a' (or 'A') to 'z' (or 'Z') inclusive are ascribed the  
 45192 values 10 to 35; only letters whose ascribed values are less than that of *base* are permitted. If the  
 45193 value of *base* is 16, the characters 0x or 0X may optionally precede the sequence of letters and  
 45194 digits, following the sign if present.

45195 The subject sequence is defined as the longest initial subsequence of the input string, starting  
 45196 with the first non-white-space character that is of the expected form. The subject sequence shall |  
 45197 contain no characters if the input string is empty or consists entirely of white-space characters, |  
 45198 or if the first non-white-space character is other than a sign or a permissible letter or digit. |

45199 If the subject sequence has the expected form and the value of *base* is 0, the sequence of |  
 45200 characters starting with the first digit shall be interpreted as an integer constant. If the subject |  
 45201 sequence has the expected form and the value of *base* is between 2 and 36, it shall be used as the |  
 45202 base for conversion, ascribing to each letter its value as given above. If the subject sequence |  
 45203 begins with a minus sign, the value resulting from the conversion shall be negated. A pointer to |  
 45204 the final string shall be stored in the object pointed to by *endptr*, provided that *endptr* is not a null |  
 45205 pointer.

45206 cx In other than the C or POSIX locales, other implementation-defined subject sequences may be  
 45207 accepted.

45208 If the subject sequence is empty or does not have the expected form, no conversion is performed;  
 45209 the value of *str* is stored in the object pointed to by *endptr*, provided that *endptr* is not a null  
 45210 pointer.

45211 CX The *strtol()* function shall not change the setting of *errno* if successful.

45212 Since 0, {LONG\_MIN} or {LLONG\_MIN}, and {LONG\_MAX} or {LLONG\_MAX} are returned on  
 45213 error and are also valid returns on success, an application wishing to check for error situations  
 45214 should set *errno* to 0, then call *strtol()* or *strtoll()*, then check *errno*.

#### 45215 RETURN VALUE

45216 Upon successful completion, these functions shall return the converted value, if any. If no  
 45217 CX conversion could be performed, 0 shall be returned and *errno* may be set to [EINVAL].

45218 If the correct value is outside the range of representable values, {LONG\_MIN}, {LONG\_MAX},  
 45219 {LLONG\_MIN} or {LLONG\_MAX} shall be returned (according to the sign of the value), and  
 45220 *errno* set to [ERANGE].

#### 45221 ERRORS

45222 These functions shall fail if:

45223 [ERANGE] The value to be returned is not representable.

45224 These functions may fail if:

45225 CX [EINVAL] The value of *base* is not supported.

#### 45226 EXAMPLES

45227 None.

#### 45228 APPLICATION USAGE

45229 None.

#### 45230 RATIONALE

45231 None.

#### 45232 FUTURE DIRECTIONS

45233 None.

#### 45234 SEE ALSO

45235 *isalpha()*, *scanf()*, *strtod()*, the Base Definitions volume of IEEE Std 1003.1-200x, <stdlib.h>

#### 45236 CHANGE HISTORY

45237 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 45238 Issue 5

45239 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.

#### 45240 Issue 6

45241 Extensions beyond the ISO C standard are now marked.

45242 The following new requirements on POSIX implementations derive from alignment with the  
 45243 Single UNIX Specification:

45244 • In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is  
 45245 added if no conversion could be performed.

45246 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

45247 • The *strtol()* prototype is updated.

45248 • The *strtoll()* function is added.

45249 **NAME**

45250            **strtold** — convert string to a double-precision number

45251 **SYNOPSIS**

45252            #include <stdlib.h>

45253            long double strtold(const char \*restrict *nptr*, char \*\*restrict *endptr*);

45254 **DESCRIPTION**

45255            Refer to *strtod*().

45256 **NAME**45257            **strtoll** — convert string to a long integer45258 **SYNOPSIS**

45259            #include &lt;stdlib.h&gt;

45260            long long strtoll(const char \*restrict *str*, char \*\*restrict *endptr*,  
45261                            int *base*);45262 **DESCRIPTION**45263            Refer to *strtol*().

## 45264 NAME

45265 strtoul, strtoull — convert string to an unsigned long

## 45266 SYNOPSIS

45267 #include &lt;stdlib.h&gt;

45268 unsigned long strtoul(const char \*restrict *str*,  
45269 char \*\*restrict *endptr*, int *base*);45270 unsigned long long strtoull(const char \*restrict *str*,  
45271 char \*\*restrict *endptr*, int *base*);

## 45272 DESCRIPTION

45273 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
45274 conflict between the requirements described here and the ISO C standard is unintentional. This  
45275 volume of IEEE Std 1003.1-200x defers to the ISO C standard.45276 These functions shall convert the initial portion of the string pointed to by *str* to a type **unsigned**  
45277 **long** and **unsigned long long** representation, respectively. First, they decompose the input  
45278 string into three parts:

- 45279 1. An initial, possibly empty, sequence of white-space characters (as specified by *isspace()*)
- 45280 2. A subject sequence interpreted as an integer represented in some radix determined by the  
45281 value of *base*
- 45282 3. A final string of one or more unrecognized characters, including the terminating null byte  
45283 of the input string

45284 Then they shall attempt to convert the subject sequence to an unsigned integer, and return the  
45285 result.45286 If the value of *base* is 0, the expected form of the subject sequence is that of a decimal constant,  
45287 octal constant, or hexadecimal constant, any of which may be preceded by a '+' or '-' sign. A  
45288 decimal constant begins with a non-zero digit, and consists of a sequence of decimal digits. An  
45289 octal constant consists of the prefix '0' optionally followed by a sequence of the digits '0' to  
45290 '7' only. A hexadecimal constant consists of the prefix 0x or 0X followed by a sequence of the  
45291 decimal digits and letters 'a' (or 'A') to 'f' (or 'F') with values 10 to 15 respectively.45292 If the value of *base* is between 2 and 36, the expected form of the subject sequence is a sequence  
45293 of letters and digits representing an integer with the radix specified by *base*, optionally preceded  
45294 by a '+' or '-' sign. The letters from 'a' (or 'A') to 'z' (or 'Z') inclusive are ascribed the  
45295 values 10 to 35; only letters whose ascribed values are less than that of *base* are permitted. If the  
45296 value of *base* is 16, the characters 0x or 0X may optionally precede the sequence of letters and  
45297 digits, following the sign if present.45298 The subject sequence is defined as the longest initial subsequence of the input string, starting  
45299 with the first non-white-space character that is of the expected form. The subject sequence shall  
45300 contain no characters if the input string is empty or consists entirely of white-space characters,  
45301 or if the first non-white-space character is other than a sign or a permissible letter or digit.45302 If the subject sequence has the expected form and the value of *base* is 0, the sequence of  
45303 characters starting with the first digit shall be interpreted as an integer constant. If the subject  
45304 sequence has the expected form and the value of *base* is between 2 and 36, it shall be used as the  
45305 base for conversion, ascribing to each letter its value as given above. If the subject sequence  
45306 begins with a minus sign, the value resulting from the conversion shall be negated. A pointer to  
45307 the final string shall be stored in the object pointed to by *endptr*, provided that *endptr* is not a null  
45308 pointer.

45309 CX In other than the C or POSIX locales, other implementation-defined subject sequences may be  
45310 accepted.

45311 If the subject sequence is empty or does not have the expected form, no conversion shall be |  
45312 performed; the value of *str* shall be stored in the object pointed to by *endptr*, provided that *endptr* |  
45313 is not a null pointer.

45314 CX The *strtoul()* function shall not change the setting of *errno* if successful.

45315 Since 0, {ULONG\_MAX}, and {ULLONG\_MAX} are returned on error and are also valid returns |  
45316 on success, an application wishing to check for error situations should set *errno* to 0, then call |  
45317 *strtoul()* or *strtoull()*, then check *errno*.

#### 45318 RETURN VALUE

45319 Upon successful completion, these functions shall return the converted value, if any. If no  
45320 CX conversion could be performed, 0 shall be returned and *errno* may be set to [EINVAL]. If the  
45321 correct value is outside the range of representable values, {ULONG\_MAX} or {ULLONG\_MAX}  
45322 shall be returned and *errno* set to [ERANGE].

#### 45323 ERRORS

45324 These functions shall fail if:

45325 CX [EINVAL] The value of *base* is not supported.

45326 [ERANGE] The value to be returned is not representable.

45327 These functions may fail if:

45328 CX [EINVAL] No conversion could be performed.

#### 45329 EXAMPLES

45330 None.

#### 45331 APPLICATION USAGE

45332 None.

#### 45333 RATIONALE

45334 None.

#### 45335 FUTURE DIRECTIONS

45336 None.

#### 45337 SEE ALSO

45338 *isalpha()*, *scanf()*, *strtod()*, *strtol()*, the Base Definitions volume of IEEE Std 1003.1-200x,  
45339 <stdlib.h>

#### 45340 CHANGE HISTORY

45341 First released in Issue 4. Derived from the ANSI C standard.

#### 45342 Issue 5

45343 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.

#### 45344 Issue 6

45345 Extensions beyond the ISO C standard are now marked.

45346 The following new requirements on POSIX implementations derive from alignment with the  
45347 Single UNIX Specification:

- 45348 • The [EINVAL] error condition is added for when the value of *base* is not supported.

45349 In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is  
45350 added if no conversion could be performed.

45351 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

45352 • The *strtoul()* prototype is updated.

45353 • The *strtoull()* function is added.

45354 **NAME**

45355            strtoumax — convert string to integer type

45356 **SYNOPSIS**

45357            #include &lt;inttypes.h&gt;

45358            uintmax\_t strtoumax(const char \*restrict *nptr*, char \*\*restrict *endptr*,  
45359                                  int *base*);45360 **DESCRIPTION**45361            Refer to *strtoimax()*.

45362 **NAME**

45363 strxfrm — string transformation

45364 **SYNOPSIS**

45365 #include &lt;string.h&gt;

45366 size\_t strxfrm(char \*restrict s1, const char \*restrict s2, size\_t n);

45367 **DESCRIPTION**

45368 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
45369 conflict between the requirements described here and the ISO C standard is unintentional. This  
45370 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

45371 The *strxfrm()* function shall transform the string pointed to by *s2* and place the resulting string  
45372 into the array pointed to by *s1*. The transformation is such that if *strcmp()* is applied to two  
45373 transformed strings, it shall return a value greater than, equal to, or less than 0, corresponding to  
45374 the result of *strcoll()* applied to the same two original strings. No more than *n* bytes are placed  
45375 into the resulting array pointed to by *s1*, including the terminating null byte. If *n* is 0, *s1*  
45376 is permitted to be a null pointer. If copying takes place between objects that overlap, the behavior  
45377 is undefined.

45378 CX The *strxfrm()* function shall not change the setting of *errno* if successful.

45379 Since no return value is reserved to indicate an error, an application wishing to check for error  
45380 situations should set *errno* to 0, then call *strxfrm()*, then check *errno*.

45381 **RETURN VALUE**

45382 Upon successful completion, *strxfrm()* shall return the length of the transformed string (not  
45383 including the terminating null byte). If the value returned is *n* or more, the contents of the array  
45384 pointed to by *s1* are unspecified.

45385 CX On error, *strxfrm()* may set *errno* but no return value is reserved to indicate an error.

45386 **ERRORS**

45387 The *strxfrm()* function may fail if:

45388 CX [EINVAL] The string pointed to by the *s2* argument contains characters outside the  
45389 domain of the collating sequence.

45390 **EXAMPLES**

45391 None.

45392 **APPLICATION USAGE**

45393 The transformation function is such that two transformed strings can be ordered by *strcmp()* as  
45394 appropriate to collating sequence information in the program's locale (category *LC\_COLLATE*).

45395 The fact that when *n* is 0 *s1* is permitted to be a null pointer is useful to determine the size of the  
45396 *s1* array prior to making the transformation.

45397 **RATIONALE**

45398 None.

45399 **FUTURE DIRECTIONS**

45400 None.

45401 **SEE ALSO**

45402 *strcmp()*, *strcoll()*, the Base Definitions volume of IEEE Std 1003.1-200x, <string.h>

45403 **CHANGE HISTORY**

45404 First released in Issue 3.

45405 Entry included for alignment with the ISO C standard.

45406 **Issue 5**45407 The DESCRIPTION is updated to indicate that *errno* does not change if the function is successful.  
4540845409 **Issue 6**

45410 Extensions beyond the ISO C standard are now marked.

45411 The following new requirements on POSIX implementations derive from alignment with the  
45412 Single UNIX Specification:

- 45413
- In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is  
45414 added if no conversion could be performed.

45415 The *strxfrm()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

45416 **NAME**

45417 swab — swap bytes

45418 **SYNOPSIS**

45419 XSI #include &lt;unistd.h&gt;

45420 void swab(const void \*restrict *src*, void \*restrict *dest*,  
45421 ssize\_t *nbytes*);

45422

45423 **DESCRIPTION**

45424 The *swab()* function shall copy *nbytes* bytes, which are pointed to by *src*, to the object pointed to  
45425 by *dest*, exchanging adjacent bytes. The *nbytes* argument should be even. If *nbytes* is odd, *swab()*  
45426 copies and exchanges *nbytes*–1 bytes and the disposition of the last byte is unspecified. If  
45427 copying takes place between objects that overlap, the behavior is undefined. If *nbytes* is  
45428 negative, *swab()* does nothing.

45429 **RETURN VALUE**

45430 None.

45431 **ERRORS**

45432 No errors are defined.

45433 **EXAMPLES**

45434 None.

45435 **APPLICATION USAGE**

45436 None.

45437 **RATIONALE**

45438 None.

45439 **FUTURE DIRECTIONS**

45440 None.

45441 **SEE ALSO**

45442 The Base Definitions volume of IEEE Std 1003.1-200x, &lt;unistd.h&gt;

45443 **CHANGE HISTORY**

45444 First released in Issue 1. Derived from Issue 1 of the SVID.

45445 **Issue 6**

45446 The **restrict** keyword is added to the *swab()* prototype for alignment with the  
45447 ISO/IEC 9899:1999 standard.

45448 **NAME**

45449 swapcontext — swap user context

45450 **SYNOPSIS**

45451 xSI #include &lt;ucontext.h&gt;

45452 int swapcontext(ucontext\_t \*restrict oucp,  
45453 const ucontext\_t \*restrict ucp);

45454

45455 **DESCRIPTION**45456 Refer to *makecontext()*.

45457 **NAME**

45458       swprintf — print formatted wide-character output

45459 **SYNOPSIS**

45460       #include <stdio.h>

45461       #include <wchar.h>

45462       int swprintf(wchar\_t \*ws, size\_t n, const wchar\_t \*format, ...);

45463 **DESCRIPTION**

45464       Refer to *fwprintf()*.

45465 **NAME**

45466 swscanf — convert formatted wide-character input

45467 **SYNOPSIS**

45468 #include &lt;stdio.h&gt;

45469 #include &lt;wchar.h&gt;

45470 int swscanf(const wchar\_t \*restrict ws, |  
45471 const wchar\_t \*restrict format, ... ); |45472 **DESCRIPTION** |45473 Refer to *fwscanf()*.

45474 **NAME**45475 `symlink` — make symbolic link to a file45476 **SYNOPSIS**45477 `#include <unistd.h>`45478 `int symlink(const char *path1, const char *path2);`45479 **DESCRIPTION**

45480 The `symlink()` function shall create a symbolic link called `path2` that contains the string pointed  
 45481 to by `path1` (`path2` is the name of the symbolic link created, `path1` is the string contained in the  
 45482 symbolic link).

45483 The string pointed to by `path1` shall be treated only as a character string and shall not be  
 45484 validated as a pathname.

45485 If the `symlink()` function fails for any reason other than [EIO], any file named by `path2` shall be  
 45486 unaffected.

45487 **RETURN VALUE**

45488 Upon successful completion, `symlink()` shall return 0; otherwise, it shall return `-1` and set `errno` to  
 45489 indicate the error.

45490 **ERRORS**45491 The `symlink()` function shall fail if:

45492 [EACCES] Write permission is denied in the directory where the symbolic link is being  
 45493 created, or search permission is denied for a component of the path prefix of  
 45494 `path2`.

45495 [EEXIST] The `path2` argument names an existing file or symbolic link.

45496 [EIO] An I/O error occurs while reading from or writing to the file system.

45497 [ELOOP] A loop exists in symbolic links encountered during resolution of the `path2`  
 45498 argument.

45499 [ENAMETOOLONG]

45500 The length of the `path2` argument exceeds {PATH\_MAX} or a pathname  
 45501 component is longer than {NAME\_MAX} or the length of the `path1` argument  
 45502 is longer than {SYMLINK\_MAX}.

45503 [ENOENT] A component of `path2` does not name an existing file or `path2` is an empty  
 45504 string.

45505 [ENOSPC] The directory in which the entry for the new symbolic link is being placed  
 45506 cannot be extended because no space is left on the file system containing the  
 45507 directory, or the new symbolic link cannot be created because no space is left  
 45508 on the file system which shall contain the link, or the file system is out of file-  
 45509 allocation resources.

45510 [ENOTDIR] A component of the path prefix of `path2` is not a directory.

45511 [EROFS] The new symbolic link would reside on a read-only file system.

45512 The `symlink()` function may fail if:

45513 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
 45514 resolution of the `path2` argument.

45515 [ENAMETOOLONG]

45516 As a result of encountering a symbolic link in resolution of the `path2`

45517 argument, the length of the substituted pathname string exceeded |  
45518 {PATH\_MAX} bytes (including the terminating null byte), or the length of the  
45519 string pointed to by *path1* exceeded {SYMLINK\_MAX}.

**45520 EXAMPLES**

45521 None.

**45522 APPLICATION USAGE**

45523 Like a hard link, a symbolic link allows a file to have multiple logical names. The presence of a  
45524 hard link guarantees the existence of a file, even after the original name has been removed. A  
45525 symbolic link provides no such assurance; in fact, the file named by the *path1* argument need not  
45526 exist when the link is created. A symbolic link can cross file system boundaries.

45527 Normal permission checks are made on each component of the symbolic link pathname during |  
45528 its resolution. |

**45529 RATIONALE**

45530 Since IEEE Std 1003.1-200x does not require any association of file times with symbolic links,  
45531 there is no requirement that file times be updated by *symlink()*.

**45532 FUTURE DIRECTIONS**

45533 None.

**45534 SEE ALSO**

45535 *lchown()*, *link()*, *lstat()*, *open()*, *readlink()*, *unlink()*, the Base Definitions volume of  
45536 IEEE Std 1003.1-200x, <**unistd.h**>

**45537 CHANGE HISTORY**

45538 First released in Issue 4, Version 2.

**45539 Issue 5**

45540 Moved from X/OPEN UNIX extension to BASE.

**45541 Issue 6**

45542 The following changes were made to align with the IEEE P1003.1a draft standard: |

- 45543
- The DESCRIPTION text is updated.
- 45544
- The [ELOOP] optional error condition is added.

45545 **NAME**

45546 sync — schedule file system updates

45547 **SYNOPSIS**45548 XSI `#include <unistd.h>`45549 `void sync(void);`

45550

45551 **DESCRIPTION**45552 The *sync()* function shall cause all information in memory that updates file systems to be  
45553 scheduled for writing out to all file systems.45554 The writing, although scheduled, is not necessarily complete upon return from *sync()*.45555 **RETURN VALUE**45556 The *sync()* function shall not return a value.45557 **ERRORS**

45558 No errors are defined.

45559 **EXAMPLES**

45560 None.

45561 **APPLICATION USAGE**

45562 None.

45563 **RATIONALE**

45564 None.

45565 **FUTURE DIRECTIONS**

45566 None.

45567 **SEE ALSO**45568 *fsync()*, the Base Definitions volume of IEEE Std 1003.1-200x, <**unistd.h**>45569 **CHANGE HISTORY**

45570 First released in Issue 4, Version 2.

45571 **Issue 5**

45572 Moved from X/OPEN UNIX extension to BASE.

45573 **NAME**

45574 sysconf — get configurable system variables

45575 **SYNOPSIS**

45576 #include &lt;unistd.h&gt;

45577 long sysconf(int name);

45578 **DESCRIPTION**

45579 The *sysconf()* function provides a method for the application to determine the current value of a  
 45580 configurable system limit or option (*variable*). Support for some system variables is dependent  
 45581 on implementation options (as indicated by the margin codes in the following table). Where an  
 45582 implementation option is not supported, the variable need not be supported.

45583 The *name* argument represents the system variable to be queried. The following table lists the  
 45584 minimal set of system variables from <limits.h> or <unistd.h> that can be returned by *sysconf()*,  
 45585 and the symbolic constants, defined in <unistd.h> that are the corresponding values used for  
 45586 *name*. Support for some configuration variables is dependent on implementation options (see  
 45587 shading and margin codes in the table below). Where an implementation option is not  
 45588 supported, the variable need not be supported.

45589

45590

45591 AIO

45592

45593

45594

45595 XSI

45596

45597

45598

45599

45600

45601

45602

45603 XSI

45604

45605

45606 XSI

45607

45608

45609

45610 TSF

45611

45612

45613

45614 MSG

45615

45616

	Variable	Value of Name
	{AIO_LISTIO_MAX}	_SC_AIO_LISTIO_MAX
	{AIO_MAX}	_SC_AIO_MAX
	{AIO_PRIO_DELTA_MAX}	_SC_AIO_PRIO_DELTA_MAX
	{ARG_MAX}	_SC_ARG_MAX
	{ATEXIT_MAX}	_SC_ATEXIT_MAX
	{BC_BASE_MAX}	_SC_BC_BASE_MAX
	{BC_DIM_MAX}	_SC_BC_DIM_MAX
	{BC_SCALE_MAX}	_SC_BC_SCALE_MAX
	{BC_STRING_MAX}	_SC_BC_STRING_MAX
	{CHILD_MAX}	_SC_CHILD_MAX
	Clock ticks/second	_SC_CLK_TCK
	{COLL_WEIGHTS_MAX}	_SC_COLL_WEIGHTS_MAX
	{DELAYTIMER_MAX}	_SC_DELAYTIMER_MAX
	{EXPR_NEST_MAX}	_SC_EXPR_NEST_MAX
	{HOST_NAME_MAX}	_SC_HOST_NAME_MAX
	{IOV_MAX}	_SC_IOV_MAX
	{LINE_MAX}	_SC_LINE_MAX
	{LOGIN_NAME_MAX}	_SC_LOGIN_NAME_MAX
	{NGROUPS_MAX}	_SC_NGROUPS_MAX
	Maximum size of <i>getgrgid_r()</i> and <i>getgrnam_r()</i> data buffers	_SC_GETGR_R_SIZE_MAX
	Maximum size of <i>getpwuid_r()</i> and <i>getpwnam_r()</i> data buffers	_SC_GETPW_R_SIZE_MAX
	{MQ_OPEN_MAX}	_SC_MQ_OPEN_MAX
	{MQ_PRIO_MAX}	_SC_MQ_PRIO_MAX
	{OPEN_MAX}	_SC_OPEN_MAX

	Variable	Value of Name
45617		
45618		
45619 ADV	_POSIX_ADVISORY_INFO	_SC_ADVISORY_INFO
45620 BAR	_POSIX_BARRIERS	_SC_BARRIERS
45621 AIO	_POSIX_ASYNCHRONOUS_IO	_SC_ASYNCHRONOUS_IO
45622	_POSIX_BASE	_SC_BASE
45623	_POSIX_C_LANG_SUPPORT	_SC_C_LANG_SUPPORT
45624	_POSIX_C_LANG_SUPPORT_R	_SC_C_LANG_SUPPORT_R
45625 CS	_POSIX_CLOCK_SELECTION	_SC_CLOCK_SELECTION
45626 CPT	_POSIX_CPUTIME	_SC_CPUTIME
45627	_POSIX_DEVICE_IO	_SC_DEVICE_IO
45628	_POSIX_DEVICE_SPECIFIC	_SC_DEVICE_SPECIFIC
45629	_POSIX_DEVICE_SPECIFIC_R	_SC_DEVICE_SPECIFIC_R
45630	_POSIX_FD_MGMT	_SC_FD_MGMT
45631	_POSIX_FIFO	_SC_FIFO
45632	_POSIX_FILE_ATTRIBUTES	_SC_FILE_ATTRIBUTES
45633	_POSIX_FILE_LOCKING	_SC_FILE_LOCKING
45634	_POSIX_FILE_SYSTEM	_SC_FILE_SYSTEM
45635 FSC	_POSIX_FSYNC	_SC_FSYNC
45636	_POSIX_JOB_CONTROL	_SC_JOB_CONTROL
45637 MF	_POSIX_MAPPED_FILES	_SC_MAPPED_FILES
45638 ML	_POSIX_MEMLOCK	_SC_MEMLOCK
45639 MLR	_POSIX_MEMLOCK_RANGE	_SC_MEMLOCK_RANGE
45640 MPR	_POSIX_MEMORY_PROTECTION	_SC_MEMORY_PROTECTION
45641 MSG	_POSIX_MESSAGE_PASSING	_SC_MESSAGE_PASSING
45642 MON	_POSIX_MONOTONIC_CLOCK	_SC_MONOTONIC_CLOCK
45643	_POSIX_MULTI_PROCESS	_SC_MULTI_PROCESS
45644	_POSIX_NETWORKING	_SC_NETWORKING
45645	_POSIX_PIPE	_SC_PIPE
45646 PIO	_POSIX_PRIORITIZED_IO	_SC_PRIORITIZED_IO
45647 PS	_POSIX_PRIORITY_SCHEDULING	_SC_PRIORITY_SCHEDULING
45648 THR	_POSIX_READER_WRITER_LOCKS	_SC_READER_WRITER_LOCKS
45649 RTS	_POSIX_REALTIME_SIGNALS	_SC_REALTIME_SIGNALS
45650	_POSIX_REGEX	_SC_REGEX
45651	_POSIX_SAVED_IDS	_SC_SAVED_IDS
45652 SEM	_POSIX_SEMAPHORES	_SC_SEMAPHORES
45653 SHM	_POSIX_SHARED_MEMORY_OBJECTS	_SC_SHARED_MEMORY_OBJECTS
45654	_POSIX_SHELL	_SC_SHELL
45655	_POSIX_SIGNALS	_SC_SIGNALS
45656	_POSIX_SINGLE_PROCESS	_SC_SINGLE_PROCESS
45657 SPN	_POSIX_SPAWN	_SC_SPAWN
45658 SPI	_POSIX_SPIN_LOCKS	_SC_SPIN_LOCKS
45659 SS	_POSIX_SPORADIC_SERVER	_SC_SPORADIC_SERVER
45660 SIO	_POSIX_SYNCHRONIZED_IO	_SC_SYNCHRONIZED_IO
45661	_POSIX_SYSTEM_DATABASE	_SC_SYSTEM_DATABASE
45662	_POSIX_SYSTEM_DATABASE_R	_SC_SYSTEM_DATABASE_R

45663

45664

45665 TSA

45666 TSS

45667 TCT

45668 TPI

45669 TPP

45670 TPS

45671 TSH

45672 TSF

45673 TSP

45674 THR

45675 TMO

45676 TMR

45677 TRC

45678 TEF

45679 TRI

45680 TRL

45681 TYM

45682

45683

45684

45685

45686

45687

45688

45689

45690

45691

45692

45693

45694

45695

45696 BE

45697

45698

45699

45700

45701

45702

45703

45704

45705 XSI

45706

	Variable	Value of Name
	_POSIX_THREAD_ATTR_STACKADDR	_SC_THREAD_ATTR_STACKADDR
	_POSIX_THREAD_ATTR_STACKSIZE	_SC_THREAD_ATTR_STACKSIZE
	_POSIX_THREAD_CPU_TIME	_SC_THREAD_CPU_TIME
	_POSIX_THREAD_PRIO_INHERIT	_SC_THREAD_PRIO_INHERIT
	_POSIX_THREAD_PRIO_PROTECT	_SC_THREAD_PRIO_PROTECT
	_POSIX_THREAD_PRIORITY_SCHEDULING	_SC_THREAD_PRIORITY_SCHEDULING
	_POSIX_THREAD_PROCESS_SHARED	_SC_THREAD_PROCESS_SHARED
	_POSIX_THREAD_SAFE_FUNCTIONS	_SC_THREAD_SAFE_FUNCTIONS
	_POSIX_THREAD_SPORADIC_SERVER	_SC_THREAD_SPORADIC_SERVER
	_POSIX_THREADS	_SC_THREADS
	_POSIX_TIMEOUTS	_SC_TIMEOUTS
	_POSIX_TIMERS	_SC_TIMERS
	_POSIX_TRACE	_SC_TRACE
	_POSIX_TRACE_EVENT_FILTER	_SC_TRACE_EVENT_FILTER
	_POSIX_TRACE_INHERIT	_SC_TRACE_INHERIT
	_POSIX_TRACE_LOG	_SC_TRACE_LOG
	_POSIX_TYPED_MEMORY_OBJECTS	_SC_TYPED_MEMORY_OBJECTS
	_POSIX_USER_GROUPS	_SC_USER_GROUPS
	_POSIX_USER_GROUPS_R	_SC_USER_GROUPS_R
	_POSIX_VERSION	_SC_VERSION
	_POSIX_V6_ILP32_OFF32	_SC_V6_ILP32_OFF32
	_POSIX_V6_ILP32_OFFBIG	_SC_V6_ILP32_OFFBIG
	_POSIX_V6_LP64_OFF64	_SC_V6_LP64_OFF64
	_POSIX_V6_LPBIG_OFFBIG	_SC_V6_LPBIG_OFFBIG
	_POSIX2_C_BIND	_SC_2_C_BIND
	_POSIX2_C_DEV	_SC_2_C_DEV
	_POSIX2_C_VERSION	_SC_2_C_VERSION
	_POSIX2_CHAR_TERM	_SC_2_CHAR_TERM
	_POSIX2_FORT_DEV	_SC_2_FORT_DEV
	_POSIX2_FORT_RUN	_SC_2_FORT_RUN
	_POSIX2_LOCALEDEF	_SC_2_LOCALEDEF
	_POSIX2_PBS	_SC_2_PBS
	_POSIX2_PBS_ACCOUNTING	_SC_2_PBS_ACCOUNTING
	_POSIX2_PBS_LOCATE	_SC_2_PBS_LOCATE
	_POSIX2_PBS_MESSAGE	_SC_2_PBS_MESSAGE
	_POSIX2_PBS_TRACK	_SC_2_PBS_TRACK
	_POSIX2_SW_DEV	_SC_2_SW_DEV
	_POSIX2_UPE	_SC_2_UPE
	_POSIX2_VERSION	_SC_2_VERSION
	_REGEX_VERSION	_SC_REGEX_VERSION
	{PAGE_SIZE}	_SC_PAGE_SIZE
	{PAGESIZE}	_SC_PAGESIZE

	Variable	Value of Name
45707		
45708		
45709 THR	{PTHREAD_DESTRUCTOR_ITERATIONS}	_SC_THREAD_DESTRUCTOR_ITERATIONS
45710	{PTHREAD_KEYS_MAX}	_SC_THREAD_KEYS_MAX
45711	{PTHREAD_STACK_MIN}	_SC_THREAD_STACK_MIN
45712	{PTHREAD_THREADS_MAX}	_SC_THREAD_THREADS_MAX
45713	{RE_DUP_MAX}	_SC_RE_DUP_MAX
45714 RTS	{RTSIG_MAX}	_SC_RTSIG_MAX
45715 SEM	{SEM_NSEMS_MAX}	_SC_SEM_NSEMS_MAX
45716	{SEM_VALUE_MAX}	_SC_SEM_VALUE_MAX
45717 RTS	{SIGQUEUE_MAX}	_SC_SIGQUEUE_MAX
45718	{STREAM_MAX}	_SC_STREAM_MAX
45719	{SYMLOOP_MAX}	_SC_SYMLOOP_MAX
45720 TMR	{TIMER_MAX}	_SC_TIMER_MAX
45721	{TTY_NAME_MAX}	_SC_TTY_NAME_MAX
45722	{TZNAME_MAX}	_SC_TZNAME_MAX
45723 XSI	_XBS5_ILP32_OFF32 (LEGACY)	_SC_XBS5_ILP32_OFF32 (LEGACY)
45724	_XBS5_ILP32_OFFBIG (LEGACY)	_SC_XBS5_ILP32_OFFBIG (LEGACY)
45725	_XBS5_LP64_OFF64 (LEGACY)	_SC_XBS5_LP64_OFF64 (LEGACY)
45726	_XBS5_LPBIG_OFFBIG (LEGACY)	_SC_XBS5_LPBIG_OFFBIG (LEGACY)
45727	_XOPEN_CRYPT	_SC_XOPEN_CRYPT
45728	_XOPEN_ENH_I18N	_SC_XOPEN_ENH_I18N
45729	_XOPEN_LEGACY	_SC_XOPEN_LEGACY
45730	_XOPEN_REALTIME	_SC_XOPEN_REALTIME
45731	_XOPEN_REALTIME_THREADS	_SC_XOPEN_REALTIME_THREADS
45732	_XOPEN_SHM	_SC_XOPEN_SHM
45733	_XOPEN_UNIX	_SC_XOPEN_UNIX
45734	_XOPEN_VERSION	_SC_XOPEN_VERSION
45735	_XOPEN_XCU_VERSION	_SC_XOPEN_XCU_VERSION

45736 **RETURN VALUE**

45737 If *name* is an invalid value, *sysconf()* shall return  $-1$  and set *errno* to indicate the error. If the  
 45738 variable corresponding to *name* has no limit, *sysconf()* shall return  $-1$  without changing the value  
 45739 of *errno*. Note that indefinite limits do not imply infinite limits; see <**limits.h**>.

45740 Otherwise, *sysconf()* shall return the current variable value on the system. The value returned  
 45741 shall not be more restrictive than the corresponding value described to the application when it  
 45742 was compiled with the implementation's <**limits.h**> or <**unistd.h**>. The value shall not change  
 45743 during the lifetime of the calling process.

45744 **ERRORS**

45745 The *sysconf()* function shall fail if:

45746 [EINVAL] The value of the *name* argument is invalid.

45747 **EXAMPLES**

45748 None.

45749 **APPLICATION USAGE**

45750 As  $-1$  is a permissible return value in a successful situation, an application wishing to check for  
 45751 error situations should set *errno* to 0, then call *sysconf()*, and, if it returns  $-1$ , check to see if *errno*  
 45752 is non-zero.

45753 If the value of *sysconf(\_SC\_2\_VERSION)* is not equal to the value of the *\_POSIX2\_VERSION*  
 45754 symbolic constant, the utilities available via *system()* or *popen()* might not behave as described in  
 45755 the Shell and Utilities volume of IEEE Std 1003.1-200x. This would mean that the application is

45756 not running in an environment that conforms to the Shell and Utilities volume of  
45757 IEEE Std 1003.1-200x. Some applications might be able to deal with this, others might not.  
45758 However, the functions defined in this volume of IEEE Std 1003.1-200x continue to operate as  
45759 specified, even if: `sysconf(_SC_2_VERSION)` reports that the utilities no longer perform as  
45760 specified.

#### 45761 RATIONALE

45762 This functionality was added in response to requirements of application developers and of  
45763 system vendors who deal with many international system configurations. It is closely related to  
45764 `pathconf()` and `fpathconf()`.

45765 Although a conforming application can run on all systems by never demanding more resources  
45766 than the minimum values published in this volume of IEEE Std 1003.1-200x, it is useful for that  
45767 application to be able to use the actual value for the quantity of a resource available on any  
45768 given system. To do this, the application makes use of the value of a symbolic constant in  
45769 `<limits.h>` or `<unistd.h>`.

45770 However, once compiled, the application must still be able to cope if the amount of resource  
45771 available is increased. To that end, an application may need a means of determining the quantity  
45772 of a resource, or the presence of an option, at execution time.

45773 Two examples are offered:

- 45774 1. Applications may wish to act differently on systems with or without job control.  
45775 Applications vendors who wish to distribute only a single binary package to all instances  
45776 of a computer architecture would be forced to assume job control is never available if it  
45777 were to rely solely on the `<unistd.h>` value published in this volume of  
45778 IEEE Std 1003.1-200x.
- 45779 2. International applications vendors occasionally require knowledge of the number of clock  
45780 ticks per second. Without these facilities, they would be required to either distribute their  
45781 applications partially in source form or to have 50Hz and 60Hz versions for the various  
45782 countries in which they operate.

45783 It is the knowledge that many applications are actually distributed widely in executable form  
45784 that leads to this facility. If limited to the most restrictive values in the headers, such  
45785 applications would have to be prepared to accept the most limited environments offered by the  
45786 smallest microcomputers. Although this is entirely portable, there was a consensus that they  
45787 should be able to take advantage of the facilities offered by large systems, without the  
45788 restrictions associated with source and object distributions.

45789 During the discussions of this feature, it was pointed out that it is almost always possible for an  
45790 application to discern what a value might be at runtime by suitably testing the various functions  
45791 themselves. And, in any event, it could always be written to adequately deal with error returns  
45792 from the various functions. In the end, it was felt that this imposed an unreasonable level of  
45793 complication and sophistication on the application writer.

45794 This runtime facility is not meant to provide ever-changing values that applications have to  
45795 check multiple times. The values are seen as changing no more frequently than once per system  
45796 initialization, such as by a system administrator or operator with an automatic configuration  
45797 program. This volume of IEEE Std 1003.1-200x specifies that they shall not change within the  
45798 lifetime of the process.

45799 Some values apply to the system overall and others vary at the file system or directory level. The  
45800 latter are described in `pathconf()`.

45801 Note that all values returned must be expressible as integers. String values were considered, but  
45802 the additional flexibility of this approach was rejected due to its added complexity of

- 45803 implementation and use.
- 45804 Some values, such as {PATH\_MAX}, are sometimes so large that they must not be used to, say,  
45805 allocate arrays. The *sysconf()* function returns a negative value to show that this symbolic  
45806 constant is not even defined in this case.
- 45807 Similar to *pathconf()*, this permits the implementation not to have a limit. When one resource is  
45808 infinite, returning an error indicating that some other resource limit has been reached is  
45809 conforming behavior.
- 45810 **FUTURE DIRECTIONS**
- 45811 None.
- 45812 **SEE ALSO**
- 45813 *confstr()*, *pathconf()*, the Base Definitions volume of IEEE Std 1003.1-200x, <**limits.h**>,  
45814 <**unistd.h**>, the Shell and Utilities volume of IEEE Std 1003.1-200x, *getconf*
- 45815 **CHANGE HISTORY**
- 45816 First released in Issue 3.
- 45817 Entry included for alignment with the POSIX.1-1988 standard.
- 45818 **Issue 5**
- 45819 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX  
45820 Threads Extension.
- 45821 The *\_XBS\_* variables and name values are added to the table of system variables in the  
45822 DESCRIPTION. These are all marked EX.
- 45823 **Issue 6**
- 45824 The symbol CLK\_TCK is obsolescent and removed. It is replaced with the phrase “clock ticks  
45825 per second”.
- 45826 The symbol {PASS\_MAX} is removed.
- 45827 The following changes were made to align with the IEEE P1003.1a draft standard:
- 45828 • Table entries added for the following variables: *\_SC\_REGEX*, *\_SC\_SHELL*,  
45829 *\_SC\_REGEX\_VERSION*, *\_SC\_SYMLoop\_MAX*.
- 45830 The following *sysconf()* variables and their associated names are added for alignment with  
45831 IEEE Std 1003.1d-1999:
- 45832 *\_POSIX\_ADVISORY\_INFO*  
45833 *\_POSIX\_CPUTIME*  
45834 *\_POSIX\_SPAWN*  
45835 *\_POSIX\_SPORADIC\_SERVER*  
45836 *\_POSIX\_THREAD\_CPUTIME*  
45837 *\_POSIX\_THREAD\_SPORADIC\_SERVER*  
45838 *\_POSIX\_TIMEOUTS*
- 45839 The following changes are made to the DESCRIPTION for alignment with IEEE Std 1003.1j-2000:
- 45840 • A statement expressing the dependency of support for some system variables on  
45841 implementation options is added.
- 45842 • The following system variables are added:

45843            \_POSIX\_BARRIERS  
45844            \_POSIX\_CLOCK\_SELECTION  
45845            \_POSIX\_MONOTONIC\_CLOCK  
45846            \_POSIX\_READER\_WRITER\_LOCKS  
45847            \_POSIX\_SPIN\_LOCKS  
45848            \_POSIX\_TYPED\_MEMORY\_OBJECTS

45849            The following system variables are added for alignment with IEEE Std 1003.2d-1994:

45850            \_POSIX2\_PBS  
45851            \_POSIX2\_PBS\_ACCOUNTING  
45852            \_POSIX2\_PBS\_LOCATE  
45853            \_POSIX2\_PBS\_MESSAGE  
45854            \_POSIX2\_PBS\_TRACK

45855            The following *sysconf()* variables and their associated names are added for alignment with  
45856            IEEE Std 1003.1q-2000:

45857            \_POSIX\_TRACE  
45858            \_POSIX\_TRACE\_EVENT\_FILTER  
45859            \_POSIX\_TRACE\_INHERIT  
45860            \_POSIX\_TRACE\_LOG

45861            The macros associated with the *c89* programming models are marked LEGACY, and new  
45862            equivalent macros associated with *c99* are introduced.

45863 **NAME**

45864            syslog — log a message

45865 **SYNOPSIS**

45866 XSI        #include <syslog.h>

45867            void syslog(int *priority*, const char \**message*, ... /\* *argument* \*/);

45868

45869 **DESCRIPTION**

45870            Refer to *closelog*().

45871 **NAME**

45872           system — issue a command

45873 **SYNOPSIS**

45874           #include &lt;stdlib.h&gt;

45875           int system(const char \**command*);45876 **DESCRIPTION**

45877 CX       The functionality described on this reference page is aligned with the ISO C standard. Any  
 45878           conflict between the requirements described here and the ISO C standard is unintentional. This  
 45879           volume of IEEE Std 1003.1-200x defers to the ISO C standard.

45880           If *command* is a null pointer, the *system()* function shall determine whether the host environment  
 45881           has a command processor. If *command* is not a null pointer, the *system()* function shall pass the  
 45882           string pointed to by *command* to that command processor to be executed in an implementation-  
 45883           defined manner; this might then cause the program calling *system()* to behave in a non-  
 45884           conforming manner or to terminate.

45885 CX       The environment of the executed command shall be as if a child process were created using  
 45886           *fork()*, and the child process invoked the *sh* utility using *execl()* as follows:

```
45887           execl(<shell path>, "sh", "-c", command, (char *)0);
```

45888           where <shell path> is an unspecified pathname for the *sh* utility.

45889           The *system()* function shall ignore the SIGINT and SIGQUIT signals, and shall block the  
 45890           SIGCHLD signal, while waiting for the command to terminate. If this might cause the  
 45891           application to miss a signal that would have killed it, then the application should examine the  
 45892           return value from *system()* and take whatever action is appropriate to the application if the  
 45893           command terminated due to receipt of a signal.

45894           The *system()* function shall not affect the termination status of any child of the calling processes  
 45895           other than the process or processes it itself creates.

45896           The *system()* function shall not return until the child process has terminated.

45897 **RETURN VALUE**

45898           If *command* is a null pointer, *system()* shall return non-zero to indicate that a command processor  
 45899 CX           is available, or zero if none is available. The *system()* function shall always return non-zero when  
 45900           *command* is NULL.

45901 CX       If *command* is not a null pointer, *system()* shall return the termination status of the command  
 45902           language interpreter in the format specified by *waitpid()*. The termination status shall be as  
 45903           defined for the *sh* utility; otherwise, the termination status is unspecified. If some error prevents  
 45904           the command language interpreter from executing after the child process is created, the return  
 45905           value from *system()* shall be as if the command language interpreter had terminated using  
 45906           *exit(127)* or *\_exit(127)*. If a child process cannot be created, or if the termination status for the  
 45907           command language interpreter cannot be obtained, *system()* shall return -1 and set *errno* to  
 45908           indicate the error.

45909 **ERRORS**

45910 CX       The *system()* function may set *errno* values as described by *fork()*.

45911           In addition, *system()* may fail if:

45912 CX       [ECHILD]       The status of the child process created by *system()* is no longer available.

45913 **EXAMPLES**

45914 None.

45915 **APPLICATION USAGE**

45916 If the return value of *system()* is not `-1`, its value can be decoded through the use of the macros  
45917 described in `<sys/wait.h>`. For convenience, these macros are also provided in `<stdlib.h>`.

45918 Note that, while *system()* must ignore `SIGINT` and `SIGQUIT` and block `SIGCHLD` while waiting  
45919 for the child to terminate, the handling of signals in the executed command is as specified by  
45920 *fork()* and *exec*. For example, if `SIGINT` is being caught or is set to `SIG_DFL` when *system()* is  
45921 called, then the child is started with `SIGINT` handling set to `SIG_DFL`.

45922 Ignoring `SIGINT` and `SIGQUIT` in the parent process prevents coordination problems (two  
45923 processes reading from the same terminal, for example) when the executed command ignores or  
45924 catches one of the signals. It is also usually the correct action when the user has given a  
45925 command to the application to be executed synchronously (as in the `'!'` command in many  
45926 interactive applications). In either case, the signal should be delivered only to the child process,  
45927 not to the application itself. There is one situation where ignoring the signals might have less  
45928 than the desired effect. This is when the application uses *system()* to perform some task invisible  
45929 to the user. If the user typed the interrupt character ("`^C`", for example) while *system()* is being  
45930 used in this way, one would expect the application to be killed, but only the executed command  
45931 is killed. Applications that use *system()* in this way should carefully check the return status from  
45932 *system()* to see if the executed command was successful, and should take appropriate action  
45933 when the command fails.

45934 Blocking `SIGCHLD` while waiting for the child to terminate prevents the application from  
45935 catching the signal and obtaining status from *system()*'s child process before *system()* can get the  
45936 status itself.

45937 The context in which the utility is ultimately executed may differ from that in which *system()*  
45938 was called. For example, file descriptors that have the `FD_CLOEXEC` flag set are closed, and the  
45939 process ID and parent process ID are different. Also, if the executed utility changes its  
45940 environment variables or its current working directory, that change is not reflected in the caller's  
45941 context.

45942 There is no defined way for an application to find the specific path for the shell. However,  
45943 *confstr()* can provide a value for *PATH* that is guaranteed to find the *sh* utility.

45944 **RATIONALE**

45945 The *system()* function should not be used by programs that have set user (or group) ID  
45946 privileges. The *fork()* and *exec* family of functions (except *execlp()* and *execvp()*), should be used  
45947 instead. This prevents any unforeseen manipulation of the environment of the user that could  
45948 cause execution of commands not anticipated by the calling program.

45949 There are three levels of specification for the *system()* function. The ISO C standard gives the  
45950 most basic. It requires that the function exists, and defines a way for an application to query  
45951 whether a command language interpreter exists. It says nothing about the command language or  
45952 the environment in which the command is interpreted.

45953 IEEE Std 1003.1-200x places additional restrictions on *system()*. It requires that if there is a  
45954 command language interpreter, the environment must be as specified by *fork()* and *exec*. This  
45955 ensures, for example, that close-on-exec works, that file locks are not inherited, and that the  
45956 process ID is different. It also specifies the return value from *system()* when the command line  
45957 can be run, thus giving the application some information about the command's completion  
45958 statu.

45959 Finally, IEEE Std 1003.1-200x requires the command to be interpreted as in the shell command  
45960 language defined in the Shell and Utilities volume of IEEE Std 1003.1-200x.

45961 Note that, *system*(NULL) is required to return non-zero, indicating that there is a command  
45962 language interpreter. At first glance, this would seem to conflict with the ISO C standard which  
45963 allows *system*(NULL) to return zero. There is no conflict, however. A system must have a  
45964 command language interpreter, and is non-conforming if none is present. It is therefore  
45965 permissible for the *system*() function on such a system to implement the behavior specified by  
45966 the ISO C standard as long as it is understood that the implementation does not conform to  
45967 IEEE Std 1003.1-200x if *system*(NULL) returns zero.

45968 It was explicitly decided that when *command* is NULL, *system*() should not be required to check  
45969 to make sure that the command language interpreter actually exists with the correct mode, that  
45970 there are enough processes to execute it, and so on. The call *system*(NULL) could, theoretically,  
45971 check for such problems as too many existing child processes, and return zero. However, it  
45972 would be inappropriate to return zero due to such a (presumably) transient condition. If some  
45973 condition exists that is not under the control of this application and that would cause any  
45974 *system*() call to fail, that system has been rendered non-conforming.

45975 Early drafts required, or allowed, *system*() to return with *errno* set to [EINTR] if it was  
45976 interrupted with a signal. This error return was removed, and a requirement that *system*() not  
45977 return until the child has terminated was added. This means that if a *waitpid*() call in *system*()  
45978 exits with *errno* set to [EINTR], *system*() must re-issue the *waitpid*(). This change was made for  
45979 two reasons:

- 45980 1. There is no way for an application to clean up if *system*() returns [EINTR], short of calling  
45981 *wait*(), and that could have the undesirable effect of returning the status of children other  
45982 than the one started by *system*().
- 45983 2. While it might require a change in some historical implementations, those  
45984 implementations already have to be changed because they use *wait*() instead of *waitpid*().

45985 Note that if the application is catching SIGCHLD signals, it will receive such a signal before a  
45986 successful *system*() call returns.

45987 To conform to IEEE Std 1003.1-200x, *system*() must use *waitpid*(), or some similar function,  
45988 instead of *wait*().

45989 The following code sample illustrates how *system*() might be implemented on an  
45990 implementation conforming to IEEE Std 1003.1-200x.

```
45991 #include <signal.h>
45992 int system(const char *cmd)
45993 {
45994     int stat;
45995     pid_t pid;
45996     struct sigaction sa, savintr, savequit;
45997     sigset_t saveblock;
45998     if (cmd == NULL)
45999         return(1);
46000     sa.sa_handler = SIG_IGN;
46001     sigemptyset(&sa.sa_mask);
46002     sa.sa_flags = 0;
46003     sigemptyset(&savintr.sa_mask);
46004     sigemptyset(&savequit.sa_mask);
46005     sigaction(SIGINT, &sa, &savintr);
46006     sigaction(SIGQUIT, &sa, &savequit);
```

```

46007     sigaddset(&sa.sa_mask, SIGCHLD);
46008     sigprocmask(SIG_BLOCK, &sa.sa_mask, &saveblock);
46009     if ((pid = fork()) == 0) {
46010         sigaction(SIGINT, &saveintr, (struct sigaction *)0);
46011         sigaction(SIGQUIT, &savequit, (struct sigaction *)0);
46012         sigprocmask(SIG_SETMASK, &saveblock, (sigset_t *)0);
46013         execl("/bin/sh", "sh", "-c", cmd, (char *)0);
46014         _exit(127);
46015     }
46016     if (pid == -1) {
46017         stat = -1; /* errno comes from fork() */
46018     } else {
46019         while (waitpid(pid, &stat, 0) == -1) {
46020             if (errno != EINTR){
46021                 stat = -1;
46022                 break;
46023             }
46024         }
46025     }
46026     sigaction(SIGINT, &saveintr, (struct sigaction *)0);
46027     sigaction(SIGQUIT, &savequit, (struct sigaction *)0);
46028     sigprocmask(SIG_SETMASK, &saveblock, (sigset_t *)0);
46029     return(stat);
46030 }

```

46031 Note that, while a particular implementation of *system()* (such as the one above) can assume a  
46032 particular path for the shell, such a path is not necessarily valid on another system. The above  
46033 example is not portable, and is not intended to be.

46034 One reviewer suggested that an implementation of *system()* might want to use an environment  
46035 variable such as *SHELL* to determine which command interpreter to use. The supposed  
46036 implementation would use the default command interpreter if the one specified by the  
46037 environment variable was not available. This would allow a user, when using an application  
46038 that prompts for command lines to be processed using *system()*, to specify a different command  
46039 interpreter. Such an implementation is discouraged. If the alternate command interpreter did not  
46040 follow the command line syntax specified in the Shell and Utilities volume of  
46041 IEEE Std 1003.1-200x, then changing *SHELL* would render *system()* non-conforming. This would  
46042 affect applications that expected the specified behavior from *system()*, and since the Shell and  
46043 Utilities volume of IEEE Std 1003.1-200x does not mention that *SHELL* affects *system()*, the  
46044 application would not know that it needed to unset *SHELL*.

#### 46045 FUTURE DIRECTIONS

46046 None.

#### 46047 SEE ALSO

46048 *exec*, *pipe()*, *waitpid()*, the Base Definitions volume of IEEE Std 1003.1-200x, <**limits.h**>,  
46049 <**signal.h**>, <**stdlib.h**>, <**sys/wait.h**>, the Shell and Utilities volume of IEEE Std 1003.1-200x, *sh* |

#### 46050 CHANGE HISTORY

46051 First released in Issue 1. Derived from Issue 1 of the SVID. |

46052 **Issue 6**

46053

The following changes were made to align with the IEEE P1003.1a draft standard:

46054

- The DESCRIPTION is adjusted to reflect the behavior on systems that do not support the Shell option.

46055

46056 **NAME**

46057 tan, tanf, tanl — tangent function

46058 **SYNOPSIS**

46059 #include &lt;math.h&gt;

46060 double tan(double x);

46061 float tanf(float x);

46062 long double tanl(long double x);

46063 **DESCRIPTION**

46064 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 46065 conflict between the requirements described here and the ISO C standard is unintentional. This  
 46066 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

46067 These functions shall compute the tangent of their argument *x*, measured in radians.

46068 An application wishing to check for error situations should set *errno* to zero and call  
 46069 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 46070 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 46071 zero, an error has occurred.

46072 **RETURN VALUE**46073 Upon successful completion, these functions shall return the tangent of *x*.

46074 If the correct value would cause underflow, and is not representable, a range error may occur,  
 46075 MX and either 0.0 (if supported), or an implementation-defined value shall be returned.

46076 MX If *x* is NaN, a NaN shall be returned.46077 If *x* is  $\pm 0$ , *x* shall be returned.46078 If *x* is subnormal, a range error may occur and *x* should be returned.

46079 If *x* is  $\pm\text{Inf}$ , a domain error shall occur, and either a NaN (if supported), or an implementation-  
 46080 defined value shall be returned.

46081 If the correct value would cause underflow, and is representable, a range error may occur and  
 46082 the correct value shall be returned.

46083 XSI If the correct value would cause overflow, a range error shall occur and *tan()*, *tanf()*, and *tanl()*  
 46084 shall return the value of the macro HUGE\_VAL, HUGE\_VALF, and HUGE\_VALL, respectively.

46085 **ERRORS**

46086 These functions shall fail if:

46087 MX **Domain Error** The value *x* is  $\pm\text{Inf}$ .

46088 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero, |  
 46089 then *errno* shall be set to [EDOM]. If the integer expression (math\_errhandling |  
 46090 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception |  
 46091 shall be raised. |

46092 XSI **Range Error** The result overflows

46093 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero, |  
 46094 then *errno* shall be set to [ERANGE]. If the integer expression |  
 46095 (math\_errhandling & MATH\_ERREXCEPT) is non-zero, then the overflow |  
 46096 floating-point exception shall be raised. |

46097 These functions may fail if:

46098 MX      Range Error      The result underflows, or the value  $x$  is subnormal.

46099                              If the integer expression (`math_errhandling` & `MATH_ERRNO`) is non-zero, |  
 46100                              then *errno* shall be set to `[ERANGE]`. If the integer expression |  
 46101                              (`math_errhandling` & `MATH_ERREXCEPT`) is non-zero, then the underflow |  
 46102                              floating-point exception shall be raised. |

#### 46103 EXAMPLES

##### 46104              Taking the Tangent of a 45-Degree Angle

```
46105              #include <math.h>
46106              ...
46107              double radians = 45.0 * M_PI / 180;
46108              double result;
46109              ...
46110              result = tan (radians);
```

#### 46111 APPLICATION USAGE

46112              There are no known floating-point representations such that for a normal argument,  $\tan(x)$  is  
 46113              either overflow or underflow.

46114              These functions may lose accuracy when their argument is near a multiple of  $\pi/2$  or is far from  
 46115              0.0.

46116              On error, the expressions (`math_errhandling` & `MATH_ERRNO`) and (`math_errhandling` &  
 46117              `MATH_ERREXCEPT`) are independent of each other, but at least one of them must be non-zero.

#### 46118 RATIONALE

46119              None.

#### 46120 FUTURE DIRECTIONS

46121              None.

#### 46122 SEE ALSO

46123              *atan()*, *feclearexcept()*, *fetestexcept()*, *isnan()*, the Base Definitions volume of IEEE Std 1003.1-200x, |  
 46124              Section 4.18, Treatment of Error Conditions for Mathematical Functions, <**math.h**> |

#### 46125 CHANGE HISTORY

46126              First released in Issue 1. Derived from Issue 1 of the SVID.

#### 46127 Issue 5

46128              The last two paragraphs of the DESCRIPTION were included as APPLICATION USAGE notes  
 46129              in previous issues.

#### 46130 Issue 6

46131              The *tanf()* and *tanl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

46132              The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
 46133              revised to align with the ISO/IEC 9899:1999 standard.

46134              IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
 46135              marked.

46136 **NAME**

46137           tanf — tangent function

46138 **SYNOPSIS**

46139           #include &lt;math.h&gt;

46140           float tanf(float x);

46141 **DESCRIPTION**46142           Refer to *tan()*.

46143 **NAME**

46144 tanh, tanhf, tanhl — hyperbolic tangent functions

46145 **SYNOPSIS**

46146 #include &lt;math.h&gt;

46147 double tanh(double x);

46148 float tanhf(float x);

46149 long double tanhl(long double x);

46150 **DESCRIPTION**

46151 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 46152 conflict between the requirements described here and the ISO C standard is unintentional. This  
 46153 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

46154 These functions shall compute the hyperbolic tangent of their argument *x*.

46155 An application wishing to check for error situations should set *errno* to zero and call  
 46156 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 46157 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 46158 zero, an error has occurred.

46159 **RETURN VALUE**46160 Upon successful completion, these functions shall return the hyperbolic tangent of *x*.46161 **MX** If *x* is NaN, a NaN shall be returned.46162 If *x* is  $\pm 0$ , *x* shall be returned.46163 If *x* is  $\pm\text{Inf}$ ,  $\pm 1$  shall be returned.46164 If *x* is subnormal, a range error may occur and *x* should be returned.46165 **ERRORS**

46166 These functions may fail if:

46167 **MX** **Range Error** The value of *x* is subnormal.

46168 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero, |  
 46169 then *errno* shall be set to [ERANGE]. If the integer expression |  
 46170 (math\_errhandling & MATH\_ERREXCEPT) is non-zero, then the underflow |  
 46171 floating-point exception shall be raised. |

46172 **EXAMPLES**

46173 None.

46174 **APPLICATION USAGE**

46175 On error, the expressions (math\_errhandling & MATH\_ERRNO) and (math\_errhandling &  
 46176 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

46177 **RATIONALE**

46178 None.

46179 **FUTURE DIRECTIONS**

46180 None.

46181 **SEE ALSO**

46182 *atanh()*, *feclearexcept()*, *fetestexcept()*, *isnan()*, *tan()*, the Base Definitions volume of |  
 46183 IEEE Std 1003.1-200x, Section 4.18, Treatment of Error Conditions for Mathematical Functions, |  
 46184 <math.h>

46185 **CHANGE HISTORY**

46186 First released in Issue 1. Derived from Issue 1 of the SVID.

46187 **Issue 5**

46188 The DESCRIPTION is updated to indicate how an application should check for an error. This  
46189 text was previously published in the APPLICATION USAGE section.

46190 **Issue 6**

46191 The *tanhf()* and *tanhl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

46192 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
46193 revised to align with the ISO/IEC 9899:1999 standard.

46194 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
46195 marked.

46196 **NAME**

46197           tanl — tangent function

46198 **SYNOPSIS**

46199           #include <math.h>

46200           long double tanl(long double x);

46201 **DESCRIPTION**

46202           Refer to *tan()*.

46203 **NAME**

46204 tcdrain — wait for transmission of output

46205 **SYNOPSIS**

46206 #include &lt;termios.h&gt;

46207 int tcdrain(int *fildev*);46208 **DESCRIPTION**46209 The *tcdrain()* function shall block until all output written to the object referred to by *fildev* is |  
46210 transmitted. The *fildev* argument is an open file descriptor associated with a terminal.46211 Any attempts to use *tcdrain()* from a process which is a member of a background process group |  
46212 on a *fildev* associated with its controlling terminal, shall cause the process group to be sent a |  
46213 SIGTTOU signal. If the calling process is blocking or ignoring SIGTTOU signals, the process |  
46214 shall be allowed to perform the operation, and no signal is sent.46215 **RETURN VALUE**46216 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to |  
46217 indicate the error.46218 **ERRORS**46219 The *tcdrain()* function shall fail if:46220 [EBADF] The *fildev* argument is not a valid file descriptor.46221 [EINTR] A signal interrupted *tcdrain()*.46222 [ENOTTY] The file associated with *fildev* is not a terminal.46223 The *tcdrain()* function may fail if:46224 [EIO] The process group of the writing process is orphaned, and the writing process |  
46225 is not ignoring or blocking SIGTTOU.46226 **EXAMPLES**

46227 None.

46228 **APPLICATION USAGE**

46229 None.

46230 **RATIONALE**

46231 None.

46232 **FUTURE DIRECTIONS**

46233 None.

46234 **SEE ALSO**46235 *tcfldsh()*, the Base Definitions volume of IEEE Std 1003.1-200x, <termios.h>, <unistd.h>, the Base |  
46236 Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface46237 **CHANGE HISTORY**

46238 First released in Issue 3.

46239 Entry included for alignment with the POSIX.1-1988 standard.

46240 **Issue 6**46241 The following new requirements on POSIX implementations derive from alignment with the |  
46242 Single UNIX Specification:

- 46243 • In the DESCRIPTION, the final paragraph is no longer conditional on |
- 
- 46244 \_POSIX\_JOB\_CONTROL. This is a FIPS requirement.

46245

- The [EIO] error is added.

46246 **NAME**

46247 tcflow — suspend or restart the transmission or reception of data

46248 **SYNOPSIS**

46249 #include &lt;termios.h&gt;

46250 int tcflow(int *fildev*, int *action*);46251 **DESCRIPTION**

46252 The *tcflow()* function shall suspend or restart transmission or reception of data on the object  
 46253 referred to by *fildev*, depending on the value of *action*. The *fildev* argument is an open file  
 46254 descriptor associated with a terminal.

- 46255 • If *action* is TCOFF, output shall be suspended.
- 46256 • If *action* is TCOON, suspended output shall be restarted.
- 46257 • If *action* is TCIOFF, the system shall transmit a STOP character, which is intended to cause  
 46258 the terminal device to stop transmitting data to the system.
- 46259 • If *action* is TCION, the system shall transmit a START character, which is intended to cause  
 46260 the terminal device to start transmitting data to the system.

46261 The default on the opening of a terminal file is that neither its input nor its output are  
 46262 suspended.

46263 Attempts to use *tcflow()* from a process which is a member of a background process group on a  
 46264 *fildev* associated with its controlling terminal, shall cause the process group to be sent a  
 46265 SIGTTOU signal. If the calling process is blocking or ignoring SIGTTOU signals, the process  
 46266 shall be allowed to perform the operation, and no signal is sent.

46267 **RETURN VALUE**

46268 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to  
 46269 indicate the error.

46270 **ERRORS**46271 The *tcflow()* function shall fail if:

- 46272 [EBADF] The *fildev* argument is not a valid file descriptor.
- 46273 [EINVAL] The *action* argument is not a supported value.
- 46274 [ENOTTY] The file associated with *fildev* is not a terminal.

46275 The *tcflow()* function may fail if:

- 46276 [EIO] The process group of the writing process is orphaned, and the writing process  
 46277 is not ignoring or blocking SIGTTOU.

46278 **EXAMPLES**

46279 None.

46280 **APPLICATION USAGE**

46281 None.

46282 **RATIONALE**

46283 None.

46284 **FUTURE DIRECTIONS**

46285 None.

46286 **SEE ALSO**

46287 *tcsendbreak()*, the Base Definitions volume of IEEE Std 1003.1-200x, <termios.h>, <unistd.h>, the  
46288 Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface

46289 **CHANGE HISTORY**

46290 First released in Issue 3.

46291 Entry included for alignment with the POSIX.1-1988 standard.

46292 **Issue 6**

46293 The following new requirements on POSIX implementations derive from alignment with the  
46294 Single UNIX Specification:

- 46295 • The [EIO] error is added.

46296 **NAME**

46297 tcflush — flush non-transmitted output data, non-read input data, or both

46298 **SYNOPSIS**

46299 #include &lt;termios.h&gt;

46300 int tcflush(int *fildev*, int *queue\_selector*);46301 **DESCRIPTION**46302 Upon successful completion, *tcflush()* shall discard data written to the object referred to by *fildev*  
46303 (an open file descriptor associated with a terminal) but not transmitted, or data received but not  
46304 read, depending on the value of *queue\_selector*:

- 46305 • If *queue\_selector* is TCIFLUSH, it shall flush data received but not read.
- 46306 • If *queue\_selector* is TCOFLUSH, it shall flush data written but not transmitted.
- 46307 • If *queue\_selector* is TCIOFLUSH, it shall flush both data received but not read and data  
46308 written but not transmitted.

46309 Attempts to use *tcflush()* from a process which is a member of a background process group on a  
46310 *fildev* associated with its controlling terminal shall cause the process group to be sent a SIGTTOU  
46311 signal. If the calling process is blocking or ignoring SIGTTOU signals, the process shall be  
46312 allowed to perform the operation, and no signal is sent.

46313 **RETURN VALUE**

46314 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to  
46315 indicate the error.

46316 **ERRORS**46317 The *tcflush()* function shall fail if:

- 46318 [EBADF] The *fildev* argument is not a valid file descriptor.
- 46319 [EINVAL] The *queue\_selector* argument is not a supported value.
- 46320 [ENOTTY] The file associated with *fildev* is not a terminal.

46321 The *tcflush()* function may fail if:

- 46322 [EIO] The process group of the writing process is orphaned, and the writing process  
46323 is not ignoring or blocking SIGTTOU.

46324 **EXAMPLES**

46325 None.

46326 **APPLICATION USAGE**

46327 None.

46328 **RATIONALE**

46329 None.

46330 **FUTURE DIRECTIONS**

46331 None.

46332 **SEE ALSO**

46333 *tcdrain()*, the Base Definitions volume of IEEE Std 1003.1-200x, <termios.h>, <unistd.h>, the  
46334 Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface

46335 **CHANGE HISTORY**

46336 First released in Issue 3.

46337 Entry included for alignment with the POSIX.1-1988 standard.

46338 **Issue 6**46339 The Open Group Corrigendum U035/1 is applied. In the ERRORS and APPLICATION USAGE  
46340 sections, references to *tcfow*() are replaced with *tcflush*().46341 The following new requirements on POSIX implementations derive from alignment with the  
46342 Single UNIX Specification:46343 • In the DESCRIPTION, the final paragraph is no longer conditional on  
46344 `_POSIX_JOB_CONTROL`. This is a FIPS requirement.

46345 • The [EIO] error is added.

46346 **NAME**

46347 tcgetattr — get the parameters associated with the terminal

46348 **SYNOPSIS**

46349 #include <termios.h>

46350 int tcgetattr(int *fildev*, struct termios \**termios\_p*);

46351 **DESCRIPTION**

46352 The *tcgetattr()* function shall get the parameters associated with the terminal referred to by *fildev*  
46353 and store them in the **termios** structure referenced by *termios\_p*. The *fildev* argument is an open  
46354 file descriptor associated with a terminal.

46355 The *termios\_p* argument is a pointer to a **termios** structure.

46356 The *tcgetattr()* operation is allowed from any process.

46357 If the terminal device supports different input and output baud rates, the baud rates stored in  
46358 the **termios** structure returned by *tcgetattr()* shall reflect the actual baud rates, even if they are  
46359 equal. If differing baud rates are not supported, the rate returned as the output baud rate shall be  
46360 the actual baud rate. If the terminal device does not support split baud rates, the input baud rate  
46361 stored in the **termios** structure shall be the output rate (as one of the symbolic values).

46362 **RETURN VALUE**

46363 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to  
46364 indicate the error.

46365 **ERRORS**

46366 The *tcgetattr()* function shall fail if:

46367 [EBADF] The *fildev* argument is not a valid file descriptor.

46368 [ENOTTY] The file associated with *fildev* is not a terminal.

46369 **EXAMPLES**

46370 None.

46371 **APPLICATION USAGE**

46372 None.

46373 **RATIONALE**

46374 Care must be taken when changing the terminal attributes. Applications should always do a  
46375 *tcgetattr()*, save the **termios** structure values returned, and then do a *tcsetattr()* changing only  
46376 the necessary fields. The application should use the values saved from the *tcgetattr()* to reset the  
46377 terminal state whenever it is done with the terminal. This is necessary because terminal  
46378 attributes apply to the underlying port and not to each individual open instance; that is, all  
46379 processes that have used the terminal see the latest attribute changes.

46380 A program that uses these functions should be written to catch all signals and take other  
46381 appropriate actions to ensure that when the program terminates, whether planned or not, the  
46382 terminal device's state is restored to its original state.

46383 Existing practice dealing with error returns when only part of a request can be honored is based  
46384 on calls to the *ioctl()* function. In historical BSD and System V implementations, the  
46385 corresponding *ioctl()* returns zero if the requested actions were semantically correct, even if  
46386 some of the requested changes could not be made. Many existing applications assume this  
46387 behavior and would no longer work correctly if the return value were changed from zero to -1  
46388 in this case.

46389 Note that either specification has a problem. When zero is returned, it implies everything  
46390 succeeded even if some of the changes were not made. When -1 is returned, it implies  
46391 everything failed even though some of the changes were made.

46392 Applications that need all of the requested changes made to work properly should follow  
46393 *tcsetattr()* with a call to *tcgetattr()* and compare the appropriate field values.

46394 **FUTURE DIRECTIONS**

46395 None.

46396 **SEE ALSO**

46397 *tcsetattr()*, the Base Definitions volume of IEEE Std 1003.1-200x, <**termios.h**>, the Base  
46398 Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface

46399 **CHANGE HISTORY**

46400 First released in Issue 3.

46401 Entry included for alignment with the POSIX.1-1988 standard.

46402 **Issue 6**

46403 In the DESCRIPTION, the rate returned as the input baud rate shall be the output rate.

46404 Previously, the number zero was also allowed but was obsolescent.

46405 **NAME**

46406 tcgetpgrp — get the foreground process group ID

46407 **SYNOPSIS**

46408 #include &lt;unistd.h&gt;

46409 pid\_t tcgetpgrp(int *fildev*);46410 **DESCRIPTION**46411 The *tcgetpgrp()* function shall return the value of the process group ID of the foreground process  
46412 group associated with the terminal.46413 If there is no foreground process group, *tcgetpgrp()* shall return a value greater than 1 that does  
46414 not match the process group ID of any existing process group.46415 The *tcgetpgrp()* function is allowed from a process that is a member of a background process  
46416 group; however, the information may be subsequently changed by a process that is a member of  
46417 a foreground process group.46418 **RETURN VALUE**46419 Upon successful completion, *tcgetpgrp()* shall return the value of the process group ID of the  
46420 foreground process associated with the terminal. Otherwise, -1 shall be returned and *errno* set to  
46421 indicate the error.46422 **ERRORS**46423 The *tcgetpgrp()* function shall fail if:46424 [EBADF] The *fildev* argument is not a valid file descriptor.46425 [ENOTTY] The calling process does not have a controlling terminal, or the file is not the  
46426 controlling terminal.46427 **EXAMPLES**

46428 None.

46429 **APPLICATION USAGE**

46430 None.

46431 **RATIONALE**

46432 None.

46433 **FUTURE DIRECTIONS**

46434 None.

46435 **SEE ALSO**46436 *setsid()*, *setpgid()*, *tcsetpgrp()*, the Base Definitions volume of IEEE Std 1003.1-200x,  
46437 <sys/types.h>, <unistd.h>46438 **CHANGE HISTORY**

46439 First released in Issue 3.

46440 Entry included for alignment with the POSIX.1-1988 standard.

46441 **Issue 6**

46442 In the SYNOPSIS, the optional include of the &lt;sys/types.h&gt; header is removed.

46443 The following new requirements on POSIX implementations derive from alignment with the  
46444 Single UNIX Specification:

- 46445 • The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was
- 
- 46446 required for conforming implementations of previous POSIX specifications, it was not
- 
- 46447 required for UNIX applications.

46448  
46449

- In the DESCRIPTION, text previously conditional on support for `_POSIX_JOB_CONTROL` is now mandatory. This is a FIPS requirement.

46450 **NAME**

46451 tcgetsid — get process group ID for session leader for controlling terminal

46452 **SYNOPSIS**

46453 XSI #include <termios.h>

46454 pid\_t tcgetsid(int *fildes*);

46455

46456 **DESCRIPTION**

46457 The *tcgetsid()* function shall obtain the process group ID of the session for which the terminal  
46458 specified by *fildes* is the controlling terminal.

46459 **RETURN VALUE**

46460 Upon successful completion, *tcgetsid()* shall return the process group ID associated with the  
46461 terminal. Otherwise, a value of (**pid\_t**)-1 shall be returned and *errno* set to indicate the error.

46462 **ERRORS**

46463 The *tcgetsid()* function shall fail if:

46464 [EBADF] The *fildes* argument is not a valid file descriptor.

46465 [ENOTTY] The calling process does not have a controlling terminal, or the file is not the  
46466 controlling terminal.

46467 **EXAMPLES**

46468 None.

46469 **APPLICATION USAGE**

46470 None.

46471 **RATIONALE**

46472 None.

46473 **FUTURE DIRECTIONS**

46474 None.

46475 **SEE ALSO**

46476 The Base Definitions volume of IEEE Std 1003.1-200x, <**termios.h**>

46477 **CHANGE HISTORY**

46478 First released in Issue 4, Version 2.

46479 **Issue 5**

46480 Moved from X/OPEN UNIX extension to BASE.

46481 The [EACCES] error has been removed from the list of mandatory errors, and the description of  
46482 [ENOTTY] has been reworded.

46483 **NAME**

46484 tcsendbreak — send a “break” for a specific duration

46485 **SYNOPSIS**

46486 #include &lt;termios.h&gt;

46487 int tcsendbreak(int *fildev*, int *duration*);46488 **DESCRIPTION**

46489 If the terminal is using asynchronous serial data transmission, *tcsendbreak()* shall cause |  
 46490 transmission of a continuous stream of zero-valued bits for a specific duration. If *duration* is 0, it |  
 46491 shall cause transmission of zero-valued bits for at least 0,25 seconds, and not more than 0,5 |  
 46492 seconds. If *duration* is not 0, it shall send zero-valued bits for an implementation-defined period |  
 46493 of time.

46494 The *fildev* argument is an open file descriptor associated with a terminal. |

46495 If the terminal is not using asynchronous serial data transmission, it is implementation-defined |  
 46496 whether *tcsendbreak()* sends data to generate a break condition or returns without taking any |  
 46497 action.

46498 Attempts to use *tcsendbreak()* from a process which is a member of a background process group |  
 46499 on a *fildev* associated with its controlling terminal shall cause the process group to be sent a |  
 46500 SIGTTOU signal. If the calling process is blocking or ignoring SIGTTOU signals, the process |  
 46501 shall be allowed to perform the operation, and no signal is sent.

46502 **RETURN VALUE**

46503 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to |  
 46504 indicate the error.

46505 **ERRORS**46506 The *tcsendbreak()* function shall fail if:46507 [EBADF] The *fildev* argument is not a valid file descriptor.46508 [ENOTTY] The file associated with *fildev* is not a terminal.46509 The *tcsendbreak()* function may fail if:

46510 [EIO] The process group of the writing process is orphaned, and the writing process |  
 46511 is not ignoring or blocking SIGTTOU.

46512 **EXAMPLES**

46513 None.

46514 **APPLICATION USAGE**

46515 None.

46516 **RATIONALE**

46517 None.

46518 **FUTURE DIRECTIONS**

46519 None.

46520 **SEE ALSO**

46521 The Base Definitions volume of IEEE Std 1003.1-200x, <termios.h>, <unistd.h>, the Base |  
 46522 Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface

46523 **CHANGE HISTORY**

46524 First released in Issue 3.

46525 Entry included for alignment with the POSIX.1-1988 standard.

46526 **Issue 6**46527 The following new requirements on POSIX implementations derive from alignment with the  
46528 Single UNIX Specification:46529 • In the DESCRIPTION, text previously conditional on `_POSIX_JOB_CONTROL` is now  
46530 mandated. This is a FIPS requirement.

46531 • The [EIO] error is added.

46532 **NAME**

46533 tcsetattr — set the parameters associated with the terminal

46534 **SYNOPSIS**

46535 #include &lt;termios.h&gt;

46536 int tcsetattr(int *fildev*, int *optional\_actions*,  
46537 const struct termios \**termios\_p*);46538 **DESCRIPTION**46539 The *tcsetattr()* function shall set the parameters associated with the terminal referred to by the  
46540 open file descriptor *fildev* (an open file descriptor associated with a terminal) from the **termios**  
46541 structure referenced by *termios\_p* as follows:

- 46542
- If *optional\_actions* is TCSANOW, the change shall occur immediately.
  - If *optional\_actions* is TCSADRAIN, the change shall occur after all output written to *fildev* is  
46543 transmitted. This function should be used when changing parameters that affect output.
  - If *optional\_actions* is TCSAFLUSH, the change shall occur after all output written to *fildev* is  
46544 transmitted, and all input so far received but not read shall be discarded before the change is  
46545 made.

46546 If the output baud rate stored in the **termios** structure pointed to by *termios\_p* is the zero baud  
46547 rate, B0, the modem control lines shall no longer be asserted. Normally, this shall disconnect the  
46548 line.46549 If the input baud rate stored in the **termios** structure pointed to by *termios\_p* is 0, the input baud  
46550 rate given to the hardware is the same as the output baud rate stored in the **termios** structure.46551 The *tcsetattr()* function shall return successfully if it was able to perform any of the requested  
46552 actions, even if some of the requested actions could not be performed. It shall set all the  
46553 attributes that the implementation supports as requested and leaves all the attributes not  
46554 supported by the implementation unchanged. If no part of the request can be honored, it shall  
46555 return  $-1$  and set *errno* to [EINVAL]. If the input and output baud rates differ and are a  
46556 combination that is not supported, neither baud rate shall be changed. A subsequent call to  
46557 *tcgetattr()* shall return the actual state of the terminal device (reflecting both the changes made  
46558 and not made in the previous *tcsetattr()* call). The *tcsetattr()* function shall not change the values  
46559 found in the **termios** structure under any circumstances.46560 The effect of *tcsetattr()* is undefined if the value of the **termios** structure pointed to by *termios\_p*  
46561 was not derived from the result of a call to *tcgetattr()* on *fildev*; an application should modify  
46562 only fields and flags defined by this volume of IEEE Std 1003.1-200x between the call to  
46563 *tcgetattr()* and *tcsetattr()*, leaving all other fields and flags unmodified.46564 No actions defined by this volume of IEEE Std 1003.1-200x, other than a call to *tcsetattr()* or a  
46565 close of the last file descriptor in the system associated with this terminal device, shall cause any  
46566 of the terminal attributes defined by this volume of IEEE Std 1003.1-200x to change.46567 If *tcsetattr()* is called from a process which is a member of a background process group on a  
46568 *fildev* associated with its controlling terminal:

- 46569
- If the calling process is blocking or ignoring SIGTTOU signals, the operation completes  
46570 normally and no signal is sent.
  - Otherwise, a SIGTTOU signal shall be sent to the process group.

46574 **RETURN VALUE**

46575       Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and  
46576       *errno* set to indicate the error.

46577 **ERRORS**

46578       The *tcsetattr()* function shall fail if:

46579       [EBADF]       The *fildev* argument is not a valid file descriptor.

46580       [EINTR]       A signal interrupted *tcsetattr()*.

46581       [EINVAL]       The *optional\_actions* argument is not a supported value, or an attempt was  
46582       made to change an attribute represented in the **termios** structure to an  
46583       unsupported value.

46584       [ENOTTY]       The file associated with *fildev* is not a terminal.

46585       The *tcsetattr()* function may fail if:

46586       [EIO]       The process group of the writing process is orphaned, and the writing process  
46587       is not ignoring or blocking SIGTTOU.

46588 **EXAMPLES**

46589       None.

46590 **APPLICATION USAGE**

46591       If trying to change baud rates, applications should call *tcsetattr()* then call *tcgetattr()* in order to  
46592       determine what baud rates were actually selected.

46593 **RATIONALE**

46594       The *tcsetattr()* function can be interrupted in the following situations:

- 46595       • It is interrupted while waiting for output to drain.
- 46596       • It is called from a process in a background process group and SIGTTOU is caught.

46597       See also the RATIONALE section in *tcgetattr()*.

46598 **FUTURE DIRECTIONS**

46599       Using an input baud rate of 0 to set the input rate equal to the output rate may not necessarily be  
46600       supported in a future version of this volume of IEEE Std 1003.1-200x.

46601 **SEE ALSO**

46602       *cfgetispeed()*, *tcgetattr()*, the Base Definitions volume of IEEE Std 1003.1-200x, <**termios.h**>,  
46603       <**unistd.h**>, the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal  
46604       Interface

46605 **CHANGE HISTORY**

46606       First released in Issue 3.

46607       Entry included for alignment with the POSIX.1-1988 standard.

46608 **Issue 6**

46609       The following new requirements on POSIX implementations derive from alignment with the  
46610       Single UNIX Specification:

- 46611       • In the DESCRIPTION, text previously conditional on `_POSIX_JOB_CONTROL` is now  
46612       mandated. This is a FIPS requirement.
- 46613       • The [EIO] error is added.

46614  
46615

In the DESCRIPTION, the text describing use of *tcsetattr()* from a process which is a member of a background process group is clarified.

46616 **NAME**

46617 tcsetpgrp — set the foreground process group ID

46618 **SYNOPSIS**

46619 #include &lt;unistd.h&gt;

46620 int tcsetpgrp(int *fildev*, pid\_t *pgid\_id*);46621 **DESCRIPTION**

46622 If the process has a controlling terminal, *tcsetpgrp()* shall set the foreground process group ID  
 46623 associated with the terminal to *pgid\_id*. The application shall ensure that the file associated with  
 46624 *fildev* is the controlling terminal of the calling process and the controlling terminal is currently  
 46625 associated with the session of the calling process. The application shall ensure that the value of  
 46626 *pgid\_id* matches a process group ID of a process in the same session as the calling process.

46627 Attempts to use *tcsetpgrp()* from a process which is a member of a background process group on  
 46628 a *fildev* associated with its controlling terminal shall cause the process group to be sent a  
 46629 SIGTTOU signal. If the calling process is blocking or ignoring SIGTTOU signals, the process  
 46630 shall be allowed to perform the operation, and no signal is sent.

46631 **RETURN VALUE**

46632 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to  
 46633 indicate the error.

46634 **ERRORS**46635 The *tcsetpgrp()* function shall fail if:

- |       |          |   |
|-------|----------|---|
| 46636 | [EBADF]  | The <i>fildev</i> argument is not a valid file descriptor.  |
| 46637 | [EINVAL] | This implementation does not support the value in the <i>pgid_id</i> argument.  |
| 46638 | [ENOTTY] | The calling process does not have a controlling terminal, or the file is not the<br>46639 controlling terminal, or the controlling terminal is no longer associated with<br>46640 the session of the calling process. |
| 46641 | [EPERM]  | The value of <i>pgid_id</i> is a value supported by the implementation, but does not<br>46642 match the process group ID of a process in the same session as the calling<br>46643 process.                            |

46644 **EXAMPLES**

46645 None.

46646 **APPLICATION USAGE**

46647 None.

46648 **RATIONALE**

46649 None.

46650 **FUTURE DIRECTIONS**

46651 None.

46652 **SEE ALSO**46653 *tcgetpgrp()*, the Base Definitions volume of IEEE Std 1003.1-200x, <sys/types.h>, <unistd.h>46654 **CHANGE HISTORY**

46655 First released in Issue 3.

46656 Entry included for alignment with the POSIX.1-1988 standard.

46657 **Issue 6**

46658 In the SYNOPSIS, the inclusion of `<sys/types.h>` is no longer required.

46659 The following new requirements on POSIX implementations derive from alignment with the  
46660 Single UNIX Specification:

46661 • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was  
46662 required for conforming implementations of previous POSIX specifications, it was not  
46663 required for UNIX applications.

46664 • In the DESCRIPTION and ERRORS sections, text previously conditional on  
46665 `_POSIX_JOB_CONTROL` is now mandated. This is a FIPS requirement.

46666 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

46667 The Open Group Corrigendum U047/4 is applied.

## 46668 NAME

46669 tdelete, tfind, tsearch, twalk — manage a binary search tree

## 46670 SYNOPSIS

```

46671 xSI #include <search.h>
46672 void *tdelete(const void *restrict key, void **restrict rootp,
46673             int(*compar)(const void *, const void *));
46674 void *tfind(const void *key, void *const *rootp,
46675            int(*compar)(const void *, const void *));
46676 void *tsearch(const void *key, void **rootp,
46677              int (*compar)(const void *, const void *));
46678 void twalk(const void *root,
46679           void (*action)(const void *, VISIT, int));
46680

```

## 46681 DESCRIPTION

46682 The *tdelete()*, *tfind()*, *tsearch()*, and *twalk()* functions manipulate binary search trees.  
46683 Comparisons are made with a user-supplied routine, the address of which is passed as the  
46684 *compar* argument. This routine is called with two arguments, the pointers to the elements being  
46685 compared. The application shall ensure that the user-supplied routine returns an integer less  
46686 than, equal to, or greater than 0, according to whether the first argument is to be considered less  
46687 than, equal to, or greater than the second argument. The comparison function need not compare  
46688 every byte, so arbitrary data may be contained in the elements in addition to the values being  
46689 compared.

46690 The *tsearch()* function shall build and access the tree. The *key* argument is a pointer to an element |  
46691 to be accessed or stored. If there is a node in the tree whose element is equal to the value pointed |  
46692 to by *key*, a pointer to this found node shall be returned. Otherwise, the value pointed to by *key* |  
46693 shall be inserted (that is, a new node is created and the value of *key* is copied to this node), and a |  
46694 pointer to this node returned. Only pointers are copied, so the application shall ensure that the |  
46695 calling routine stores the data. The *rootp* argument points to a variable that points to the root |  
46696 node of the tree. A null pointer value for the variable pointed to by *rootp* denotes an empty tree; |  
46697 in this case, the variable shall be set to point to the node which shall be at the root of the new |  
46698 tree.

46699 Like *tsearch()*, *tfind()* shall search for a node in the tree, returning a pointer to it if found.  
46700 However, if it is not found, *tfind()* shall return a null pointer. The arguments for *tfind()* are the  
46701 same as for *tsearch()*.

46702 The *tdelete()* function shall delete a node from a binary search tree. The arguments are the same |  
46703 as for *tsearch()*. The variable pointed to by *rootp* shall be changed if the deleted node was the |  
46704 root of the tree. The *tdelete()* function shall return a pointer to the parent of the deleted node, or a |  
46705 null pointer if the node is not found.

46706 The *twalk()* function shall traverse a binary search tree. The *root* argument is a pointer to the root |  
46707 node of the tree to be traversed. (Any node in a tree may be used as the root for a walk below |  
46708 that node.) The argument *action* is the name of a routine to be invoked at each node. This routine |  
46709 is, in turn, called with three arguments. The first argument shall be the address of the node being |  
46710 visited. The structure pointed to by this argument is unspecified and shall not be modified by |  
46711 the application, but it shall be possible to cast a pointer-to-node into a pointer-to-pointer-to- |  
46712 element to access the element stored in the node. The second argument shall be a value from an |  
46713 enumeration data type:

```

46714 typedef enum { preorder, postorder, endorder, leaf } VISIT;

```

46715 (defined in `<search.h>`), depending on whether this is the first, second, or third time that the  
 46716 node is visited (during a depth-first, left-to-right traversal of the tree), or whether the node is a  
 46717 leaf. The third argument shall be the level of the node in the tree, with the root being level 0. |

46718 If the calling function alters the pointer to the root, the result is undefined.

#### 46719 RETURN VALUE

46720 If the node is found, both `tsearch()` and `tfind()` shall return a pointer to it. If not, `tfind()` shall  
 46721 return a null pointer, and `tsearch()` shall return a pointer to the inserted item.

46722 A null pointer shall be returned by `tsearch()` if there is not enough space available to create a new  
 46723 node.

46724 A null pointer shall be returned by `tdelete()`, `tfind()`, and `tsearch()` if `rootp` is a null pointer on  
 46725 entry.

46726 The `tdelete()` function shall return a pointer to the parent of the deleted node, or a null pointer if  
 46727 the node is not found.

46728 The `twalk()` function shall not return a value.

#### 46729 ERRORS

46730 No errors are defined.

#### 46731 EXAMPLES

46732 The following code reads in strings and stores structures containing a pointer to each string and  
 46733 a count of its length. It then walks the tree, printing out the stored strings and their lengths in  
 46734 alphabetical order.

```
46735 #include <search.h>
46736 #include <string.h>
46737 #include <stdio.h>

46738 #define STRSZ    10000
46739 #define NODSZ    500

46740 struct node {      /* Pointers to these are stored in the tree. */
46741     char    *string;
46742     int     length;
46743 };

46744 char    string_space[STRSZ]; /* Space to store strings. */
46745 struct node nodes[NODSZ];   /* Nodes to store. */
46746 void    *root = NULL;      /* This points to the root. */

46747 int main(int argc, char *argv[])
46748 {
46749     char    *strptr = string_space;
46750     struct node *nodeptr = nodes;
46751     void    print_node(const void *, VISIT, int);
46752     int     i = 0, node_compare(const void *, const void *);

46753     while (gets(strptr) != NULL && i++ < NODSZ) {
46754         /* Set node. */
46755         nodeptr->string = strptr;
46756         nodeptr->length = strlen(strptr);
46757         /* Put node into the tree. */
46758         (void) tsearch((void *)nodeptr, (void **)&root,
46759             node_compare);
```

```

46760         /* Adjust pointers, so we do not overwrite tree. */
46761         strptr += nodeptr->length + 1;
46762         nodeptr++;
46763     }
46764     twalk(root, print_node);
46765     return 0;
46766 }
46767 /*
46768  * This routine compares two nodes, based on an
46769  * alphabetical ordering of the string field.
46770  */
46771 int
46772 node_compare(const void *node1, const void *node2)
46773 {
46774     return strcmp(((const struct node *) node1)->string,
46775                 ((const struct node *) node2)->string);
46776 }
46777 /*
46778  * This routine prints out a node, the second time
46779  * twalk encounters it or if it is a leaf.
46780  */
46781 void
46782 print_node(const void *ptr, VISIT order, int level)
46783 {
46784     const struct node *p = *(const struct node **) ptr;
46785     if (order == postorder || order == leaf) {
46786         (void) printf("string = %s, length = %d\n",
46787                     p->string, p->length);
46788     }
46789 }

```

#### 46790 APPLICATION USAGE

46791 The *root* argument to *twalk()* is one level of indirection less than the *rootp* arguments to *tdelete()*  
 46792 and *tsearch()*.

46793 There are two nomenclatures used to refer to the order in which tree nodes are visited. The  
 46794 *tsearch()* function uses **preorder**, **postorder**, and **endorder** to refer respectively to visiting a node  
 46795 before any of its children, after its left child and before its right, and after both its children. The  
 46796 alternative nomenclature uses **preorder**, **inorder**, and **postorder** to refer to the same visits, which  
 46797 could result in some confusion over the meaning of **postorder**.

#### 46798 RATIONALE

46799 None.

#### 46800 FUTURE DIRECTIONS

46801 None.

#### 46802 SEE ALSO

46803 *hcreate()*, *tsearch()*, the Base Definitions volume of IEEE Std 1003.1-200x, <[search.h](#)>

46804 **CHANGE HISTORY**

46805 First released in Issue 1. Derived from Issue 1 of the SVID.

46806 **Issue 5**

46807 The last paragraph of the DESCRIPTION was included as an APPLICATION USAGE note in  
46808 previous issues.

46809 **Issue 6**

46810 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

46811 The **restrict** keyword is added to the *tdelete()* prototype for alignment with the  
46812 ISO/IEC 9899:1999 standard.

46813 **NAME**

46814 tellmdir — current location of a named directory stream

46815 **SYNOPSIS**

46816 XSI #include &lt;dirent.h&gt;

46817 long tellmdir(DIR \*dirp);

46818

46819 **DESCRIPTION**46820 The *tellmdir()* function shall obtain the current location associated with the directory stream |  
46821 specified by *dirp*. |46822 If the most recent operation on the directory stream was a *seekdir()*, the directory position  
46823 returned from the *tellmdir()* shall be the same as that supplied as a *loc* argument for *seekdir()*.46824 **RETURN VALUE**46825 Upon successful completion, *tellmdir()* shall return the current location of the specified directory  
46826 stream.46827 **ERRORS**

46828 No errors are defined.

46829 **EXAMPLES**

46830 None.

46831 **APPLICATION USAGE**

46832 None.

46833 **RATIONALE**

46834 None.

46835 **FUTURE DIRECTIONS**

46836 None.

46837 **SEE ALSO**46838 *opendir()*, *readdir()*, *seekdir()*, the Base Definitions volume of IEEE Std 1003.1-200x, <dirent.h>46839 **CHANGE HISTORY**

46840 First released in Issue 2.

46841 **NAME**

46842 tempnam — create a name for a temporary file

46843 **SYNOPSIS**46844 XSI 

```
#include <stdio.h>
```

46845 

```
char *tempnam(const char *dir, const char *pfx);
```

46846

46847 **DESCRIPTION**46848 The *tempnam()* function shall generate a pathname that may be used for a temporary file. |

46849 The *tempnam()* function allows the user to control the choice of a directory. The *dir* argument  
 46850 points to the name of the directory in which the file is to be created. If *dir* is a null pointer or  
 46851 points to a string which is not a name for an appropriate directory, the path prefix defined as  
 46852 P\_tmpdir in the <stdio.h> header shall be used. If that directory is not accessible, an  
 46853 implementation-defined directory may be used.

46854 Many applications prefer their temporary files to have certain initial letter sequences in their  
 46855 names. The *pfx* argument should be used for this. This argument may be a null pointer or point  
 46856 to a string of up to five bytes to be used as the beginning of the filename.

46857 Some implementations of *tempnam()* may use *tmpnam()* internally. On such implementations, if  
 46858 called more than {TMP\_MAX} times in a single process, the behavior is implementation-defined.

46859 **RETURN VALUE**

46860 Upon successful completion, *tempnam()* shall allocate space for a string, put the generated  
 46861 pathname in that space, and return a pointer to it. The pointer shall be suitable for use in a  
 46862 subsequent call to *free()*. Otherwise, it shall return a null pointer and set *errno* to indicate the  
 46863 error.

46864 **ERRORS**46865 The *tempnam()* function shall fail if:

46866 [ENOMEM] Insufficient storage space is available.

46867 **EXAMPLES**46868 **Generating a Pathname** |

46869 The following example generates a pathname for a temporary file in directory **/tmp**, with the  
 46870 prefix *file*. After the filename has been created, the call to *free()* deallocates the space used to  
 46871 store the filename.

```
46872 #include <stdio.h>
46873 #include <stdlib.h>
46874 ...
46875 char *directory = "/tmp";
46876 char *fileprefix = "file";
46877 char *file;

46878 file = tempnam(directory, fileprefix);
46879 free(file);
```

46880 **APPLICATION USAGE**

46881 This function only creates pathnames. It is the application's responsibility to create and remove  
 46882 the files. Between the time a pathname is created and the file is opened, it is possible for some  
 46883 other process to create a file with the same name. Applications may find *tmpfile()* more useful.

46884 **RATIONALE**

46885           None.

46886 **FUTURE DIRECTIONS**

46887           None.

46888 **SEE ALSO**

46889           *fopen()*, *free()*, *open()*, *tmpfile()*, *tmpnam()*, *unlink()*, the Base Definitions volume of  
46890           IEEE Std 1003.1-200x, <**stdio.h**>

46891 **CHANGE HISTORY**

46892           First released in Issue 1. Derived from Issue 1 of the SVID.

46893 **Issue 5**

46894           The last paragraph of the DESCRIPTION was included as an APPLICATION USAGE note in  
46895           previous issues.

46896 **NAME**

46897       tfind — search binary search tree

46898 **SYNOPSIS**

46899 XSI       #include &lt;search.h&gt;

46900       void \*tfind(const void \*key, void \*const \*rootp,  
46901                 int (\*compar)(const void \*, const void \*));

46902

46903 **DESCRIPTION**46904       Refer to *tdelete()*.

46905 **NAME**

46906 tgamma, tgammaf, tgammaL — compute gamma() function

46907 **SYNOPSIS**

```
46908 #include <math.h>
46909 double tgamma(double x);
46910 float tgammaf(float x);
46911 long double tgammaL(long double x);
```

46912 **DESCRIPTION**

46913 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 46914 conflict between the requirements described here and the ISO C standard is unintentional. This  
 46915 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

46916 These functions shall compute the *gamma()* function of *x*.

46917 An application wishing to check for error situations should set *errno* to zero and call  
 46918 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 46919 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 46920 zero, an error has occurred.

46921 **RETURN VALUE**

46922 Upon successful completion, these functions shall return *Gamma(x)*.

46923 If *x* is a negative integer, a domain error shall occur, and either a NaN (if supported), or an  
 46924 implementation-defined value shall be returned.

46925 If the correct value would cause overflow, a range error shall occur and *tgamma()*, *tgammaf()*,  
 46926 and *tgammaL()* shall return the value of the macro HUGE\_VAL, HUGE\_VALF, or HUGE\_VALL,  
 46927 respectively.

46928 **MX** If *x* is NaN, a NaN shall be returned.

46929 If *x* is +Inf, *x* shall be returned.

46930 If *x* is ±0, a pole error shall occur, and *tgamma()*, *tgammaf()*, and *tgammaL()* shall return  
 46931 ±HUGE\_VAL, ±HUGE\_VALF, and ±HUGE\_VALL, respectively.

46932 If *x* is -Inf, a domain error shall occur, and either a NaN (if supported), or an implementation-  
 46933 defined value shall be returned.

46934 **ERRORS**

46935 These functions shall fail if:

46936 **MX** Domain Error The value of *x* is a negative integer, or *x* is -Inf.

46937 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero, |  
 46938 then *errno* shall be set to [EDOM]. If the integer expression (math\_errhandling |  
 46939 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception |  
 46940 shall be raised. |

46941 **MX** Pole Error The value of *x* is zero.

46942 If the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero, |  
 46943 then *errno* shall be set to [ERANGE]. If the integer expression |  
 46944 (math\_errhandling & MATH\_ERREXCEPT) is non-zero, then the divide-by- |  
 46945 zero floating-point exception shall be raised. |

46946 Range Error The value overflows.

46947 If the integer expression (`math_errhandling & MATH_ERRNO`) is non-zero, |  
46948 then *errno* shall be set to [ERANGE]. If the integer expression |  
46949 (`math_errhandling & MATH_ERREXCEPT`) is non-zero, then the overflow |  
46950 floating-point exception shall be raised. |

**46951 EXAMPLES**

46952 None.

**46953 APPLICATION USAGE**

46954 For IEEE Std 754-1985 **double**, overflow happens when  $0 < x < 1/\text{DBL\_MAX}$ , and  $171.7 < x$ .  
46955 Overflow also happens near negative integers.

46956 On error, the expressions (`math_errhandling & MATH_ERRNO`) and (`math_errhandling &`  
46957 `MATH_ERREXCEPT`) are independent of each other, but at least one of them must be non-zero.

**46958 RATIONALE**

46959 This function is named *tgamma()* in order to avoid conflicts with the historical *gamma()* and  
46960 *lgamma()* functions.

**46961 FUTURE DIRECTIONS**

46962 It is possible that the error response for a negative integer argument may be changed to a pole  
46963 error and a return value of  $\pm\text{Inf}$ .

**46964 SEE ALSO**

46965 *feclearexcept()*, *fetestexcept()*, *lgamma()*, the Base Definitions volume of IEEE Std 1003.1-200x, |  
46966 Section 4.18, Treatment of Error Conditions for Mathematical Functions, <math.h> |

**46967 CHANGE HISTORY**

46968 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

46969 **NAME**

46970           time — get time

46971 **SYNOPSIS**

46972           #include &lt;time.h&gt;

46973           time\_t time(time\_t \*tloc);

46974 **DESCRIPTION**

46975 CX       The functionality described on this reference page is aligned with the ISO C standard. Any  
 46976       conflict between the requirements described here and the ISO C standard is unintentional. This  
 46977       volume of IEEE Std 1003.1-200x defers to the ISO C standard.

46978 CX       The *time()* function shall return the value of time in seconds since the Epoch.

46979       The *tloc* argument points to an area where the return value is also stored. If *tloc* is a null pointer,  
 46980       no value is stored.

46981 **RETURN VALUE**

46982       Upon successful completion, *time()* shall return the value of time. Otherwise, **(time\_t)–1** shall be  
 46983       returned.

46984 **ERRORS**

46985       No errors are defined.

46986 **EXAMPLES**46987       **Getting the Current Time**

46988       The following example uses the *time()* function to calculate the time elapsed, in seconds, since  
 46989       January 1, 1970 0:00 UTC, *localtime()* to convert that value to a broken-down time, and *asctime()*  
 46990       to convert the broken-down time values into a printable string.

```
46991       #include <stdio.h>
46992       #include <time.h>
46993       main()
46994       {
46995       time_t result;
46996           result = time(NULL);
46997           printf("%s%ld secs since the Epoch\n",
46998                asctime(localtime(&result)),
46999                (long)result);
47000       return(0);
47001       }
```

47002       This example writes the current time to *stdout* in a form like this:

```
47003       Wed Jun 26 10:32:15 1996
47004       835810335 secs since the Epoch
```

47005 **Timing an Event**

47006 The following example gets the current time, prints it out in the user's format, and prints the  
47007 number of minutes to an event being timed.

```
47008 #include <time.h>
47009 #include <stdio.h>
47010 ...
47011 time_t now;
47012 int minutes_to_event;
47013 ...
47014 time(&now);
47015 minutes_to_event = ...;
47016 printf("The time is ");
47017 puts(asctime(localtime(&now)));
47018 printf("There are %d minutes to the event.\n",
47019         minutes_to_event);
47020 ...
```

47021 **APPLICATION USAGE**

47022 None.

47023 **RATIONALE**

47024 The *time()* function returns a value in seconds (type **time\_t**) while *times()* returns a set of values  
47025 in clock ticks (type **clock\_t**). Some historical implementations, such as 4.3 BSD, have  
47026 mechanisms capable of returning more precise times (see below). A generalized timing scheme  
47027 to unify these various timing mechanisms has been proposed but not adopted.

47028 Implementations in which **time\_t** is a 32-bit signed integer (many historical implementations)  
47029 fail in the year 2038. IEEE Std 1003.1-200x does not address this problem. However, the use of  
47030 the **time\_t** type is mandated in order to ease the eventual fix.

47031 The use of the **<time.h>**, header instead of **<sys/types.h>**, allows compatibility with the ISO C  
47032 standard.

47033 Many historical implementations (including Version 7) and the 1984 /usr/group standard use  
47034 **long** instead of **time\_t**. This volume of IEEE Std 1003.1-200x uses the latter type in order to agree  
47035 with the ISO C standard.

47036 4.3 BSD includes *time()* only as an alternate function to the more flexible *gettimeofday()* function.

47037 **FUTURE DIRECTIONS**

47038 In a future version of this volume of IEEE Std 1003.1-200x, **time\_t** is likely to be required to be  
47039 capable of representing times far in the future. Whether this will be mandated as a 64-bit type or  
47040 a requirement that a specific date in the future be representable (for example, 10000 AD) is not  
47041 yet determined. Systems purchased after the approval of this volume of IEEE Std 1003.1-200x  
47042 should be evaluated to determine whether their lifetime will extend past 2038.

47043 **SEE ALSO**

47044 *asctime()*, *clock()*, *ctime()*, *difftime()*, *gmtime()*, *localtime()*, *mktime()*, *strptime()*, *strptime()*, *utime()*,  
47045 the Base Definitions volume of IEEE Std 1003.1-200x, **<time.h>**

47046 **CHANGE HISTORY**

47047 First released in Issue 1. Derived from Issue 1 of the SVID.

47048 **Issue 6**

47049 Extensions beyond the ISO C standard are now marked. |

47050 The EXAMPLES, RATIONALE, and FUTURE DIRECTIONS sections are added. |

## 47051 NAME

47052 timer\_create — create a per-process timer (**REALTIME**)

## 47053 SYNOPSIS

47054 TMR #include &lt;signal.h&gt;

47055 #include &lt;time.h&gt;

47056 int timer\_create(clockid\_t *clockid*, struct sigevent \*restrict *evp*,47057 timer\_t \*restrict *timerid*);

47058

## 47059 DESCRIPTION

47060 The *timer\_create()* function shall create a per-process timer using the specified clock, *clock\_id*, as  
 47061 the timing base. The *timer\_create()* function shall return, in the location referenced by *timerid*, a  
 47062 timer ID of type **timer\_t** used to identify the timer in timer requests. This timer ID shall be  
 47063 unique within the calling process until the timer is deleted. The particular clock, *clock\_id*, is  
 47064 defined in <**time.h**>. The timer whose ID is returned shall be in a disarmed state upon return  
 47065 from *timer\_create()*.

47066 The *evp* argument, if non-NULL, points to a **sigevent** structure. This structure, allocated by the  
 47067 application, defines the asynchronous notification to occur as specified in Section 2.4.1 (on page  
 47068 478) when the timer expires. If the *evp* argument is NULL, the effect is as if the *evp* argument  
 47069 pointed to a **sigevent** structure with the *sigev\_notify* member having the value SIGEV\_SIGNAL,  
 47070 the *sigev\_signo* having a default signal number, and the *sigev\_value* member having the value of  
 47071 the timer ID.

47072 Each implementation shall define a set of clocks that can be used as timing bases for per-process  
 47073 MON timers. All implementations shall support a *clock\_id* of CLOCK\_REALTIME. If the Monotonic  
 47074 Clock option is supported, implementations shall support a *clock\_id* of CLOCK\_MONOTONIC.

47075 Per-process timers shall not be inherited by a child process across a *fork()* and shall be disarmed  
 47076 and deleted by an *exec*.

47077 CPT If \_POSIX\_CPUTIME is defined, implementations shall support *clock\_id* values representing the  
 47078 CPU-time clock of the calling process.

47079 TCT If \_POSIX\_THREAD\_CPUTIME is defined, implementations shall support *clock\_id* values  
 47080 representing the CPU-time clock of the calling thread.

47081 CPT|TCT It is implementation-defined whether a *timer\_create()* function will succeed if the value defined  
 47082 by *clock\_id* corresponds to the CPU-time clock of a process or thread different from the process  
 47083 or thread invoking the function.

## 47084 RETURN VALUE

47085 If the call succeeds, *timer\_create()* shall return zero and update the location referenced by *timerid*  
 47086 to a **timer\_t**, which can be passed to the per-process timer calls. If an error occurs, the function  
 47087 shall return a value of -1 and set *errno* to indicate the error. The value of *timerid* is undefined if  
 47088 an error occurs.

## 47089 ERRORS

47090 The *timer\_create()* function shall fail if:

47091 [EAGAIN] The system lacks sufficient signal queuing resources to honor the request.

47092 [EAGAIN] The calling process has already created all of the timers it is allowed by this  
47093 implementation.

47094 [EINVAL] The specified clock ID is not defined.

47095 CPT|TCT [ENOTSUP] The implementation does not support the creation of a timer attached to the  
47096 CPU-time clock that is specified by *clock\_id* and associated with a process or  
47097 thread different from the process or thread invoking *timer\_create()*.

47098 **EXAMPLES**

47099 None.

47100 **APPLICATION USAGE**

47101 None.

47102 **RATIONALE**

47103 **Periodic Timer Overrun and Resource Allocation**

47104 The specified timer facilities may deliver realtime signals (that is, queued signals) on |  
47105 implementations that support this option. Since realtime applications cannot afford to lose |  
47106 notifications of asynchronous events, like timer expirations or asynchronous I/O completions, it |  
47107 must be possible to ensure that sufficient resources exist to deliver the signal when the event |  
47108 occurs. In general, this is not a difficulty because there is a one-to-one correspondence between a |  
47109 request and a subsequent signal generation. If the request cannot allocate the signal delivery |  
47110 resources, it can fail the call with an [EAGAIN] error.

47111 Periodic timers are a special case. A single request can generate an unspecified number of |  
47112 signals. This is not a problem if the requesting process can service the signals as fast as they are |  
47113 generated, thus making the signal delivery resources available for delivery of subsequent |  
47114 periodic timer expiration signals. But, in general, this cannot be assured—processing of periodic |  
47115 timer signals may “overrun”; that is, subsequent periodic timer expirations may occur before the |  
47116 currently pending signal has been delivered.

47117 Also, for signals, according to the POSIX.1-1990 standard, if subsequent occurrences of a |  
47118 pending signal are generated, it is implementation-defined whether a signal is delivered for each |  
47119 occurrence. This is not adequate for some realtime applications. So a mechanism is required to |  
47120 allow applications to detect how many timer expirations were delayed without requiring an |  
47121 indefinite amount of system resources to store the delayed expirations.

47122 The specified facilities provide for an overrun count. The overrun count is defined as the number |  
47123 of extra timer expirations that occurred between the time a timer expiration signal is generated |  
47124 and the time the signal is delivered. The signal-catching function, if it is concerned with |  
47125 overruns, can retrieve this count on entry. With this method, a periodic timer only needs one |  
47126 “signal queuing resource” that can be allocated at the time of the *timer\_create()* function call.

47127 A function is defined to retrieve the overrun count so that an application need not allocate static |  
47128 storage to contain the count, and an implementation need not update this storage |  
47129 asynchronously on timer expirations. But, for some high-frequency periodic applications, the |  
47130 overhead of an additional system call on each timer expiration may be prohibitive. The |  
47131 functions, as defined, permit an implementation to maintain the overrun count in user space, |  
47132 associated with the *timerid*. The *timer\_getoverrun()* function can then be implemented as a macro |  
47133 that uses the *timerid* argument (which may just be a pointer to a user space structure containing |  
47134 the counter) to locate the overrun count with no system call overhead. Other implementations, |  
47135 less concerned with this class of applications, can avoid the asynchronous update of user space |  
47136 by maintaining the count in a system structure at the cost of the extra system call to obtain it.

47137 **Timer Expiration Signal Parameters**

47138 The Realtime Signals Extension option supports an application-specific datum that is delivered  
47139 to the extended signal handler. This value is explicitly specified by the application, along with  
47140 the signal number to be delivered, in a **sigevent** structure. The type of the application-defined  
47141 value can be either an integer constant or a pointer. This explicit specification of the value, as  
47142 opposed to always sending the timer ID, was selected based on existing practice.

47143 It is common practice for realtime applications (on non-POSIX systems or realtime extended  
47144 POSIX systems) to use the parameters of event handlers as the case label of a switch statement  
47145 or as a pointer to an application-defined data structure. Since *timer\_ids* are dynamically allocated  
47146 by the *timer\_create()* function, they can be used for neither of these functions without additional  
47147 application overhead in the signal handler; for example, to search an array of saved timer IDs to  
47148 associate the ID with a constant or application data structure.

47149 **FUTURE DIRECTIONS**

47150 None.

47151 **SEE ALSO**

47152 *clock\_getres()*, *timer\_delete()*, *timer\_getoverrun()*, the Base Definitions volume of  
47153 IEEE Std 1003.1-200x, <**time.h**>

47154 **CHANGE HISTORY**

47155 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

47156 **Issue 6**

47157 The *timer\_create()* function is marked as part of the Timers option.

47158 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
47159 implementation does not support the Timers option.

47160 CPU-time clocks are added for alignment with IEEE Std 1003.1d-1999.

47161 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by adding the  
47162 requirement for the CLOCK\_MONOTONIC clock under the Monotonic Clock option.

47163 The **restrict** keyword is added to the *timer\_create()* prototype for alignment with the  
47164 ISO/IEC 9899:1999 standard.

47165 **NAME**

47166 timer\_delete — delete a per-process timer (**REALTIME**)

47167 **SYNOPSIS**

```
47168 TMR #include <time.h>
```

```
47169 int timer_delete(timer_t timerid);
```

47170

47171 **DESCRIPTION**

47172 The *timer\_delete()* function deletes the specified timer, *timerid*, previously created by the  
47173 *timer\_create()* function. If the timer is armed when *timer\_delete()* is called, the behavior shall be  
47174 as if the timer is automatically disarmed before removal. The disposition of pending signals for  
47175 the deleted timer is unspecified.

47176 **RETURN VALUE**

47177 If successful, the *timer\_delete()* function shall return a value of zero. Otherwise, the function shall  
47178 return a value of  $-1$  and set *errno* to indicate the error.

47179 **ERRORS**

47180 The *timer\_delete()* function shall fail if:

47181 [EINVAL] The timer ID specified by *timerid* is not a valid timer ID.

47182 **EXAMPLES**

47183 None.

47184 **APPLICATION USAGE**

47185 None.

47186 **RATIONALE**

47187 None.

47188 **FUTURE DIRECTIONS**

47189 None.

47190 **SEE ALSO**

47191 *timer\_create()*, the Base Definitions volume of IEEE Std 1003.1-200x, <**time.h**>

47192 **CHANGE HISTORY**

47193 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

47194 **Issue 6**

47195 The *timer\_delete()* function is marked as part of the Timers option.

47196 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
47197 implementation does not support the Timers option.

## 47198 NAME

47199 timer\_getoverrun, timer\_gettime, timer\_settime — per-process timers (REALTIME)

## 47200 SYNOPSIS

```
47201 TMR #include <time.h>
47202
47202 int timer_getoverrun(timer_t timerid);
47203 int timer_gettime(timer_t timerid, struct itimerspec *value);
47204 int timer_settime(timer_t timerid, int flags,
47205                 const struct itimerspec *restrict value,
47206                 struct itimerspec *restrict ovalue);
47207
```

## 47208 DESCRIPTION

47209 The *timer\_gettime()* function shall store the amount of time until the specified timer, *timerid*, |  
 47210 expires and the reload value of the timer into the space pointed to by the *value* argument. The |  
 47211 *it\_value* member of this structure shall contain the amount of time before the timer expires, or |  
 47212 zero if the timer is disarmed. This value is returned as the interval until timer expiration, even if |  
 47213 the timer was armed with absolute time. The *it\_interval* member of *value* shall contain the reload |  
 47214 value last set by *timer\_settime()*.

47215 The *timer\_settime()* function shall set the time until the next expiration of the timer specified by |  
 47216 *timerid* from the *it\_value* member of the *value* argument and arms the timer if the *it\_value* |  
 47217 member of *value* is non-zero. If the specified timer was already armed when *timer\_settime()* is |  
 47218 called, this call shall reset the time until next expiration to the *value* specified. If the *it\_value* |  
 47219 member of *value* is zero, the timer shall be disarmed. The effect of disarming or resetting a timer |  
 47220 with pending expiration notifications is unspecified.

47221 If the flag `TIMER_ABSTIME` is not set in the argument *flags*, *timer\_settime()* shall behave as if the |  
 47222 time until next expiration is set to be equal to the interval specified by the *it\_value* member of |  
 47223 *value*. That is, the timer shall expire in *it\_value* nanoseconds from when the call is made. If the |  
 47224 flag `TIMER_ABSTIME` is set in the argument *flags*, *timer\_settime()* shall behave as if the time |  
 47225 until next expiration is set to be equal to the difference between the absolute time specified by |  
 47226 the *it\_value* member of *value* and the current value of the clock associated with *timerid*. That is, |  
 47227 the timer shall expire when the clock reaches the value specified by the *it\_value* member of *value*. |  
 47228 If the specified time has already passed, the function shall succeed and the expiration |  
 47229 notification shall be made.

47230 The reload value of the timer shall be set to the value specified by the *it\_interval* member of |  
 47231 *value*. When a timer is armed with a non-zero *it\_interval*, a periodic (or repetitive) timer is |  
 47232 specified.

47233 Time values that are between two consecutive non-negative integer multiples of the resolution |  
 47234 of the specified timer shall be rounded up to the larger multiple of the resolution. Quantization |  
 47235 error shall not cause the timer to expire earlier than the rounded time value.

47236 If the argument *ovalue* is not NULL, the function *timer\_settime()* shall store, in the location |  
 47237 referenced by *ovalue*, a value representing the previous amount of time before the timer would |  
 47238 have expired, or zero if the timer was disarmed, together with the previous timer reload value. |  
 47239 Timers shall not expire before their scheduled time.

47240 Only a single signal shall be queued to the process for a given timer at any point in time. When a |  
 47241 timer for which a signal is still pending expires, no signal shall be queued, and a timer overrun |  
 47242 shall occur. When a timer expiration signal is delivered to or accepted by a process, if the |  
 47243 implementation supports the Realtime Signals Extension, the *timer\_getoverrun()* function shall |  
 47244 return the timer expiration overrun count for the specified timer. The overrun count returned |  
 47245 contains the number of extra timer expirations that occurred between the time the signal was

47246 generated (queued) and when it was delivered or accepted, up to but not including an  
47247 implementation-defined maximum of {DELAYTIMER\_MAX}. If the number of such extra  
47248 expirations is greater than or equal to {DELAYTIMER\_MAX}, then the overrun count shall be set  
47249 to {DELAYTIMER\_MAX}. The value returned by *timer\_getoverrun()* shall apply to the most  
47250 recent expiration signal delivery or acceptance for the timer. If no expiration signal has been  
47251 delivered for the timer, or if the Realtime Signals Extension is not supported, the return value of  
47252 *timer\_getoverrun()* is unspecified.

#### 47253 RETURN VALUE

47254 If the *timer\_getoverrun()* function succeeds, it shall return the timer expiration overrun count as  
47255 explained above.

47256 If the *timer\_gettime()* or *timer\_settime()* functions succeed, a value of 0 shall be returned.

47257 If an error occurs for any of these functions, the value -1 shall be returned, and *errno* set to  
47258 indicate the error.

#### 47259 ERRORS

47260 The *timer\_getoverrun()*, *timer\_gettime()*, and *timer\_settime()* functions shall if:

47261 [EINVAL] The *timerid* argument does not correspond to an ID returned by *timer\_create()*  
47262 but not yet deleted by *timer\_delete()*.

47263 The *timer\_settime()* function shall fail if:

47264 [EINVAL] A *value* structure specified a nanosecond value less than zero or greater than  
47265 or equal to 1,000 million, and the *it\_value* member of that structure did not  
47266 specify zero seconds and nanoseconds.

#### 47267 EXAMPLES

47268 None.

#### 47269 APPLICATION USAGE

47270 None.

#### 47271 RATIONALE

47272 Practical clocks tick at a finite rate, with rates of 100 Hertz and 1,000 Hertz being common. The  
47273 inverse of this tick rate is the clock resolution, also called the clock granularity, which in either  
47274 case is expressed as a time duration, being 10 milliseconds and 1 millisecond respectively for  
47275 these common rates. The granularity of practical clocks implies that if one reads a given clock  
47276 twice in rapid succession, one may get the same time value twice; and that timers must wait for  
47277 the next clock tick after the theoretical expiration time, to ensure that a timer never returns too  
47278 soon. Note also that the granularity of the clock may be significantly coarser than the resolution  
47279 of the data format used to set and get time and interval values. Also note that some  
47280 implementations may choose to adjust time and/or interval values to exactly match the ticks of  
47281 the underlying clock.

47282 This volume of IEEE Std 1003.1-200x defines functions that allow an application to determine the  
47283 implementation-supported resolution for the clocks and requires an implementation to  
47284 document the resolution supported for timers and *nanosleep()* if they differ from the supported  
47285 clock resolution. This is more of a procurement issue than a runtime application issue.

#### 47286 FUTURE DIRECTIONS

47287 None.

47288 **SEE ALSO**

47289 *clock\_getres()*, *timer\_create()*, the Base Definitions volume of IEEE Std 1003.1-200x, <**time.h**>

47290 **CHANGE HISTORY**

47291 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

47292 **Issue 6**

47293 The *timer\_getoverrun()*, *timer\_gettime()*, and *timer\_settime()* functions are marked as part of the  
47294 Timers option.

47295 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
47296 implementation does not support the Timers option.

47297 The [EINVAL] error condition is updated to include the following: “and the *it\_value* member of  
47298 that structure did not specify zero seconds and nanoseconds.” This change is for IEEE PASC  
47299 Interpretation 1003.1 #89.

47300 The DESCRIPTION for *timer\_getoverrun()* is updated to clarify that “If no expiration signal has  
47301 been delivered for the timer, or if the Realtime Signals Extension is not supported, the return  
47302 value of *timer\_getoverrun()* is unspecified”.

47303 The **restrict** keyword is added to the *timer\_settime()* prototype for alignment with the  
47304 ISO/IEC 9899:1999 standard.

47305 **NAME**

47306 times — get process and waited-for child process times

47307 **SYNOPSIS**

47308 #include &lt;sys/times.h&gt;

47309 clock\_t times(struct tms \*buffer);

47310 **DESCRIPTION**47311 The *times()* function shall fill the **tms** structure pointed to by *buffer* with time-accounting  
47312 information. The **tms** structure is defined in <sys/times.h>.

47313 All times are measured in terms of the number of clock ticks used.

47314 The times of a terminated child process shall be included in the *tms\_cutime* and *tms\_cstime*  
47315 elements of the parent when *wait()* or *waitpid()* returns the process ID of this terminated child. If  
47316 a child process has not waited for its children, their times shall not be included in its times.47317 • The *tms\_utime* structure member is the CPU time charged for the execution of user  
47318 instructions of the calling process.47319 • The *tms\_stime* structure member is the CPU time charged for execution by the system on  
47320 behalf of the calling process.47321 • The *tms\_cutime* structure member is the sum of the *tms\_utime* and *tms\_cutime* times of the  
47322 child processes.47323 • The *tms\_cstime* structure member is the sum of the *tms\_stime* and *tms\_cstime* times of the child  
47324 processes.47325 **RETURN VALUE**47326 Upon successful completion, *times()* shall return the elapsed real time, in clock ticks, since an  
47327 arbitrary point in the past (for example, system start-up time). This point does not change from  
47328 one invocation of *times()* within the process to another. The return value may overflow the  
47329 possible range of type **clock\_t**. If *times()* fails, **(clock\_t)-1** shall be returned and *errno* set to  
47330 indicate the error.47331 **ERRORS**

47332 No errors are defined.

47333 **EXAMPLES**47334 **Timing a Database Lookup**47335 The following example defines two functions, *start\_clock()* and *end\_clock()*, that are used to time  
47336 a lookup. It also defines variables of type **clock\_t** and **tms** to measure the duration of  
47337 transactions. The *start\_clock()* function saves the beginning times given by the *times()* function.  
47338 The *end\_clock()* function gets the ending times and prints the difference between the two times.47339 #include <sys/times.h>  
47340 #include <stdio.h>  
47341 ...  
47342 void start\_clock(void);  
47343 void end\_clock(char \*msg);  
47344 ...  
47345 static clock\_t st\_time;  
47346 static clock\_t en\_time;  
47347 static struct tms st\_cpu;  
47348 static struct tms en\_cpu;

```

47349     ...
47350     void
47351     start_clock()
47352     {
47353         st_time = times(&st_cpu);
47354     }
47355     void
47356     end_clock(char *msg)
47357     {
47358         en_time = times(&en_cpu);
47359
47359         printf(msg);
47360         printf("Real Time: %ld, User Time %ld, System Time %ld\n",
47361             en_time - st_time,
47362             en_cpu.tms_utime - st_cpu.tms_utime,
47363             en_cpu.tms_stime - st_cpu.tms_stime);
47364     }

```

**47365 APPLICATION USAGE**

47366 Applications should use `sysconf(_SC_CLK_TCK)` to determine the number of clock ticks per  
47367 second as it may vary from system to system.

**47368 RATIONALE**

47369 The accuracy of the times reported is intentionally left unspecified to allow implementations  
47370 flexibility in design, from uniprocessor to multi-processor networks.

47371 The inclusion of times of child processes is recursive, so that a parent process may collect the  
47372 total times of all of its descendants. But the times of a child are only added to those of its parent  
47373 when its parent successfully waits on the child. Thus, it is not guaranteed that a parent process  
47374 can always see the total times of all its descendants; see also the discussion of the term *realtime* in  
47375 *alarm()*.

47376 If the type `clock_t` is defined to be a signed 32-bit integer, it overflows in somewhat more than a  
47377 year if there are 60 clock ticks per second, or less than a year if there are 100. There are individual  
47378 systems that run continuously for longer than that. This volume of IEEE Std 1003.1-200x permits  
47379 an implementation to make the reference point for the returned value be the start-up time of the  
47380 process, rather than system start-up time.

47381 The term *charge* in this context has nothing to do with billing for services. The operating system  
47382 accounts for time used in this way. That information must be correct, regardless of how that  
47383 information is used.

**47384 FUTURE DIRECTIONS**

47385 None.

**47386 SEE ALSO**

47387 `exec`, `fork()`, `sysconf()`, `time()`, `wait()`, the Base Definitions volume of IEEE Std 1003.1-200x,  
47388 `<sys/times.h>`

**47389 CHANGE HISTORY**

47390 First released in Issue 1. Derived from Issue 1 of the SVID.

47391 **NAME**

47392            timezone — difference from UTC and local standard time

47393 **SYNOPSIS**

47394 xSI        #include <time.h>

47395            extern long timezone;

47396

47397 **DESCRIPTION**

47398            Refer to *tzset()*.

47399 **NAME**

47400 tmpfile — create a temporary file

47401 **SYNOPSIS**

47402 #include &lt;stdio.h&gt;

47403 FILE \*tmpfile(void);

47404 **DESCRIPTION**

47405 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 47406 conflict between the requirements described here and the ISO C standard is unintentional. This  
 47407 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

47408 The *tmpfile()* function shall create a temporary file and open a corresponding stream. The file  
 47409 shall be automatically deleted when all references to the file are closed. The file is opened as in  
 47410 *fopen()* for update (*w+*).

47411 CX In some implementations, a permanent file may be left behind if the process calling *tmpfile()* is  
 47412 killed while it is processing a call to *tmpfile()*.

47413 An error message may be written to standard error if the stream cannot be opened.

47414 **RETURN VALUE**

47415 Upon successful completion, *tmpfile()* shall return a pointer to the stream of the file that is  
 47416 CX created. Otherwise, it shall return a null pointer and set *errno* to indicate the error.

47417 **ERRORS**47418 The *tmpfile()* function shall fail if:

47419 CX [EINTR] A signal was caught during *tmpfile()*.

47420 CX [EMFILE] {OPEN\_MAX} file descriptors are currently open in the calling process.

47421 CX [ENFILE] The maximum allowable number of files is currently open in the system.

47422 CX [ENOSPC] The directory or file system which would contain the new file cannot be  
 47423 expanded.

47424 CX [EOVERFLOW] The file is a regular file and the size of the file cannot be represented correctly  
 47425 in an object of type *off\_t*.

47426 The *tmpfile()* function may fail if:

47427 CX [EMFILE] {FOPEN\_MAX} streams are currently open in the calling process.

47428 CX [ENOMEM] Insufficient storage space is available.

47429 **EXAMPLES**47430 **Creating a Temporary File**

47431 The following example creates a temporary file for update, and returns a pointer to a stream for  
 47432 the created file in the *fp* variable.

47433 #include &lt;stdio.h&gt;

47434 ...

47435 FILE \*fp;

47436 fp = tmpfile ();

47437 **APPLICATION USAGE**

47438 It should be possible to open at least {TMP\_MAX} temporary files during the lifetime of the  
47439 program (this limit may be shared with *tmpnam()*) and there should be no limit on the number  
47440 simultaneously open other than this limit and any limit on the number of open files  
47441 ({FOPEN\_MAX}).

47442 **RATIONALE**

47443 None.

47444 **FUTURE DIRECTIONS**

47445 None.

47446 **SEE ALSO**

47447 *fopen()*, *tmpnam()*, *unlink()*, the Base Definitions volume of IEEE Std 1003.1-200x, <**stdio.h**>

47448 **CHANGE HISTORY**

47449 First released in Issue 1. Derived from Issue 1 of the SVID.

47450 **Issue 5**

47451 Large File Summit extensions are added.

47452 The last two paragraphs of the DESCRIPTION were included as APPLICATION USAGE notes  
47453 in previous issues.

47454 **Issue 6**

47455 Extensions beyond the ISO C standard are now marked.

47456 The following new requirements on POSIX implementations derive from alignment with the  
47457 Single UNIX Specification:

- 47458 • In the ERRORS section, the [Eoverflow] condition is added. This change is to support  
47459 large files.
- 47460 • The [EMFILE] optional error condition is added.

47461 The APPLICATION USAGE section is added for alignment with the ISO/IEC 9899:1999  
47462 standard.

47463 **NAME**

47464 tmpnam — create a name for a temporary file

47465 **SYNOPSIS**

47466 #include &lt;stdio.h&gt;

47467 char \*tmpnam(char \*s);

47468 **DESCRIPTION**

47469 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
 47470 conflict between the requirements described here and the ISO C standard is unintentional. This  
 47471 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

47472 The *tmpnam()* function shall generate a string that is a valid filename and that is not the same as  
 47473 the name of an existing file. The function is potentially capable of generating {TMP\_MAX}  
 47474 different strings, but any or all of them may already be in use by existing files and thus not be  
 47475 suitable return values.

47476 The *tmpnam()* function generates a different string each time it is called from the same process,  
 47477 up to {TMP\_MAX} times. If it is called more than {TMP\_MAX} times, the behavior is  
 47478 implementation-defined.

47479 The implementation shall behave as if no function defined in this volume of  
 47480 IEEE Std 1003.1-200x calls *tmpnam()*.

47481 cx If the application uses any of the functions guaranteed to be available if either  
 47482 `_POSIX_THREAD_SAFE_FUNCTIONS` or `_POSIX_THREADS` is defined, the application shall  
 47483 ensure that the *tmpnam()* function is called with a non-NULL parameter.

47484 **RETURN VALUE**

47485 Upon successful completion, *tmpnam()* shall return a pointer to a string. If no suitable string can  
 47486 be generated, the *tmpnam()* function shall return a null pointer.

47487 If the argument *s* is a null pointer, *tmpnam()* shall leave its result in an internal static object and  
 47488 return a pointer to that object. Subsequent calls to *tmpnam()* may modify the same object. If the  
 47489 argument *s* is not a null pointer, it is presumed to point to an array of at least `L_tmpnam` **chars**;  
 47490 *tmpnam()* shall write its result in that array and shall return the argument as its value.

47491 **ERRORS**

47492 No errors are defined.

47493 **EXAMPLES**47494 **Generating a Filename**47495 The following example generates a unique filename and stores it in the array pointed to by *ptr*.

47496 #include &lt;stdio.h&gt;

47497 ...

47498 char filename[L\_tmpnam+1];

47499 char \*ptr;

47500 ptr = tmpnam(filename);

47501 **APPLICATION USAGE**

47502 This function only creates filenames. It is the application's responsibility to create and remove  
 47503 the files.

47504 Between the time a pathname is created and the file is opened, it is possible for some other  
 47505 process to create a file with the same name. Applications may find *tmpfile()* more useful.

47506 **RATIONALE**

47507           None.

47508 **FUTURE DIRECTIONS**

47509           None.

47510 **SEE ALSO**47511           *fopen()*, *open()*, *tmpnam()*, *tmpfile()*, *unlink()*, the Base Definitions volume of  
47512           IEEE Std 1003.1-200x, <**stdio.h**>47513 **CHANGE HISTORY**

47514           First released in Issue 1. Derived from Issue 1 of the SVID.

47515 **Issue 5**

47516           The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

47517 **Issue 6**

47518           Extensions beyond the ISO C standard are now marked.

47519           The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

47520           The DESCRIPTION is expanded for alignment with the ISO/IEC 9899:1999 standard.

47521 **NAME**

47522           toascii — translate integer to a 7-bit ASCII character

47523 **SYNOPSIS**

47524 XSI       #include &lt;ctype.h&gt;

47525           int toascii(int c);

47526

47527 **DESCRIPTION**47528           The *toascii()* function shall convert its argument into a 7-bit ASCII character.47529 **RETURN VALUE**47530           The *toascii()* function shall return the value (*c* &0x7f).47531 **ERRORS**

47532           No errors are returned.

47533 **EXAMPLES**

47534           None.

47535 **APPLICATION USAGE**

47536           None.

47537 **RATIONALE**

47538           None.

47539 **FUTURE DIRECTIONS**

47540           None.

47541 **SEE ALSO**47542           *isascii()*, the Base Definitions volume of IEEE Std 1003.1-200x, <ctype.h>47543 **CHANGE HISTORY**

47544           First released in Issue 1. Derived from Issue 1 of the SVID.

47545 **NAME**

47546           tolower — transliterate uppercase characters to lowercase

47547 **SYNOPSIS**

47548           #include &lt;ctype.h&gt;

47549           int tolower(int c);

47550 **DESCRIPTION**

47551 cx       The functionality described on this reference page is aligned with the ISO C standard. Any  
47552       conflict between the requirements described here and the ISO C standard is unintentional. This  
47553       volume of IEEE Std 1003.1-200x defers to the ISO C standard.

47554       The *tolower()* function has as a domain a type **int**, the value of which is representable as an  
47555       **unsigned char** or the value of EOF. If the argument has any other value, the behavior is  
47556       undefined. If the argument of *tolower()* represents an uppercase letter, and there exists a  
47557 cx       corresponding lowercase letter (as defined by character type information in the program locale  
47558       category *LC\_CTYPE*), the result shall be the corresponding lowercase letter. All other arguments  
47559       in the domain are returned unchanged. |

47560 **RETURN VALUE**

47561       Upon successful completion, *tolower()* shall return the lowercase letter corresponding to the  
47562       argument passed; otherwise, it shall return the argument unchanged.

47563 **ERRORS**

47564       No errors are defined.

47565 **EXAMPLES**

47566       None.

47567 **APPLICATION USAGE**

47568       None.

47569 **RATIONALE**

47570       None.

47571 **FUTURE DIRECTIONS**

47572       None.

47573 **SEE ALSO**

47574       *setlocale()*, the Base Definitions volume of IEEE Std 1003.1-200x, <ctype.h>, the Base Definitions  
47575       volume of IEEE Std 1003.1-200x, Chapter 7, Locale

47576 **CHANGE HISTORY**

47577       First released in Issue 1. Derived from Issue 1 of the SVID.

47578 **Issue 6**

47579       Extensions beyond the ISO C standard are now marked.

47580 **NAME**

47581           toupper — transliterate lowercase characters to uppercase

47582 **SYNOPSIS**

47583           #include <ctype.h>

47584           int toupper(int c);

47585 **DESCRIPTION**

47586 cx       The functionality described on this reference page is aligned with the ISO C standard. Any  
47587       conflict between the requirements described here and the ISO C standard is unintentional. This  
47588       volume of IEEE Std 1003.1-200x defers to the ISO C standard.

47589       The *toupper()* function has as a domain a type **int**, the value of which is representable as an  
47590       **unsigned char** or the value of EOF. If the argument has any other value, the behavior is  
47591       undefined. If the argument of *toupper()* represents a lowercase letter, and there exists a  
47592 cx       corresponding uppercase letter (as defined by character type information in the program locale  
47593       category *LC\_CTYPE*), the result shall be the corresponding uppercase letter. All other arguments  
47594       in the domain are returned unchanged. |

47595 **RETURN VALUE**

47596       Upon successful completion, *toupper()* shall return the uppercase letter corresponding to the  
47597       argument passed.

47598 **ERRORS**

47599       No errors are defined.

47600 **EXAMPLES**

47601       None.

47602 **APPLICATION USAGE**

47603       None.

47604 **RATIONALE**

47605       None.

47606 **FUTURE DIRECTIONS**

47607       None.

47608 **SEE ALSO**

47609       *setlocale()*, the Base Definitions volume of IEEE Std 1003.1-200x, <ctype.h>, the Base Definitions  
47610       volume of IEEE Std 1003.1-200x, Chapter 7, Locale

47611 **CHANGE HISTORY**

47612       First released in Issue 1. Derived from Issue 1 of the SVID.

47613 **Issue 6**

47614       Extensions beyond the ISO C standard are now marked.

47615 **NAME**

47616 towctrans — wide-character transliteration

47617 **SYNOPSIS**

47618 #include &lt;wctype.h&gt;

47619 wint\_t towctrans(wint\_t *wc*, wctrans\_t *desc*);47620 **DESCRIPTION**

47621 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
47622 conflict between the requirements described here and the ISO C standard is unintentional. This  
47623 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

47624 The *towctrans()* function shall transliterate the wide-character code *wc* using the mapping  
47625 described by *desc*. The current setting of the *LC\_CTYPE* category should be the same as during  
47626 CX the call to *wctrans()* that returned the value *desc*. If the value of *desc* is invalid (that is, not  
47627 obtained by a call to *wctrans()* or *desc* is invalidated by a subsequent call to *setlocale()* that has  
47628 affected category *LC\_CTYPE*), the result is unspecified.

47629 An application wishing to check for error situations should set *errno* to 0 before calling  
47630 *towctrans()*. If *errno* is non-zero on return, an error has occurred.

47631 **RETURN VALUE**

47632 If successful, the *towctrans()* function shall return the mapped value of *wc* using the mapping  
47633 described by *desc*. Otherwise, it shall return *wc* unchanged.

47634 **ERRORS**47635 The *towctrans()* function may fail if:47636 CX [EINVAL] *desc* contains an invalid transliteration descriptor.47637 **EXAMPLES**

47638 None.

47639 **APPLICATION USAGE**

47640 The strings "tolower" and "toupper" are reserved for the standard mapping names. In the  
47641 table below, the functions in the left column are equivalent to the functions in the right column.

47642 tolower(*wc*) towctrans(*wc*, wctrans("tolower"))47643 toupper(*wc*) towctrans(*wc*, wctrans("toupper"))47644 **RATIONALE**

47645 None.

47646 **FUTURE DIRECTIONS**

47647 None.

47648 **SEE ALSO**

47649 *tolower()*, *toupper()*, *wctrans()*, the Base Definitions volume of IEEE Std 1003.1-200x,  
47650 <wctype.h>

47651 **CHANGE HISTORY**

47652 First released in Issue 5. Derived from ISO/IEC 9899:1990/Amendment 1:1995 (E).

47653 **Issue 6**

47654 Extensions beyond the ISO C standard are now marked.

47655 **NAME**

47656 tolower — transliterate uppercase wide-character code to lowercase

47657 **SYNOPSIS**

47658 #include &lt;wctype.h&gt;

47659 wint\_t tolower(wint\_t wc);

47660 **DESCRIPTION**

47661 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
 47662 conflict between the requirements described here and the ISO C standard is unintentional. This  
 47663 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

47664 The *tolower()* function has as a domain a type **wint\_t**, the value of which the application shall  
 47665 ensure is a character representable as a **wchar\_t**, and a wide-character code corresponding to a  
 47666 valid character in the current locale or the value of WEOF. If the argument has any other value,  
 47667 the behavior is undefined. If the argument of *tolower()* represents an uppercase wide-character  
 47668 code, and there exists a corresponding lowercase wide-character code (as defined by character  
 47669 type information in the program locale category *LC\_CTYPE*), the result shall be the  
 47670 corresponding lowercase wide-character code. All other arguments in the domain are returned  
 47671 unchanged.

47672 **RETURN VALUE**

47673 Upon successful completion, *tolower()* shall return the lowercase letter corresponding to the  
 47674 argument passed; otherwise, it shall return the argument unchanged.

47675 **ERRORS**

47676 No errors are defined.

47677 **EXAMPLES**

47678 None.

47679 **APPLICATION USAGE**

47680 None.

47681 **RATIONALE**

47682 None.

47683 **FUTURE DIRECTIONS**

47684 None.

47685 **SEE ALSO**

47686 *setlocale()*, the Base Definitions volume of IEEE Std 1003.1-200x, <wctype.h>, <wchar.h>, the  
 47687 Base Definitions volume of IEEE Std 1003.1-200x, Chapter 7, Locale

47688 **CHANGE HISTORY**

47689 First released in Issue 4.

47690 **Issue 5**

47691 The following change has been made in this issue for alignment with  
 47692 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 47693 • The SYNOPSIS has been changed to indicate that this function and associated data types are  
 47694 now made visible by inclusion of the <wctype.h> header rather than <wchar.h>.

47695 **Issue 6**

47696 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

47697 **NAME**

47698 towupper — transliterate lowercase wide-character code to uppercase

47699 **SYNOPSIS**

47700 #include &lt;wctype.h&gt;

47701 wint\_t towupper(wint\_t wc);

47702 **DESCRIPTION**

47703 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
47704 conflict between the requirements described here and the ISO C standard is unintentional. This  
47705 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

47706 The *towupper()* function has as a domain a type **wint\_t**, the value of which the application shall  
47707 ensure is a character representable as a **wchar\_t**, and a wide-character code corresponding to a  
47708 valid character in the current locale or the value of WEOF. If the argument has any other value,  
47709 the behavior is undefined. If the argument of *towupper()* represents a lowercase wide-character  
47710 code, and there exists a corresponding uppercase wide-character code (as defined by character  
47711 type information in the program locale category *LC\_CTYPE*), the result shall be the  
47712 corresponding uppercase wide-character code. All other arguments in the domain are returned  
47713 unchanged.

47714 **RETURN VALUE**

47715 Upon successful completion, *towupper()* shall return the uppercase letter corresponding to the  
47716 argument passed. Otherwise, it shall return the argument unchanged.

47717 **ERRORS**

47718 No errors are defined.

47719 **EXAMPLES**

47720 None.

47721 **APPLICATION USAGE**

47722 None.

47723 **RATIONALE**

47724 None.

47725 **FUTURE DIRECTIONS**

47726 None.

47727 **SEE ALSO**

47728 *setlocale()*, the Base Definitions volume of IEEE Std 1003.1-200x, <wctype.h>, <wchar.h>, the  
47729 Base Definitions volume of IEEE Std 1003.1-200x, Chapter 7, Locale

47730 **CHANGE HISTORY**

47731 First released in Issue 4.

47732 **Issue 5**

47733 The following change has been made in this issue for alignment with  
47734 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 47735 • The SYNOPSIS has been changed to indicate that this function and associated data types are  
47736 now made visible by inclusion of the <wctype.h> header rather than <wchar.h>.

47737 **Issue 6**

47738 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

47739 **NAME**

47740 trunc, truncf, trunc1 — round to truncated integer value

47741 **SYNOPSIS**

47742 #include &lt;math.h&gt;

47743 double trunc(double x);

47744 float truncf(float x);

47745 long double trunc1(long double x);

47746 **DESCRIPTION**47747 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
47748 conflict between the requirements described here and the ISO C standard is unintentional. This  
47749 volume of IEEE Std 1003.1-200x defers to the ISO C standard.47750 These functions shall round their argument to the integer value, in floating format, nearest to but  
47751 no larger in magnitude than the argument.47752 **RETURN VALUE**

47753 Upon successful completion, these functions shall return the truncated integer value.

47754 **MX** If  $x$  is NaN, a NaN shall be returned.47755 If  $x$  is  $\pm 0$ , or  $\pm \text{Inf}$ ,  $x$  shall be returned.47756 **ERRORS**

47757 No errors are defined.

47758 **EXAMPLES**

47759 None.

47760 **APPLICATION USAGE**

47761 None.

47762 **RATIONALE**

47763 None.

47764 **FUTURE DIRECTIONS**

47765 None.

47766 **SEE ALSO**

47767 The Base Definitions volume of IEEE Std 1003.1-200x, &lt;math.h&gt;

47768 **CHANGE HISTORY**

47769 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

47770 **NAME**

47771 truncate — truncate a file to a specified length

47772 **SYNOPSIS**47773 XSI `#include <unistd.h>`47774 `int truncate(const char *path, off_t length);`

47775

47776 **DESCRIPTION**47777 The `truncate()` function shall cause the regular file named by *path* to have a size which shall be  
47778 equal to *length* bytes.47779 If the file previously was larger than *length*, the extra data is discarded. If the file was previously  
47780 shorter than *length*, its size is increased, and the extended area appears as if it were zero-filled.

47781 The application shall ensure that the process has write permission for the file.

47782 If the request would cause the file size to exceed the soft file size limit for the process, the  
47783 request shall fail and the implementation shall generate the SIGXFSZ signal for the process.47784 This function shall not modify the file offset for any open file descriptions associated with the  
47785 file. Upon successful completion, if the file size is changed, this function shall mark for update  
47786 the *st\_ctime* and *st\_mtime* fields of the file, and the S\_ISUID and S\_ISGID bits of the file mode  
47787 may be cleared.47788 **RETURN VALUE**47789 Upon successful completion, `truncate()` shall return 0. Otherwise, `-1` shall be returned, and *errno*  
47790 set to indicate the error.47791 **ERRORS**47792 The `truncate()` function shall fail if:

47793 [EINTR] A signal was caught during execution.

47794 [EINVAL] The *length* argument was less than 0.

47795 [EFBIG] or [EINVAL]

47796 The *length* argument was greater than the maximum file size.

47797 [EIO] An I/O error occurred while reading from or writing to a file system.

47798 [EACCES] A component of the path prefix denies search permission, or write permission  
47799 is denied on the file.

47800 [EISDIR] The named file is a directory.

47801 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*  
47802 argument.

47803 [ENAMETOOLONG]

47804 The length of the *path* argument exceeds {PATH\_MAX} or a pathname |  
47805 component is longer than {NAME\_MAX}. |47806 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.47807 [ENOTDIR] A component of the path prefix of *path* is not a directory.

47808 [EROFS] The named file resides on a read-only file system.

47809 The `truncate()` function may fail if:

- 47810 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
47811 resolution of the *path* argument.
- 47812 [ENAMETOOLONG]  
47813 Pathname resolution of a symbolic link produced an intermediate result |  
47814 whose length exceeds {PATH\_MAX}.
- 47815 **EXAMPLES**  
47816 None.
- 47817 **APPLICATION USAGE**  
47818 None.
- 47819 **RATIONALE**  
47820 None.
- 47821 **FUTURE DIRECTIONS**  
47822 None.
- 47823 **SEE ALSO**  
47824 *open()*, the Base Definitions volume of IEEE Std 1003.1-200x, <**unistd.h**>
- 47825 **CHANGE HISTORY**  
47826 First released in Issue 4, Version 2.
- 47827 **Issue 5**  
47828 Moved from X/OPEN UNIX extension to BASE.  
47829 Large File Summit extensions are added.
- 47830 **Issue 6**  
47831 This reference page is split out from the *truncate()* reference page.  
47832 The DESCRIPTION is updated to avoid use of the term “must” for application requirements. |  
47833 The wording of the mandatory [ELOOP] error condition is updated, and a second optional  
47834 [ELOOP] error condition is added.

47835 **NAME**

47836           truncf, trunc1 — round to truncated integer value

47837 **SYNOPSIS**

47838           #include &lt;math.h&gt;

47839           float truncf(float x);

47840           long double trunc1(long double x);

47841 **DESCRIPTION**47842           Refer to *trunc()*.

47843 **NAME**

47844           tsearch — search a binary search tree

47845 **SYNOPSIS**

47846 XSI       #include &lt;search.h&gt;

47847           void \*tsearch(const void \*key, void \*\*rootp,  
47848                        int (\*compar)(const void \*, const void \*));

47849

47850 **DESCRIPTION**47851           Refer to *tdelete*().

47852 **NAME**

47853 `ttyname`, `ttyname_r` — find pathname of a terminal

47854 **SYNOPSIS**

47855 `#include <unistd.h>`

47856 `char *ttyname(int fd);`

47857 TSF `int ttyname_r(int fd, char *name, size_t namesize);`

47858

47859 **DESCRIPTION**

47860 The `ttyname()` function shall return a pointer to a string containing a null-terminated pathname  
47861 of the terminal associated with file descriptor *fd*. The return value may point to static data  
47862 whose content is overwritten by each call.

47863 The `ttyname()` function need not be reentrant. A function that is not required to be reentrant is  
47864 not required to be thread-safe.

47865 TSF The `ttyname_r()` function shall store the null-terminated pathname of the terminal associated  
47866 with the file descriptor *fd* in the character array referenced by *name*. The array is *namesize*  
47867 characters long and should have space for the name and the terminating null character. The  
47868 maximum length of the terminal name shall be {TTY\_NAME\_MAX}.

47869 **RETURN VALUE**

47870 Upon successful completion, `ttyname()` shall return a pointer to a string. Otherwise, a null  
47871 pointer shall be returned and *errno* set to indicate the error.

47872 TSF If successful, the `ttyname_r()` function shall return zero. Otherwise, an error number shall be  
47873 returned to indicate the error.

47874 **ERRORS**

47875 The `ttyname()` function may fail if:

47876 [EBADF] The *fd* argument is not a valid file descriptor.

47877 [ENOTTY] The *fd* argument does not refer to a terminal.

47878 The `ttyname_r()` function may fail if:

47879 TSF [EBADF] The *fd* argument is not a valid file descriptor.

47880 TSF [ENOTTY] The *fd* argument does not refer to a terminal.

47881 TSF [ERANGE] The value of *namesize* is smaller than the length of the string to be returned  
47882 including the terminating null character.

47883 **EXAMPLES**

47884 None.

47885 **APPLICATION USAGE**

47886 None.

47887 **RATIONALE**

47888 The term *terminal* is used instead of the historical term *terminal device* in order to avoid a  
47889 reference to an undefined term.

47890 The thread-safe version places the terminal name in a user-supplied buffer and returns a non-  
47891 zero value if it fails. The non-thread-safe version may return the name in a static data area that  
47892 may be overwritten by each call.

47893 **FUTURE DIRECTIONS**

47894       None.

47895 **SEE ALSO**

47896       The Base Definitions volume of IEEE Std 1003.1-200x, &lt;unistd.h&gt;

47897 **CHANGE HISTORY**

47898       First released in Issue 1. Derived from Issue 1 of the SVID.

47899 **Issue 5**47900       The *ttyname\_r()* function is included for alignment with the POSIX Threads Extension.47901       A note indicating that the *ttyname()* function need not be reentrant is added to the  
47902       DESCRIPTION.47903 **Issue 6**47904       The *ttyname\_r()* function is marked as part of the Thread-Safe Functions option.47905       The following new requirements on POSIX implementations derive from alignment with the  
47906       Single UNIX Specification:

- 47907
- The statement that *errno* is set on error is added.
  - The [EBADF] and [ENOTTY] optional error conditions are added.
- 47908

47909 **NAME**

47910 twalk — traverse a binary search tree

47911 **SYNOPSIS**

47912 XSI `#include <search.h>`

47913 `void twalk(const void *root,`  
47914 `void (*action)(const void *, VISIT, int ));`

47915

47916 **DESCRIPTION**

47917 Refer to *tdelete()*.

47918 **NAME**

47919           tzname — timezone strings

47920 **SYNOPSIS**

47921 cx       #include &lt;time.h&gt;

47922           extern char \*tzname[2];

47923

47924 **DESCRIPTION**47925           Refer to *tzset()*.

47926 **NAME**

47927 daylight, timezone, tzname, tzset — set timezone conversion information

47928 **SYNOPSIS**

47929 #include &lt;time.h&gt;

47930 XSI extern int daylight;

47931 extern long timezone;

47932 CX extern char \*tzname[2];

47933 void tzset(void);

47934

47935 **DESCRIPTION**

47936 The *tzset()* function shall use the value of the environment variable *TZ* to set time conversion  
 47937 information used by *ctime()*, *localtime()*, *mktime()*, and *strftime()*. If *TZ* is absent from the  
 47938 environment, implementation-defined default timezone information shall be used.

47939 The *tzset()* function shall set the external variable *tzname* as follows:

47940 tzname[0] = "std";

47941 tzname[1] = "dst";

47942 where *std* and *dst* are as described in the Base Definitions volume of IEEE Std 1003.1-200x,  
 47943 Chapter 8, Environment Variables.

47944 XSI The *tzset()* function also shall set the external variable *daylight* to 0 if Daylight Savings Time  
 47945 conversions should never be applied for the timezone in use; otherwise, non-zero. The external  
 47946 variable *timezone* shall be set to the difference, in seconds, between Coordinated Universal Time  
 47947 (UTC) and local standard time.

47948 **RETURN VALUE**47949 The *tzset()* function shall not return a value.47950 **ERRORS**

47951 No errors are defined.

47952 **EXAMPLES**

47953 Example TZ variables and their timezone differences are given in the table below:

47954

	<b>TZ</b>	<i>timezone</i>
47956	EST5EDT	5*60*60
47957	GMT0	0*60*60
47958	JST-9	-9*60*60
47959	MET-1MEST	-1*60*60
47960	MST7MDT	7*60*60
47961	PST8PDT	8*60*60

47962 **APPLICATION USAGE**

47963 None.

47964 **RATIONALE**

47965 None.

47966 **FUTURE DIRECTIONS**

47967 None.

47968 **SEE ALSO**

47969 *ctime()*, *localtime()*, *mktime()*, *strftime()*, the Base Definitions volume of IEEE Std 1003.1-200x,  
47970 **<time.h>**

47971 **CHANGE HISTORY**

47972 First released in Issue 1. Derived from Issue 1 of the SVID. |

47973 **Issue 6** |

47974 The example is corrected. |

47975 **NAME**

47976       ualarm — set the interval timer

47977 **SYNOPSIS**

47978 OB XSI   #include &lt;unistd.h&gt;

47979       useconds\_t ualarm(useconds\_t *useconds*, useconds\_t *interval*);

47980

47981 **DESCRIPTION**

47982       The *ualarm()* function shall cause the SIGALRM signal to be generated for the calling process  
47983       after the number of realtime microseconds specified by the *useconds* argument has elapsed.  
47984       When the *interval* argument is non-zero, repeated timeout notification occurs with a period in  
47985       microseconds specified by the *interval* argument. If the notification signal, SIGALRM, is not  
47986       caught or ignored, the calling process is terminated.

47987       Implementations may place limitations on the granularity of timer values. For each interval  
47988       timer, if the requested timer value requires a finer granularity than the implementation supports,  
47989       the actual timer value shall be rounded up to the next supported value.

47990       Interactions between *ualarm()* and any of the following are unspecified:

47991            *alarm()*  
47992            *nanosleep()*  
47993            *setitimer()*  
47994            *timer\_create()*  
47995            *timer\_delete()*  
47996            *timer\_getoverrun()*  
47997            *timer\_gettime()*  
47998            *timer\_settime()*  
47999            *sleep()*

48000 **RETURN VALUE**

48001       The *ualarm()* function shall return the number of microseconds remaining from the previous  
48002       *ualarm()* call. If no timeouts are pending or if *ualarm()* has not previously been called, *ualarm()*  
48003       shall return 0.

48004 **ERRORS**

48005       No errors are defined.

48006 **EXAMPLES**

48007       None.

48008 **APPLICATION USAGE**

48009       Applications are recommended to use *nanosleep()* if the Timers option is supported, or  
48010       *setitimer()*, *timer\_create()*, *timer\_delete()*, *timer\_getoverrun()*, *timer\_gettime()*, or *timer\_settime()*  
48011       instead of this function.

48012 **RATIONALE**

48013       None.

48014 **FUTURE DIRECTIONS**

48015       None.

48016 **SEE ALSO**

48017       *alarm()*, *nanosleep()*, *setitimer()*, *sleep()*, *timer\_create()*, *timer\_delete()*, *timer\_getoverrun()*, the Base  
48018       Definitions volume of IEEE Std 1003.1-200x, <unistd.h>

48019 **CHANGE HISTORY**

48020 First released in Issue 4, Version 2.

48021 **Issue 5**

48022 Moved from X/OPEN UNIX extension to BASE.

48023 **Issue 6**

48024 This function is marked obsolescent.

48025 **NAME**

48026 ulimit — get and set process limits

48027 **SYNOPSIS**48028 XSI `#include <ulimit.h>`48029 `long ulimit(int cmd, ...);`

48030

48031 **DESCRIPTION**

48032 The *ulimit()* function shall control process limits. The process limits that can be controlled by  
 48033 this function include the maximum size of a single file that can be written (this is equivalent to  
 48034 using *setrlimit()* with `RLIMIT_FSIZE`). The *cmd* values, defined in `<ulimit.h>` include:

48035 `UL_GETFSIZE` Return the file size limit (`RLIMIT_FSIZE`) of the process. The limit shall be in  
 48036 units of 512-byte blocks and shall be inherited by child processes. Files of any  
 48037 size can be read. The return value shall be the integer part of the soft file size  
 48038 limit divided by 512. If the result cannot be represented as a **long**, the result is  
 48039 unspecified.

48040 `UL_SETFSIZE` Set the file size limit for output operations of the process to the value of the  
 48041 second argument, taken as a **long**, multiplied by 512. If the result would  
 48042 overflow an `rlim_t`, the actual value set is unspecified. Any process may  
 48043 decrease its own limit, but only a process with appropriate privileges may  
 48044 increase the limit. The return value shall be the integer part of the new file size  
 48045 limit divided by 512.

48046 The *ulimit()* function shall not change the setting of *errno* if successful.

48047 As all return values are permissible in a successful situation, an application wishing to check for  
 48048 error situations should set *errno* to 0, then call *ulimit()*, and, if it returns `-1`, check to see if *errno* is  
 48049 non-zero.

48050 **RETURN VALUE**

48051 Upon successful completion, *ulimit()* shall return the value of the requested limit. Otherwise, `-1`  
 48052 shall be returned and *errno* set to indicate the error.

48053 **ERRORS**

48054 The *ulimit()* function shall fail and the limit shall be unchanged if:

48055 `[EINVAL]` The *cmd* argument is not valid.

48056 `[EPERM]` A process not having appropriate privileges attempts to increase its file size  
 48057 limit.

48058 **EXAMPLES**

48059 None.

48060 **APPLICATION USAGE**

48061 None.

48062 **RATIONALE**

48063 None.

48064 **FUTURE DIRECTIONS**

48065 None.

48066 **SEE ALSO**

48067 *getrlimit()*, *setrlimit()*, *write()*, the Base Definitions volume of IEEE Std 1003.1-200x, <ulimit.h>

48068 **CHANGE HISTORY**

48069 First released in Issue 1. Derived from Issue 1 of the SVID.

48070 **Issue 5**

48071 In the description of UL\_SETFSIZE, the text is corrected to refer to **rlim\_t** rather than the spurious **rlimit\_t**.

48073 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.

48074 **NAME**

48075 umask — set and get file mode creation mask

48076 **SYNOPSIS**

48077 #include <sys/stat.h>

48078 mode\_t umask(mode\_t *cmask*);

48079 **DESCRIPTION**

48080 The *umask()* function shall set the process' file mode creation mask to *cmask* and return the  
48081 previous value of the mask. Only the file permission bits of *cmask* (see <sys/stat.h>) are used; the  
48082 meaning of the other bits is implementation-defined.

48083 The process' file mode creation mask is used during *open()*, *creat()*, *mkdir()*, and *mkfifo()* to turn  
48084 off permission bits in the *mode* argument supplied. Bit positions that are set in *cmask* are cleared  
48085 in the mode of the created file.

48086 **RETURN VALUE**

48087 The file permission bits in the value returned by *umask()* shall be the previous value of the file  
48088 mode creation mask. The state of any other bits in that value is unspecified, except that a  
48089 subsequent call to *umask()* with the returned value as *cmask* shall leave the state of the mask the  
48090 same as its state before the first call, including any unspecified use of those bits.

48091 **ERRORS**

48092 No errors are defined.

48093 **EXAMPLES**

48094 None.

48095 **APPLICATION USAGE**

48096 None.

48097 **RATIONALE**

48098 Unsigned argument and return types for *umask()* were proposed. The return type and the  
48099 argument were both changed to **mode\_t**.

48100 Historical implementations have made use of additional bits in *cmask* for their implementation-  
48101 defined purposes. The addition of the text that the meaning of other bits of the field is  
48102 implementation-defined permits these implementations to conform to this volume of  
48103 IEEE Std 1003.1-200x.

48104 **FUTURE DIRECTIONS**

48105 None.

48106 **SEE ALSO**

48107 *creat()*, *mkdir()*, *mkfifo()*, *open()*, the Base Definitions volume of IEEE Std 1003.1-200x,  
48108 <sys/stat.h>, <sys/types.h>

48109 **CHANGE HISTORY**

48110 First released in Issue 1. Derived from Issue 1 of the SVID.

48111 **Issue 6**

48112 In the SYNOPSIS, the optional include of the <sys/types.h> header is removed. |

- 48113 The following new requirements on POSIX implementations derive from alignment with the |  
48114 Single UNIX Specification:
- 48115 • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was  
48116 required for conforming implementations of previous POSIX specifications, it was not  
48117 required for UNIX applications.

48118 **NAME**

48119            **uname** — get name of current system

48120 **SYNOPSIS**

48121            #include <sys/utsname.h>

48122            int uname(struct utsname \*name);

48123 **DESCRIPTION**

48124            The *uname()* function shall store information identifying the current system in the structure pointed to by *name*.

48126            The *uname()* function uses the **utsname** structure defined in <sys/utsname.h>.

48127            The *uname()* function shall return a string naming the current system in the character array *sysname*. Similarly, *nodename* shall contain the name of this node within an implementation-defined communications network. The arrays *release* and *version* shall further identify the operating system. The array *machine* shall contain a name that identifies the hardware that the system is running on.

48132            The format of each member is implementation-defined.

48133 **RETURN VALUE**

48134            Upon successful completion, a non-negative value shall be returned. Otherwise, -1 shall be returned and *errno* set to indicate the error.

48136 **ERRORS**

48137            No errors are defined.

48138 **EXAMPLES**

48139            None.

48140 **APPLICATION USAGE**

48141            The inclusion of the *nodename* member in this structure does not imply that it is sufficient information for interfacing to communications networks.

48143 **RATIONALE**

48144            The values of the structure members are not constrained to have any relation to the version of this volume of IEEE Std 1003.1-200x implemented in the operating system. An application should instead depend on `_POSIX_VERSION` and related constants defined in <unistd.h>.

48147            This volume of IEEE Std 1003.1-200x does not define the sizes of the members of the structure and permits them to be of different sizes, although most implementations define them all to be the same size: eight bytes plus one byte for the string terminator. That size for *nodename* is not enough for use with many networks.

48151            The *uname()* function originated in System III, System V, and related implementations, and it does not exist in Version 7 or 4.3 BSD. The values it returns are set at system compile time in those historical implementations.

48154            4.3 BSD has *gethostname()* and *gethostid()*, which return a symbolic name and a numeric value, respectively. There are related *sethostname()* and *sethostid()* functions that are used to set the values the other two functions return. The former functions are included in this specification, the latter are not.

48158 **FUTURE DIRECTIONS**

48159            None.

48160 **SEE ALSO**

48161           The Base Definitions volume of IEEE Std 1003.1-200x, <sys/utsname.h>

48162 **CHANGE HISTORY**

48163           First released in Issue 1. Derived from Issue 1 of the SVID.

48164 **NAME**

48165 ungetc — push byte back into input stream

48166 **SYNOPSIS**

48167 #include &lt;stdio.h&gt;

48168 int ungetc(int *c*, FILE \**stream*);48169 **DESCRIPTION**

48170 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
48171 conflict between the requirements described here and the ISO C standard is unintentional. This  
48172 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

48173 The *ungetc()* function shall push the byte specified by *c* (converted to an **unsigned char**) back  
48174 onto the input stream pointed to by *stream*. The pushed-back bytes shall be returned by  
48175 subsequent reads on that stream in the reverse order of their pushing. A successful intervening  
48176 call (with the stream pointed to by *stream*) to a file-positioning function (*fseek()*, *fsetpos()*, or  
48177 *rewind()*) shall discard any pushed-back bytes for the stream. The external storage  
48178 corresponding to the stream shall be unchanged.

48179 One byte of push-back shall be provided. If *ungetc()* is called too many times on the same stream  
48180 without an intervening read or file-positioning operation on that stream, the operation may fail.

48181 If the value of *c* equals that of the macro EOF, the operation shall fail and the input stream shall  
48182 be left unchanged.

48183 A successful call to *ungetc()* shall clear the end-of-file indicator for the stream. The value of the  
48184 file-position indicator for the stream after reading or discarding all pushed-back bytes shall be  
48185 the same as it was before the bytes were pushed back. The file-position indicator is decremented  
48186 by each successful call to *ungetc()*; if its value was 0 before a call, its value is unspecified after  
48187 the call.

48188 **RETURN VALUE**

48189 Upon successful completion, *ungetc()* shall return the byte pushed back after conversion.  
48190 Otherwise, it shall return EOF.

48191 **ERRORS**

48192 No errors are defined.

48193 **EXAMPLES**

48194 None.

48195 **APPLICATION USAGE**

48196 None.

48197 **RATIONALE**

48198 None.

48199 **FUTURE DIRECTIONS**

48200 None.

48201 **SEE ALSO**

48202 *fseek()*, *getc()*, *fsetpos()*, *read()*, *rewind()*, *setbuf()*, the Base Definitions volume of  
48203 IEEE Std 1003.1-200x, <stdio.h>

48204 **CHANGE HISTORY**

48205 First released in Issue 1. Derived from Issue 1 of the SVID.

48206 **NAME**

48207 ungetwc — push wide-character code back into input stream

48208 **SYNOPSIS**

48209 #include &lt;stdio.h&gt;

48210 #include &lt;wchar.h&gt;

48211 wint\_t ungetwc(wint\_t *wc*, FILE \**stream*);48212 **DESCRIPTION**

48213 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 48214 conflict between the requirements described here and the ISO C standard is unintentional. This  
 48215 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

48216 The *ungetwc()* function shall push the character corresponding to the wide-character code  
 48217 specified by *wc* back onto the input stream pointed to by *stream*. The pushed-back characters  
 48218 shall be returned by subsequent reads on that stream in the reverse order of their pushing. A  
 48219 successful intervening call (with the stream pointed to by *stream*) to a file-positioning function  
 48220 (*fseek()*, *fsetpos()*, or *rewind()*) discards any pushed-back characters for the stream. The external  
 48221 storage corresponding to the stream is unchanged.

48222 At least one character of push-back shall be provided. If *ungetwc()* is called too many times on  
 48223 the same stream without an intervening read or file-positioning operation on that stream, the  
 48224 operation may fail.

48225 If the value of *wc* equals that of the macro WEOF, the operation shall fail and the input stream  
 48226 shall be left unchanged.

48227 A successful call to *ungetwc()* shall clear the end-of-file indicator for the stream. The value of the  
 48228 file-position indicator for the stream after reading or discarding all pushed-back characters shall  
 48229 be the same as it was before the characters were pushed back. The file-position indicator is  
 48230 decremented (by one or more) by each successful call to *ungetwc()*; if its value was 0 before a  
 48231 call, its value is unspecified after the call.

48232 **RETURN VALUE**

48233 Upon successful completion, *ungetwc()* shall return the wide-character code corresponding to  
 48234 the pushed-back character. Otherwise, it shall return WEOF.

48235 **ERRORS**48236 The *ungetwc()* function may fail if:

48237 **CX** [EILSEQ] An invalid character sequence is detected, or a wide-character code does not  
 48238 correspond to a valid character.

48239 **EXAMPLES**

48240 None.

48241 **APPLICATION USAGE**

48242 None.

48243 **RATIONALE**

48244 None.

48245 **FUTURE DIRECTIONS**

48246 None.

48247 **SEE ALSO**

48248 *fseek()*, *fsetpos()*, *read()*, *rewind()*, *setbuf()*, the Base Definitions volume of IEEE Std 1003.1-200x,  
48249 `<stdio.h>`, `<wchar.h>`

48250 **CHANGE HISTORY**

48251 First released in Issue 4. Derived from the MSE working draft.

48252 **Issue 5**

48253 The Optional Header (OH) marking is removed from `<stdio.h>`. |

48254 **Issue 6**

48255 The [EILSEQ] optional error condition is marked CX. |

48256 **NAME**

48257 unlink — remove a directory entry

48258 **SYNOPSIS**

48259 #include &lt;unistd.h&gt;

48260 int unlink(const char \*path);

48261 **DESCRIPTION**

48262 The *unlink()* function shall remove a link to a file. If *path* names a symbolic link, *unlink()* shall  
 48263 remove the symbolic link named by *path* and shall not affect any file or directory named by the  
 48264 contents of the symbolic link. Otherwise, *unlink()* shall remove the link named by the pathname  
 48265 pointed to by *path* and shall decrement the link count of the file referenced by the link.

48266 When the file's link count becomes 0 and no process has the file open, the space occupied by the  
 48267 file shall be freed and the file shall no longer be accessible. If one or more processes have the file  
 48268 open when the last link is removed, the link shall be removed before *unlink()* returns, but the  
 48269 removal of the file contents shall be postponed until all references to the file are closed.

48270 The *path* argument shall not name a directory unless the process has appropriate privileges and  
 48271 the implementation supports using *unlink()* on directories.

48272 Upon successful completion, *unlink()* shall mark for update the *st\_ctime* and *st\_mtime* fields of  
 48273 the parent directory. Also, if the file's link count is not 0, the *st\_ctime* field of the file shall be  
 48274 marked for update.

48275 **RETURN VALUE**

48276 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to  
 48277 indicate the error. If -1 is returned, the named file shall not be changed.

48278 **ERRORS**48279 The *unlink()* function shall fail and shall not unlink the file if:

48280 [EACCES] Search permission is denied for a component of the path prefix, or write  
 48281 permission is denied on the directory containing the directory entry to be  
 48282 removed.

48283 [EBUSY] The file named by the *path* argument cannot be unlinked because it is being  
 48284 used by the system or another process and the implementation considers this  
 48285 an error.

48286 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*  
 48287 argument.

48288 [ENAMETOOLONG] The length of the *path* argument exceeds {PATH\_MAX} or a pathname  
 48289 component is longer than {NAME\_MAX}.

48291 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

48292 [ENOTDIR] A component of the path prefix is not a directory.

48293 [EPERM] The file named by *path* is a directory, and either the calling process does not  
 48294 have appropriate privileges, or the implementation prohibits using *unlink()*  
 48295 on directories.

48296 XSI [EPERM] or [EACCES]

48297 The S\_ISVTX flag is set on the directory containing the file referred to by the  
 48298 *path* argument and the caller is not the file owner, nor is the caller the  
 48299 directory owner, nor does the caller have appropriate privileges.

48300 [EROFS] The directory entry to be unlinked is part of a read-only file system.

48301 The *unlink()* function may fail and not unlink the file if:

48302 XSI [EBUSY] The file named by *path* is a named STREAM.

48303 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
48304 resolution of the *path* argument.

48305 [ENAMETOOLONG]  
48306 As a result of encountering a symbolic link in resolution of the *path* argument, |  
48307 the length of the substituted pathname string exceeded {PATH\_MAX}. |

48308 [ETXTBSY] The entry to be unlinked is the last directory entry to a pure procedure (shared  
48309 text) file that is being executed.

48310 **EXAMPLES**

48311 **Removing a Link to a File**

48312 The following example shows how to remove a link to a file named `/home/cnd/mod1` by  
48313 removing the entry named `/modules/pass1`.

```
48314 #include <unistd.h>
48315 char *path = "/modules/pass1";
48316 int status;
48317 ...
48318 status = unlink(path);
```

48319 **Checking for an Error**

48320 The following example fragment creates a temporary password lock file named **LOCKFILE**,  
48321 which is defined as `/etc/ptmp`, and gets a file descriptor for it. If the file cannot be opened for  
48322 writing, *unlink()* is used to remove the link between the file descriptor and **LOCKFILE**.

```
48323 #include <sys/types.h>
48324 #include <stdio.h>
48325 #include <fcntl.h>
48326 #include <errno.h>
48327 #include <unistd.h>
48328 #include <sys/stat.h>
48329 #define LOCKFILE "/etc/ptmp"
48330 int pfd; /* Integer for file descriptor returned by open call. */
48331 FILE *fpfd; /* File pointer for use in putpwent(). */
48332 ...
48333 /* Open password Lock file. If it exists, this is an error. */
48334 if ((pfd = open(LOCKFILE, O_WRONLY | O_CREAT | O_EXCL, S_IRUSR
48335 | S_IWUSR | S_IRGRP | S_IROTH)) == -1) {
48336     fprintf(stderr, "Cannot open /etc/ptmp. Try again later.\n");
48337     exit(1);
48338 }
48339 /* Lock file created, proceed with fdopen of lock file so that
48340 putpwent() can be used.
48341 */
48342 if ((fpfd = fdopen(pfd, "w")) == NULL) {
```

```

48343         close(pfd);
48344         unlink(LOCKFILE);
48345         exit(1);
48346     }

```

### 48347 **Replacing Files**

48348 The following example fragment uses *unlink()* to discard links to files, so that they can be  
48349 replaced with new versions of the files. The first call remove the link to **LOCKFILE** if an error  
48350 occurs. Successive calls remove the links to **SAVEFILE** and **PASSWDFILE** so that new links can  
48351 be created, then removes the link to **LOCKFILE** when it is no longer needed.

```

48352     #include <sys/types.h>
48353     #include <stdio.h>
48354     #include <fcntl.h>
48355     #include <errno.h>
48356     #include <unistd.h>
48357     #include <sys/stat.h>

48358     #define LOCKFILE "/etc/ptmp"
48359     #define PASSWDFILE "/etc/passwd"
48360     #define SAVEFILE "/etc/opasswd"
48361     ...
48362     /* If no change was made, assume error and leave passwd unchanged. */
48363     if (!valid_change) {
48364         fprintf(stderr, "Could not change password for user %s\n", user);
48365         unlink(LOCKFILE);
48366         exit(1);
48367     }

48368     /* Change permissions on new password file. */
48369     chmod(LOCKFILE, S_IRUSR | S_IRGRP | S_IROTH);

48370     /* Remove saved password file. */
48371     unlink(SAVEFILE);

48372     /* Save current password file. */
48373     link(PASSWDFILE, SAVEFILE);

48374     /* Remove current password file. */
48375     unlink(PASSWDFILE);

48376     /* Save new password file as current password file. */
48377     link(LOCKFILE, PASSWDFILE);

48378     /* Remove lock file. */
48379     unlink(LOCKFILE);

48380     exit(0);

```

### 48381 **APPLICATION USAGE**

48382 Applications should use *rmdir()* to remove a directory.

### 48383 **RATIONALE**

48384 Unlinking a directory is restricted to the superuser in many historical implementations for  
48385 reasons given in *link()* (see also *rename()*).

48386 The meaning of [EBUSY] in historical implementations is “mount point busy”. Since this volume  
 48387 of IEEE Std 1003.1-200x does not cover the system administration concepts of mounting and  
 48388 unmounting, the description of the error was changed to “resource busy”. (This meaning is used  
 48389 by some device drivers when a second process tries to open an exclusive use device.) The  
 48390 wording is also intended to allow implementations to refuse to remove a directory if it is the  
 48391 root or current working directory of any process.

48392 **FUTURE DIRECTIONS**

48393 None.

48394 **SEE ALSO**

48395 *close()*, *link()*, *remove()*, *rmdir()*, the Base Definitions volume of IEEE Std 1003.1-200x,  
 48396 <unistd.h>

48397 **CHANGE HISTORY**

48398 First released in Issue 1. Derived from Issue 1 of the SVID.

48399 **Issue 5**

48400 The [EBUSY] error is added to the “may fail” part of the ERRORS section.

48401 **Issue 6**

48402 The following new requirements on POSIX implementations derive from alignment with the  
 48403 Single UNIX Specification:

- 48404 • In the DESCRIPTION, the effect is specified if *path* specifies a symbolic link.
- 48405 • The [ELOOP] mandatory error condition is added.
- 48406 • A second [ENAMETOOLONG] is added as an optional error condition.
- 48407 • The [ETXTBSY] optional error condition is added.

48408 The following changes were made to align with the IEEE P1003.1a draft standard:

- 48409 • The [ELOOP] optional error condition is added.

48410 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

48411 **NAME**

48412 unlockpt — unlock a pseudo-terminal master/slave pair

48413 **SYNOPSIS**48414 XSI `#include <stdlib.h>`48415 `int unlockpt(int fildev);`

48416

48417 **DESCRIPTION**48418 The *unlockpt()* function shall unlock the slave pseudo-terminal device associated with the  
48419 master to which *fildev* refers.48420 Conforming applications shall ensure that they call *unlockpt()* before opening the slave side of a  
48421 pseudo-terminal device.48422 **RETURN VALUE**48423 Upon successful completion, *unlockpt()* shall return 0. Otherwise, it shall return -1 and set *errno*  
48424 to indicate the error.48425 **ERRORS**48426 The *unlockpt()* function may fail if:48427 [EBADF] The *fildev* argument is not a file descriptor open for writing.48428 [EINVAL] The *fildev* argument is not associated with a master pseudo-terminal device.48429 **EXAMPLES**

48430 None.

48431 **APPLICATION USAGE**

48432 None.

48433 **RATIONALE**

48434 None.

48435 **FUTURE DIRECTIONS**

48436 None.

48437 **SEE ALSO**48438 *grantpt()*, *open()*, *ptsname()*, the Base Definitions volume of IEEE Std 1003.1-200x, `<stdlib.h>`48439 **CHANGE HISTORY**

48440 First released in Issue 4, Version 2.

48441 **Issue 5**

48442 Moved from X/OPEN UNIX extension to BASE.

48443 **Issue 6**

48444 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

48445 **NAME**

48446           unsetenv — remove environment variable

48447 **SYNOPSIS**48448 `cx`       #include <stdlib.h>

48449           int unsetenv(const char \*name);

48450

48451 **DESCRIPTION**

48452       The *unsetenv()* function shall remove an environment variable from the environment of the  
48453       calling process. The *name* argument points to a string, which is the name of the variable to be  
48454       removed. The named argument shall not contain an '=' character. If the named variable does  
48455       not exist in the current environment, the environment shall be unchanged and the function is  
48456       considered to have completed successfully.

48457       If the application modifies *environ* or the pointers to which it points, the behavior of *unsetenv()* is  
48458       undefined. The *unsetenv()* function shall update the list of pointers to which *environ* points.

48459       The *unsetenv()* function need not be reentrant. A function that is not required to be reentrant is  
48460       not required to be thread-safe.

48461 **RETURN VALUE**

48462       Upon successful completion, zero shall be returned. Otherwise, -1 shall be returned, *errno* set to  
48463       indicate the error, and the environment shall be unchanged.

48464 **ERRORS**48465       The *unsetenv()* function shall fail if:

48466       [EINVAL]       The *name* argument is a null pointer, points to an empty string, or points to a  
48467       string containing an '=' character.

48468 **EXAMPLES**

48469       None.

48470 **APPLICATION USAGE**

48471       None.

48472 **RATIONALE**48473       Refer to the RATIONALE section in *setenv()*.48474 **FUTURE DIRECTIONS**

48475       None.

48476 **SEE ALSO**

48477       *getenv()*, *setenv()*, the Base Definitions volume of IEEE Std 1003.1-200x, <stdlib.h>,  
48478       <sys/types.h>, <unistd.h>

48479 **CHANGE HISTORY**

48480       First released in Issue 6. Derived from the IEEE P1003.1a draft standard.

48481 **NAME**

48482            usleep — suspend execution for an interval

48483 **SYNOPSIS**

48484 OB XSI    #include &lt;unistd.h&gt;

48485            int usleep(useconds\_t useconds);

48486

48487 **DESCRIPTION**

48488        The *usleep()* function shall cause the calling thread to be suspended from execution until either  
 48489        the number of realtime microseconds specified by the argument *useconds* has elapsed or a signal  
 48490        is delivered to the calling thread and its action is to invoke a signal-catching function or to  
 48491        terminate the process. The suspension time may be longer than requested due to the scheduling  
 48492        of other activity by the system.

48493        The *useconds* argument shall be less than one million. If the value of *useconds* is 0, then the call  
 48494        has no effect.

48495        If a SIGALRM signal is generated for the calling process during execution of *usleep()* and if the  
 48496        SIGALRM signal is being ignored or blocked from delivery, it is unspecified whether *usleep()*  
 48497        returns when the SIGALRM signal is scheduled. If the signal is being blocked, it is also  
 48498        unspecified whether it remains pending after *usleep()* returns or it is discarded.

48499        If a SIGALRM signal is generated for the calling process during execution of *usleep()*, except as a  
 48500        result of a prior call to *alarm()*, and if the SIGALRM signal is not being ignored or blocked from  
 48501        delivery, it is unspecified whether that signal has any effect other than causing *usleep()* to return.

48502        If a signal-catching function interrupts *usleep()* and examines or changes either the time a  
 48503        SIGALRM is scheduled to be generated, the action associated with the SIGALRM signal, or  
 48504        whether the SIGALRM signal is blocked from delivery, the results are unspecified.

48505        If a signal-catching function interrupts *usleep()* and calls *siglongjmp()* or *longjmp()* to restore an  
 48506        environment saved prior to the *usleep()* call, the action associated with the SIGALRM signal and  
 48507        the time at which a SIGALRM signal is scheduled to be generated are unspecified. It is also  
 48508        unspecified whether the SIGALRM signal is blocked, unless the process' signal mask is restored  
 48509        as part of the environment.

48510        Implementations may place limitations on the granularity of timer values. For each interval  
 48511        timer, if the requested timer value requires a finer granularity than the implementation supports,  
 48512        the actual timer value shall be rounded up to the next supported value.

48513        Interactions between *usleep()* and any of the following are unspecified:

48514            *nanosleep()*48515            *setitimer()*48516            *timer\_create()*48517            *timer\_delete()*48518            *timer\_getoverrun()*48519            *timer\_gettime()*48520            *timer\_settime()*48521            *ualarm()*48522            *sleep()*

## 48523 RETURN VALUE

48524       Upon successful completion, *usleep()* shall return 0; otherwise, it shall return -1 and set *errno* to  
48525       indicate the error.

## 48526 ERRORS

48527       The *usleep()* function may fail if:

48528       [EINVAL]       The time interval specified one million or more microseconds.

## 48529 EXAMPLES

48530       None.

## 48531 APPLICATION USAGE

48532       Applications are recommended to use *nanosleep()* if the Timers option is supported, or  
48533       *setitimer()*, *timer\_create()*, *timer\_delete()*, *timer\_getoverrun()*, *timer\_gettime()*, or *timer\_settime()*  
48534       instead of this function.

## 48535 RATIONALE

48536       None.

## 48537 FUTURE DIRECTIONS

48538       None.

## 48539 SEE ALSO

48540       *alarm()*, *getitimer()*, *nanosleep()*, *sigaction()*, *sleep()*, *timer\_create()*, *timer\_delete()*,  
48541       *timer\_getoverrun()*, the Base Definitions volume of IEEE Std 1003.1-200x, <**unistd.h**>

## 48542 CHANGE HISTORY

48543       First released in Issue 4, Version 2.

### 48544 Issue 5

48545       Moved from X/OPEN UNIX extension to BASE.

48546       The DESCRIPTION is changed to indicate that timers are now thread-based rather than  
48547       process-based.

### 48548 Issue 6

48549       The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

48550       This function is marked obsolescent.

48551 **NAME**

48552 utime — set file access and modification times

48553 **SYNOPSIS**

48554 #include &lt;utime.h&gt;

48555 int utime(const char \*path, const struct utimbuf \*times);

48556 **DESCRIPTION**48557 The *utime()* function shall set the access and modification times of the file named by the *path*  
48558 argument.48559 If *times* is a null pointer, the access and modification times of the file shall be set to the current |  
48560 time. The effective user ID of the process shall match the owner of the file, or the process has |  
48561 write permission to the file or has appropriate privileges, to use *utime()* in this manner. |48562 If *times* is not a null pointer, *times* shall be interpreted as a pointer to a **utimbuf** structure and the |  
48563 access and modification times shall be set to the values contained in the designated structure. |  
48564 Only a process with effective user ID equal to the user ID of the file or a process with |  
48565 appropriate privileges may use *utime()* this way. |48566 The **utimbuf** structure is defined in the <**utime.h**> header. The times in the structure **utimbuf** |  
48567 are measured in seconds since the Epoch.48568 Upon successful completion, *utime()* shall mark the time of the last file status change, *st\_ctime*,  
48569 to be updated; see <**sys/stat.h**>.48570 **RETURN VALUE**48571 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* shall  
48572 be set to indicate the error, and the file times shall not be affected.48573 **ERRORS**48574 The *utime()* function shall fail if:48575 [EACCES] Search permission is denied by a component of the path prefix; or the *times*  
48576 argument is a null pointer and the effective user ID of the process does not  
48577 match the owner of the file, the process does not have write permission for the  
48578 file, and the process does not have appropriate privileges.48579 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*  
48580 argument.48581 [ENAMETOOLONG]  
48582 The length of the *path* argument exceeds {PATH\_MAX} or a pathname |  
48583 component is longer than {NAME\_MAX}. |48584 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

48585 [ENOTDIR] A component of the path prefix is not a directory.

48586 [EPERM] The *times* argument is not a null pointer and the calling process' effective user  
48587 ID does not match the owner of the file and the calling process does not have  
48588 the appropriate privileges.

48589 [EROFS] The file system containing the file is read-only.

48590 The *utime()* function may fail if:48591 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
48592 resolution of the *path* argument.

48593 [ENAMETOOLONG]  
48594 As a result of encountering a symbolic link in resolution of the *path* argument, |  
48595 the length of the substituted pathname string exceeded {PATH\_MAX}. |

**48596 EXAMPLES**

48597 None.

**48598 APPLICATION USAGE**

48599 None.

**48600 RATIONALE**

48601 The *actime* structure member must be present so that an application may set it, even though an  
48602 implementation may ignore it and not change the access time on the file. If an application  
48603 intends to leave one of the times of a file unchanged while changing the other, it should use  
48604 *stat()* to retrieve the file's *st\_atime* and *st\_mtime* parameters, set *actime* and *modtime* in the buffer,  
48605 and change one of them before making the *utime()* call.

**48606 FUTURE DIRECTIONS**

48607 None.

**48608 SEE ALSO**

48609 The Base Definitions volume of IEEE Std 1003.1-200x, <**sys/types.h**>, <**utime.h**>

**48610 CHANGE HISTORY**

48611 First released in Issue 1. Derived from Issue 1 of the SVID.

**48612 Issue 6**

48613 In the SYNOPSIS, the optional include of the <**sys/types.h**> header is removed.

48614 The following new requirements on POSIX implementations derive from alignment with the |  
48615 Single UNIX Specification:

48616 • The requirement to include <**sys/types.h**> has been removed. Although <**sys/types.h**> was  
48617 required for conforming implementations of previous POSIX specifications, it was not  
48618 required for UNIX applications.

48619 • The [ELOOP] mandatory error condition is added.

48620 • A second [ENAMETOOLONG] is added as an optional error condition.

48621 The following changes were made to align with the IEEE P1003.1a draft standard:

48622 • The [ELOOP] optional error condition is added.

48623 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

48624 **NAME**48625 utimes — set file access and modification times (**LEGACY**)48626 **SYNOPSIS**48627 XSI `#include <sys/time.h>`48628 `int utimes(const char *path, const struct timeval times[2]);`

48629

48630 **DESCRIPTION**

48631 The *utimes()* function shall set the access and modification times of the file pointed to by the *path*  
 48632 argument to the value of the *times* argument. The *utimes()* function allows time specifications  
 48633 accurate to the microsecond.

48634 For *utimes()*, the *times* argument is an array of **timeval** structures. The first array member  
 48635 represents the date and time of last access, and the second member represents the date and time  
 48636 of last modification. The times in the **timeval** structure are measured in seconds and  
 48637 microseconds since the Epoch, although rounding toward the nearest second may occur.

48638 If the *times* argument is a null pointer, the access and modification times of the file shall be set to |  
 48639 the current time. The effective user ID of the process shall match the owner of the file, or has |  
 48640 write access to the file or appropriate privileges to use this call in this manner. Upon completion, |  
 48641 *utimes()* shall mark the time of the last file status change, *st\_ctime*, for update.

48642 **RETURN VALUE**

48643 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* shall  
 48644 be set to indicate the error, and the file times shall not be affected.

48645 **ERRORS**48646 The *utimes()* function shall fail if:

48647 [EACCES] Search permission is denied by a component of the path prefix; or the *times*  
 48648 argument is a null pointer and the effective user ID of the process does not  
 48649 match the owner of the file and write access is denied.

48650 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*  
 48651 argument.

48652 [ENAMETOOLONG]  
 48653 The length of the *path* argument exceeds {PATH\_MAX} or a pathname |  
 48654 component is longer than {NAME\_MAX}.

48655 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

48656 [ENOTDIR] A component of the path prefix is not a directory.

48657 [EPERM] The *times* argument is not a null pointer and the calling process' effective user  
 48658 ID has write access to the file but does not match the owner of the file and the  
 48659 calling process does not have the appropriate privileges.

48660 [EROFS] The file system containing the file is read-only.

48661 The *utimes()* function may fail if:

48662 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
 48663 resolution of the *path* argument.

48664 [ENAMETOOLONG]  
 48665 Pathname resolution of a symbolic link produced an intermediate result |  
 48666 whose length exceeds {PATH\_MAX}.

48667 **EXAMPLES**

48668           None.

48669 **APPLICATION USAGE**48670           For applications portability, the *utime()* function should be used to set file access and  
48671           modification times instead of *utimes()*.48672 **RATIONALE**

48673           None.

48674 **FUTURE DIRECTIONS**

48675           This function may be withdrawn in a future version.

48676 **SEE ALSO**48677           The Base Definitions volume of IEEE Std 1003.1-200x, <**sys/time.h**>48678 **CHANGE HISTORY**

48679           First released in Issue 4, Version 2.

48680 **Issue 5**

48681           Moved from X/OPEN UNIX extension to BASE.

48682 **Issue 6**

48683           This function is marked LEGACY.

48684           The DESCRIPTION is updated to avoid use of the term “must” for application requirements. |

48685           The wording of the mandatory [ELOOP] error condition is updated, and a second optional  
48686           [ELOOP] error condition is added.

48687 **NAME**

48688       va\_arg, va\_copy, va\_end, va\_start — handle variable argument list

48689 **SYNOPSIS**

48690       #include &lt;stdarg.h&gt;

48691       type va\_arg(va\_list ap, type);

48692       void va\_copy(va\_list dest, va\_list src);

48693       void va\_end(va\_list ap);

48694       void va\_start(va\_list ap, argN);

48695 **DESCRIPTION**

48696       Refer to the Base Definitions volume of IEEE Std 1003.1-200x, &lt;stdarg.h&gt;.

48697 **NAME**

48698 vfork — create new process; share virtual memory

48699 **SYNOPSIS**

48700 OB XSI #include &lt;unistd.h&gt;

48701 pid\_t vfork(void);

48702

48703 **DESCRIPTION**

48704 The *vfork()* function shall be equivalent to *fork()*, except that the behavior is undefined if the  
 48705 process created by *vfork()* either modifies any data other than a variable of type **pid\_t** used to  
 48706 store the return value from *vfork()*, or returns from the function in which *vfork()* was called, or  
 48707 calls any other function before successfully calling *\_exit()* or one of the *exec* family of functions.

48708 **RETURN VALUE**

48709 Upon successful completion, *vfork()* shall return 0 to the child process and return the process ID  
 48710 of the child process to the parent process. Otherwise, -1 shall be returned to the parent, no child  
 48711 process shall be created, and *errno* shall be set to indicate the error.

48712 **ERRORS**48713 The *vfork()* function shall fail if:

48714 [EAGAIN] The system-wide limit on the total number of processes under execution  
 48715 would be exceeded, or the system-imposed limit on the total number of  
 48716 processes under execution by a single user would be exceeded.

48717 [ENOMEM] There is insufficient swap space for the new process.

48718 **EXAMPLES**

48719 None.

48720 **APPLICATION USAGE**

48721 Conforming applications are recommended not to depend on *vfork()*, but to use *fork()* instead.  
 48722 The *vfork()* function may be withdrawn in a future version.

48723 On some implementations, *vfork()* is equivalent to *fork()*.

48724 The *vfork()* function differs from *fork()* only in that the child process can share code and data  
 48725 with the calling process (parent process). This speeds cloning activity significantly at a risk to  
 48726 the integrity of the parent process if *vfork()* is misused.

48727 The use of *vfork()* for any purpose except as a prelude to an immediate call to a function from  
 48728 the *exec* family, or to *\_exit()*, is not advised.

48729 The *vfork()* function can be used to create new processes without fully copying the address  
 48730 space of the old process. If a forked process is simply going to call *exec*, the data space copied  
 48731 from the parent to the child by *fork()* is not used. This is particularly inefficient in a paged  
 48732 environment, making *vfork()* particularly useful. Depending upon the size of the parent's data  
 48733 space, *vfork()* can give a significant performance improvement over *fork()*.

48734 The *vfork()* function can normally be used just like *fork()*. It does not work, however, to return  
 48735 while running in the child's context from the caller of *vfork()* since the eventual return from  
 48736 *vfork()* would then return to a no longer existent stack frame. Care should be taken, also, to call  
 48737 *\_exit()* rather than *exit()* if *exec* cannot be used, since *exit()* flushes and closes standard I/O  
 48738 channels, thereby damaging the parent process' standard I/O data structures. (Even with *fork()*,  
 48739 it is wrong to call *exit()*, since buffered data would then be flushed twice.)

48740 If signal handlers are invoked in the child process after *vfork()*, they must follow the same rules  
 48741 as other code in the child process.

48742 **RATIONALE**

48743           None.

48744 **FUTURE DIRECTIONS**

48745           This function may be withdrawn in a future version.

48746 **SEE ALSO**48747           *exec*, *exit()*, *fork()*, *wait()*, the Base Definitions volume of IEEE Std 1003.1-200x, <**unistd.h**>48748 **CHANGE HISTORY**

48749           First released in Issue 4, Version 2.

48750 **Issue 5**

48751           Moved from X/OPEN UNIX extension to BASE.

48752 **Issue 6**

48753           Marked obsolescent.

48754 **NAME**

48755 vfprintf, vprintf, vsnprintf, vsprintf — format output of a stdarg argument list

48756 **SYNOPSIS**

48757 #include &lt;stdarg.h&gt;

48758 #include &lt;stdio.h&gt;

48759 int vfprintf(FILE \*restrict stream, const char \*restrict format,  
48760 va\_list ap);

48761 int vprintf(const char \*restrict format, va\_list ap);

48762 int vsnprintf(char \*restrict s, size\_t n, const char \*restrict format,  
48763 va\_list ap);

48764 int vsprintf(char \*restrict s, const char \*restrict format, va\_list ap);

48765 **DESCRIPTION**48766 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
48767 conflict between the requirements described here and the ISO C standard is unintentional. This  
48768 volume of IEEE Std 1003.1-200x defers to the ISO C standard.48769 The *vprintf()*, *vfprintf()*, *vsnprintf()*, and *vsprintf()* functions shall be equivalent to *printf()*,  
48770 *fprintf()*, *snprintf()*, and *sprintf()* respectively, except that instead of being called with a variable  
48771 number of arguments, they are called with an argument list as defined by <stdarg.h>.48772 These functions shall not invoke the *va\_end* macro. As these functions invoke the *va\_arg* macro, |  
48773 the value of *ap* after the return is unspecified. |48774 **RETURN VALUE**48775 Refer to *fprintf()*.48776 **ERRORS**48777 Refer to *fprintf()*.48778 **EXAMPLES**

48779 None.

48780 **APPLICATION USAGE**48781 Applications using these functions should call *va\_end(ap)* afterwards to clean up.48782 **RATIONALE**

48783 None.

48784 **FUTURE DIRECTIONS**

48785 None.

48786 **SEE ALSO**48787 *fprintf()*, the Base Definitions volume of IEEE Std 1003.1-200x, <stdarg.h>, <stdio.h>48788 **CHANGE HISTORY**

48789 First released in Issue 1. Derived from Issue 1 of the SVID.

48790 **Issue 5**48791 The *vsnprintf()* function is added.48792 **Issue 6**48793 The *vfprintf()*, *vprintf()*, *vsnprintf()*, and *vsprintf()* functions are updated for alignment with the  
48794 ISO/IEC 9899:1999 standard.

48795 **NAME**48796 `vfscanf`, `vscanf`, `vsscanf` — format input of a stdarg list48797 **SYNOPSIS**48798 `#include <stdarg.h>`48799 `#include <stdio.h>`48800 `int vfscanf(FILE *restrict stream, const char *restrict format,`  
48801 `va_list arg);`48802 `int vscanf(const char *restrict format, va_list arg);`48803 `int vsscanf(const char *restrict s, const char *restrict format,`  
48804 `va_list arg);`48805 **DESCRIPTION**48806 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
48807 conflict between the requirements described here and the ISO C standard is unintentional. This  
48808 volume of IEEE Std 1003.1-200x defers to the ISO C standard.48809 The `vscanf()`, `vfscanf()`, and `vsscanf()` functions shall be equivalent to the `scanf()`, `fscanf()`, and  
48810 `sscanf()` functions, respectively, except that instead of being called with a variable number of  
48811 arguments, they are called with an argument list as defined in the `<stdarg.h>` header. These  
48812 functions shall not invoke the `va_end` macro. As these functions invoke the `va_arg` macro, the  
48813 value of `ap` after the return is unspecified. |48814 **RETURN VALUE**48815 Refer to `fscanf()`.48816 **ERRORS**48817 Refer to `fscanf()`.48818 **EXAMPLES**

48819 None.

48820 **APPLICATION USAGE**48821 Applications using these functions should call `va_end(ap)` afterwards to clean up.48822 **RATIONALE**

48823 None.

48824 **FUTURE DIRECTIONS**

48825 None.

48826 **SEE ALSO**48827 `fscanf()`, the Base Definitions volume of IEEE Std 1003.1-200x, `<stdarg.h>`, `<stdio.h>`48828 **CHANGE HISTORY**

48829 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

48830 **NAME**

48831 vfwprintf, vswprintf, vwprintf — wide-character formatted output of a stdarg argument list

48832 **SYNOPSIS**

48833 #include &lt;stdarg.h&gt;

48834 #include &lt;stdio.h&gt;

48835 #include &lt;wchar.h&gt;

48836 int vfwprintf(FILE \*restrict stream, const wchar\_t \*restrict format,  
48837 va\_list arg);48838 int vswprintf(wchar\_t \*restrict ws, size\_t n,  
48839 const wchar\_t \*restrict format, va\_list arg);

48840 int vwprintf(const wchar\_t \*restrict format, va\_list arg);

48841 **DESCRIPTION**48842 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
48843 conflict between the requirements described here and the ISO C standard is unintentional. This  
48844 volume of IEEE Std 1003.1-200x defers to the ISO C standard.48845 The *vfwprintf()*, *vswprintf()*, and *vwprintf()* functions shall be equivalent to *fwprintf()*, *swprintf()*,  
48846 and *wprintf()* respectively, except that instead of being called with a variable number of  
48847 arguments, they are called with an argument list as defined by <stdarg.h>.48848 These functions shall not invoke the *va\_end* macro. However, as these functions do invoke the  
48849 *va\_arg* macro, the value of *ap* after the return is unspecified. |48850 **RETURN VALUE**48851 Refer to *fwprintf()*.48852 **ERRORS**48853 Refer to *fwprintf()*.48854 **EXAMPLES**

48855 None.

48856 **APPLICATION USAGE**48857 Applications using these functions should call *va\_end(ap)* afterwards to clean up.48858 **RATIONALE**

48859 None.

48860 **FUTURE DIRECTIONS**

48861 None.

48862 **SEE ALSO**48863 *fwprintf()*, the Base Definitions volume of IEEE Std 1003.1-200x, <stdarg.h>, <stdio.h>,  
48864 <wchar.h>48865 **CHANGE HISTORY**48866 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995  
48867 (E).48868 **Issue 6**48869 The *vfwprintf()*, *vswprintf()*, and *vwprintf()* prototypes are updated for alignment with the  
48870 ISO/IEC 9899:1999 standard. ()

48871 **NAME**

48872 vfwscanf, vswscanf, vwscanf — wide-character formatted input of a stdarg list

48873 **SYNOPSIS**

48874 #include &lt;stdarg.h&gt;

48875 #include &lt;stdio.h&gt;

48876 #include &lt;wchar.h&gt;

48877 int vfwscanf(FILE \*restrict stream, const wchar\_t \*restrict format,  
48878 va\_list arg);48879 int vswscanf(const wchar\_t \*restrict ws, const wchar\_t \*restrict format,  
48880 va\_list arg);

48881 int vwscanf(const wchar\_t \*restrict format, va\_list arg);

48882 **DESCRIPTION**48883 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
48884 conflict between the requirements described here and the ISO C standard is unintentional. This  
48885 volume of IEEE Std 1003.1-200x defers to the ISO C standard.48886 The *vfwscanf()*, *vswscanf()*, and *vwscanf()* functions shall be equivalent to the *fwscanf()*, |  
48887 *swscanf()*, and *wscanf()* functions, respectively, except that instead of being called with a |  
48888 variable number of arguments, they are called with an argument list as defined in the <stdarg.h> |  
48889 header. These functions shall not invoke the *va\_end* macro. As these functions invoke the *va\_arg* |  
48890 macro, the value of *ap* after the return is unspecified. |48891 **RETURN VALUE**48892 Refer to *fwscanf()*.48893 **ERRORS**48894 Refer to *fwscanf()*.48895 **EXAMPLES**

48896 None.

48897 **APPLICATION USAGE**48898 Applications using these functions should call *va\_end(ap)* afterwards to clean up.48899 **RATIONALE**

48900 None.

48901 **FUTURE DIRECTIONS**

48902 None.

48903 **SEE ALSO**48904 *fwscanf()*, the Base Definitions volume of IEEE Std 1003.1-200x, <stdarg.h>, <stdio.h>,  
48905 <wchar.h>48906 **CHANGE HISTORY**

48907 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

48908 **NAME**

48909           vprintf — format output of a stdarg argument list

48910 **SYNOPSIS**

48911           #include <stdarg.h>

48912           #include <stdio.h>

48913           int vprintf(const char \*restrict *format*, va\_list *ap*);

48914 **DESCRIPTION**

48915           Refer to *fprintf()*.

48916 **NAME**

48917           vscanf — format input of a stdarg list

48918 **SYNOPSIS**

48919           #include &lt;stdarg.h&gt;

48920           #include &lt;stdio.h&gt;

48921           int vscanf(const char \*restrict *format*, va\_list *arg*);48922 **DESCRIPTION**48923           Refer to *vscanf()*.

48924 **NAME**

48925           vsprintf, vsprintf — format output of a stdarg argument list

48926 **SYNOPSIS**

48927           #include &lt;stdarg.h&gt;

48928           #include &lt;stdio.h&gt;

48929           int vsnprintf(char \*restrict *s*, size\_t *n*,48930                        const char \*restrict *format*, va\_list *ap*);48931           int vsprintf(char \*restrict *s*, const char \*restrict *format*,48932                        va\_list *ap*);48933 **DESCRIPTION**48934           Refer to *fprintf()*.

48935 **NAME**

48936 vsscanf — format input of a stdarg list

48937 **SYNOPSIS**

48938 #include &lt;stdarg.h&gt;

48939 #include &lt;stdio.h&gt;

48940 int vsscanf(const char \*restrict *s*, const char \*restrict *format*,48941 va\_list *arg*);48942 **DESCRIPTION**48943 Refer to *vfscanf()*.

48944 **NAME**

48945       vswprintf — wide-character formatted output of a stdarg argument list

48946 **SYNOPSIS**

48947       #include <stdarg.h>

48948       #include <stdio.h>

48949       #include <wchar.h>

48950       int vswprintf(wchar\_t \*restrict ws, size\_t n,  
48951                    const wchar\_t \*restrict format, va\_list arg);

48952 **DESCRIPTION**

48953       Refer to *vfwprintf()*.

48954 **NAME**

48955 vswscanf — wide-character formatted input of a stdarg list

48956 **SYNOPSIS**

48957 #include &lt;stdarg.h&gt;

48958 #include &lt;stdio.h&gt;

48959 #include &lt;wchar.h&gt;

48960 int vswscanf(const wchar\_t \*restrict ws, const wchar\_t \*restrict format,  
48961 va\_list arg);48962 **DESCRIPTION**48963 Refer to *vfwscanf()*.

48964 **NAME**

48965       vwprintf — wide-character formatted output of a stdarg argument list

48966 **SYNOPSIS**

48967       #include <stdarg.h>

48968       #include <stdio.h>

48969       #include <wchar.h>

48970       int vwprintf(const wchar\_t \*restrict *format*, va\_list *arg*);

48971 **DESCRIPTION**

48972       Refer to *vfwprintf()*.

48973 **NAME**

48974 vwscanf — wide-character formatted input of a stdarg list

48975 **SYNOPSIS**

48976 #include &lt;stdarg.h&gt;

48977 #include &lt;stdio.h&gt;

48978 #include &lt;wchar.h&gt;

48979 int vwscanf(const wchar\_t \*restrict *format*, va\_list *arg*);48980 **DESCRIPTION**48981 Refer to *vfwscanf()*.

## 48982 NAME

48983 wait, waitpid — wait for a child process to stop or terminate

## 48984 SYNOPSIS

48985 #include &lt;sys/wait.h&gt;

48986 pid\_t wait(int \*stat\_loc);

48987 pid\_t waitpid(pid\_t pid, int \*stat\_loc, int options);

## 48988 DESCRIPTION

48989 The *wait()* and *waitpid()* functions shall obtain status information pertaining to one of the |  
 48990 caller's child processes. Various options permit status information to be obtained for child |  
 48991 processes that have terminated or stopped. If status information is available for two or more |  
 48992 child processes, the order in which their status is reported is unspecified. |

48993 The *wait()* function shall suspend execution of the calling thread until status information for one |  
 48994 of the terminated child processes of the calling process is available, or until delivery of a signal |  
 48995 whose action is either to execute a signal-catching function or to terminate the process. If more |  
 48996 than one thread is suspended in *wait()* or *waitpid()* awaiting termination of the same process, |  
 48997 exactly one thread shall return the process status at the time of the target process termination. If |  
 48998 status information is available prior to the call to *wait()*, return shall be immediate.

48999 The *waitpid()* function shall be equivalent to *wait()* if the *pid* argument is (**pid\_t**)−1 and the |  
 49000 *options* argument is 0. Otherwise, its behavior shall be modified by the values of the *pid* and |  
 49001 *options* arguments.

49002 The *pid* argument specifies a set of child processes for which *status* is requested. The *waitpid()* |  
 49003 function shall only return the status of a child process from this set:

- 49004 • If *pid* is equal to (**pid\_t**)−1, *status* is requested for any child process. In this respect, *waitpid()* |  
 49005 is then equivalent to *wait()*.
- 49006 • If *pid* is greater than 0, it specifies the process ID of a single child process for which *status* is |  
 49007 requested.
- 49008 • If *pid* is 0, *status* is requested for any child process whose process group ID is equal to that of |  
 49009 the calling process.
- 49010 • If *pid* is less than (**pid\_t**)−1, *status* is requested for any child process whose process group ID |  
 49011 is equal to the absolute value of *pid*.

49012 The *options* argument is constructed from the bitwise-inclusive OR of zero or more of the |  
 49013 following flags, defined in the <sys/wait.h> header:

49014 XSI WCONTINUED The *waitpid()* function shall report the status of any continued child process |  
 49015 specified by *pid* whose status has not been reported since it continued from a |  
 49016 job control stop.

49017 WNOHANG The *waitpid()* function shall not suspend execution of the calling thread if |  
 49018 *status* is not immediately available for one of the child processes specified by |  
 49019 *pid*.

49020 WUNTRACED The status of any child processes specified by *pid* that are stopped, and whose |  
 49021 status has not yet been reported since they stopped, shall also be reported to |  
 49022 the requesting process.

49023 XSI If the calling process has SA\_NOCLDWAIT set or has SIGCHLD set to SIG\_IGN, and the |  
 49024 process has no unwaited-for children that were transformed into zombie processes, the calling |  
 49025 thread shall block until all of the children of the process containing the calling thread terminate, |  
 49026 and *wait()* and *waitpid()* shall fail and set *errno* to [ECHILD].

49027 If *wait()* or *waitpid()* return because the status of a child process is available, these functions  
 49028 shall return a value equal to the process ID of the child process. In this case, if the value of the  
 49029 argument *stat\_loc* is not a null pointer, information shall be stored in the location pointed to by  
 49030 *stat\_loc*. The value stored at the location pointed to by *stat\_loc* shall be 0 if and only if the status  
 49031 returned is from a terminated child process that terminated by one of the following means:

- 49032 1. The process returned 0 from *main()*.
- 49033 2. The process called *\_exit()* or *exit()* with a *status* argument of 0.
- 49034 3. The process was terminated because the last thread in the process terminated.

49035 Regardless of its value, this information may be interpreted using the following macros, which  
 49036 are defined in `<sys/wait.h>` and evaluate to integral expressions; the *stat\_val* argument is the  
 49037 integer value pointed to by *stat\_loc*.

49038 **WIFEXITED(*stat\_val*)**

49039 Evaluates to a non-zero value if *status* was returned for a child process that terminated  
 49040 normally.

49041 **WEXITSTATUS(*stat\_val*)**

49042 If the value of **WIFEXITED(*stat\_val*)** is non-zero, this macro evaluates to the low-order 8 bits  
 49043 of the *status* argument that the child process passed to *\_exit()* or *exit()*, or the value the child  
 49044 process returned from *main()*.

49045 **WIFSIGNALED(*stat\_val*)**

49046 Evaluates to non-zero value if *status* was returned for a child process that terminated due to  
 49047 the receipt of a signal that was not caught (see `<signal.h>`).

49048 **WTERMSIG(*stat\_val*)**

49049 If the value of **WIFSIGNALED(*stat\_val*)** is non-zero, this macro evaluates to the number of  
 49050 the signal that caused the termination of the child process.

49051 **WIFSTOPPED(*stat\_val*)**

49052 Evaluates to a non-zero value if *status* was returned for a child process that is currently  
 49053 stopped.

49054 **WSTOPSIG(*stat\_val*)**

49055 If the value of **WIFSTOPPED(*stat\_val*)** is non-zero, this macro evaluates to the number of the  
 49056 signal that caused the child process to stop.

49057 XSI **WIFCONTINUED(*stat\_val*)**

49058 Evaluates to a non-zero value if *status* was returned for a child process that has continued  
 49059 from a job control stop.

49060 SPN It is unspecified whether the *status* value returned by calls to *wait()* or *waitpid()* for processes  
 49061 created by *posix\_spawn()* or *posix\_spawnnp()* can indicate a **WIFSTOPPED(*stat\_val*)** before  
 49062 subsequent calls to *wait()* or *waitpid()* indicate **WIFEXITED(*stat\_val*)** as the result of an error  
 49063 detected before the new process image starts executing.

49064 It is unspecified whether the *status* value returned by calls to *wait()* or *waitpid()* for processes  
 49065 created by *posix\_spawn()* or *posix\_spawnnp()* can indicate a **WIFSIGNALED(*stat\_val*)** if a signal is  
 49066 sent to the parent's process group after *posix\_spawn()* or *posix\_spawnnp()* is called.

49067 If the information pointed to by *stat\_loc* was stored by a call to *waitpid()* that specified the  
 49068 XSI **WUNTRACED** flag and did not specify the **WCONTINUED** flag, exactly one of the macros  
 49069 **WIFEXITED(\**stat\_loc*)**, **WIFSIGNALED(\**stat\_loc*)**, and **WIFSTOPPED(\**stat\_loc*)** shall evaluate to  
 49070 a non-zero value.

49071 If the information pointed to by *stat\_loc* was stored by a call to *waitpid()* that specified the  
 49072 XSI WUNTRACED and WCONTINUED flags, exactly one of the macros WIFEXITED(\**stat\_loc*),  
 49073 XSI WIFSIGNALED(\**stat\_loc*), WIFSTOPPED(\**stat\_loc*), and WIFCONTINUED(\**stat\_loc*) shall  
 49074 evaluate to a non-zero value.

49075 If the information pointed to by *stat\_loc* was stored by a call to *waitpid()* that did not specify the  
 49076 XSI WUNTRACED or WCONTINUED flags, or by a call to the *wait()* function, exactly one of the  
 49077 macros WIFEXITED(\**stat\_loc*) and WIFSIGNALED(\**stat\_loc*) shall evaluate to a non-zero value.

49078 If the information pointed to by *stat\_loc* was stored by a call to *waitpid()* that did not specify the  
 49079 XSI WUNTRACED flag and specified the WCONTINUED flag, or by a call to the *wait()* function,  
 49080 XSI exactly one of the macros WIFEXITED(\**stat\_loc*), WIFSIGNALED(\**stat\_loc*), and  
 49081 WIFCONTINUED(\**stat\_loc*) shall evaluate to a non-zero value.

49082 If `_POSIX_REALTIME_SIGNALS` is defined, and the implementation queues the SIGCHLD  
 49083 signal, then if *wait()* or *waitpid()* returns because the status of a child process is available, any  
 49084 pending SIGCHLD signal associated with the process ID of the child process shall be discarded.  
 49085 Any other pending SIGCHLD signals shall remain pending.

49086 Otherwise, if SIGCHLD is blocked, if *wait()* or *waitpid()* return because the status of a child  
 49087 process is available, any pending SIGCHLD signal shall be cleared unless the status of another  
 49088 child process is available.

49089 For all other conditions, it is unspecified whether child *status* will be available when a SIGCHLD  
 49090 signal is delivered.

49091 There may be additional implementation-defined circumstances under which *wait()* or *waitpid()*  
 49092 report *status*. This shall not occur unless the calling process or one of its child processes explicitly  
 49093 makes use of a non-standard extension. In these cases the interpretation of the reported *status* is  
 49094 implementation-defined.

49095 XSI If a parent process terminates without waiting for all of its child processes to terminate, the  
 49096 remaining child processes shall be assigned a new parent process ID corresponding to an  
 49097 implementation-defined system process.

#### 49098 RETURN VALUE

49099 If *wait()* or *waitpid()* returns because the status of a child process is available, these functions  
 49100 shall return a value equal to the process ID of the child process for which *status* is reported. If  
 49101 *wait()* or *waitpid()* returns due to the delivery of a signal to the calling process, `-1` shall be  
 49102 returned and *errno* set to `[EINTR]`. If *waitpid()* was invoked with `WNOHANG` set in *options*, it  
 49103 has at least one child process specified by *pid* for which *status* is not available, and *status* is not  
 49104 available for any process specified by *pid*, `0` is returned. Otherwise, `(pid_t)-1` shall be returned,  
 49105 and *errno* set to indicate the error.

#### 49106 ERRORS

49107 The *wait()* function shall fail if:

49108 `[ECHILD]` The calling process has no existing unwaited-for child processes.

49109 `[EINTR]` The function was interrupted by a signal. The value of the location pointed to  
 49110 by *stat\_loc* is undefined.

49111 The *waitpid()* function shall fail if:

49112 `[ECHILD]` The process specified by *pid* does not exist or is not a child of the calling  
 49113 process, or the process group specified by *pid* does not exist or does not have  
 49114 any member process that is a child of the calling process.

49115 [EINTR] The function was interrupted by a signal. The value of the location pointed to  
49116 by *stat\_loc* is undefined.

49117 [EINVAL] The *options* argument is not valid.

#### 49118 EXAMPLES

49119 None.

#### 49120 APPLICATION USAGE

49121 None.

#### 49122 RATIONALE

49123 A call to the *wait()* or *waitpid()* function only returns *status* on an immediate child process of the  
49124 calling process; that is, a child that was produced by a single *fork()* call (perhaps followed by an  
49125 *exec* or other function calls) from the parent. If a child produces grandchildren by further use of  
49126 *fork()*, none of those grandchildren nor any of their descendants affect the behavior of a *wait()*  
49127 from the original parent process. Nothing in this volume of IEEE Std 1003.1-200x prevents an  
49128 implementation from providing extensions that permit a process to get *status* from a grandchild  
49129 or any other process, but a process that does not use such extensions must be guaranteed to see  
49130 *status* from only its direct children.

49131 The *waitpid()* function is provided for three reasons:

- 49132 1. To support job control
- 49133 2. To permit a non-blocking version of the *wait()* function
- 49134 3. To permit a library routine, such as *system()* or *pclose()*, to wait for its children without  
49135 interfering with other terminated children for which the process has not waited

49136 The first two of these facilities are based on the *wait3()* function provided by 4.3 BSD. The  
49137 function uses the *options* argument, which is equivalent to an argument to *wait3()*. The  
49138 WUNTRACED flag is used only in conjunction with job control on systems supporting job  
49139 control. Its name comes from 4.3 BSD and refers to the fact that there are two types of stopped  
49140 processes in that implementation: processes being traced via the *ptrace()* debugging facility and  
49141 (untraced) processes stopped by job control signals. Since *ptrace()* is not part of this volume of  
49142 IEEE Std 1003.1-200x, only the second type is relevant. The name WUNTRACED was retained  
49143 because its usage is the same, even though the name is not intuitively meaningful in this context.

49144 The third reason for the *waitpid()* function is to permit independent sections of a process to  
49145 spawn and wait for children without interfering with each other. For example, the following  
49146 problem occurs in developing a portable shell, or command interpreter:

```
49147 stream = popen("/bin/true");
49148 (void) system("sleep 100");
49149 (void) pclose(stream);
```

49150 On all historical implementations, the final *pclose()* fails to reap the *wait()* *status* of the *popen()*.

49151 The status values are retrieved by macros, rather than given as specific bit encodings as they are  
49152 in most historical implementations (and thus expected by existing programs). This was  
49153 necessary to eliminate a limitation on the number of signals an implementation can support that  
49154 was inherent in the traditional encodings. This volume of IEEE Std 1003.1-200x does require that  
49155 a *status* value of zero corresponds to a process calling *\_exit(0)*, as this is the most common  
49156 encoding expected by existing programs. Some of the macro names were adopted from 4.3 BSD.

49157 These macros syntactically operate on an arbitrary integer value. The behavior is undefined  
49158 unless that value is one stored by a successful call to *wait()* or *waitpid()* in the location pointed  
49159 to by the *stat\_loc* argument. An early proposal attempted to make this clearer by specifying each

49160 argument as *\*stat\_loc* rather than *stat\_val*. However, that did not follow the conventions of other  
49161 specifications in this volume of IEEE Std 1003.1-200x or traditional usage. It also could have  
49162 implied that the argument to the macro must literally be *\*stat\_loc*; in fact, that value can be  
49163 stored or passed as an argument to other functions before being interpreted by these macros.

49164 The extension that affects *wait()* and *waitpid()* and is common in historical implementations is  
49165 the *ptrace()* function. It is called by a child process and causes that child to stop and return a  
49166 *status* that appears identical to the *status* indicated by WIFSTOPPED. The *status* of *ptrace()*  
49167 children is traditionally returned regardless of the WUNTRACED flag (or by the *wait()*  
49168 function). Most applications do not need to concern themselves with such extensions because  
49169 they have control over what extensions they or their children use. However, applications, such  
49170 as command interpreters, that invoke arbitrary processes may see this behavior when those  
49171 arbitrary processes misuse such extensions.

49172 Implementations that support *core* file creation or other implementation-defined actions on  
49173 termination of some processes traditionally provide a bit in the *status* returned by *wait()* to  
49174 indicate that such actions have occurred.

49175 Allowing the *wait()* family of functions to discard a pending SIGCHLD signal that is associated  
49176 with a successfully waited-for child process puts them into the *sigwait()* and *sigwaitinfo()*  
49177 category with respect to SIGCHLD.

49178 This definition allows implementations to treat a pending SIGCHLD signal as accepted by the  
49179 process in *wait()*, with the same meaning of “accepted” as when that word is applied to the  
49180 *sigwait()* family of functions.

49181 Allowing the *wait()* family of functions to behave this way permits an implementation to be able  
49182 to deal precisely with SIGCHLD signals.

49183 In particular, an implementation that does accept (discard) the SIGCHLD signal can make the  
49184 following guarantees regardless of the queuing depth of signals in general (the list of waitable  
49185 children can hold the SIGCHLD queue):

- 49186 1. If a SIGCHLD signal handler is established via *sigaction()* without the SA\_RESETHAND  
49187 flag, SIGCHLD signals can be accurately counted; that is, exactly one SIGCHLD signal will  
49188 be delivered to or accepted by the process for every child process that terminates.
- 49189 2. A single *wait()* issued from a SIGCHLD signal handler can be guaranteed to return  
49190 immediately with status information for a child process.
- 49191 3. When SA\_SIGINFO is requested, the SIGCHLD signal handler can be guaranteed to  
49192 receive a non-NULL pointer to a **siginfo\_t** structure that describes a child process for  
49193 which a wait via *waitpid()* or *waitid()* will not block or fail.
- 49194 4. The *system()* function will not cause a process's SIGCHLD handler to be called as a result of  
49195 the *fork()/exec* executed within *system()* because *system()* will accept the SIGCHLD signal  
49196 when it performs a *waitpid()* for its child process. This is a desirable behavior of *system()*  
49197 so that it can be used in a library without causing side effects to the application linked with  
49198 the library.

49199 An implementation that does not permit the *wait()* family of functions to accept (discard) a  
49200 pending SIGCHLD signal associated with a successfully waited-for child, cannot make the  
49201 guarantees described above for the following reasons:

#### 49202 Guarantee #1

49203 Although it might be assumed that reliable queuing of all SIGCHLD signals generated by  
49204 the system can make this guarantee, the counter example is the case of a process that blocks  
49205 SIGCHLD and performs an indefinite loop of *fork()/wait()* operations. If the

49206 implementation supports queued signals, then eventually the system will run out of  
 49207 memory for the queue. The guarantee cannot be made because there must be some limit to  
 49208 the depth of queuing.

49209 Guarantees #2 and #3

49210 These cannot be guaranteed unless the *wait()* family of functions accepts the SIGCHLD  
 49211 signal. Otherwise, a *fork()/wait()* executed while SIGCHLD is blocked (as in the *system()*  
 49212 function) will result in an invocation of the handler when SIGCHLD is unblocked, after the  
 49213 process has disappeared.

49214 Guarantee #4

49215 Although possible to make this guarantee, *system()* would have to set the SIGCHLD  
 49216 handler to SIG\_DFL so that the SIGCHLD signal generated by its *fork()* would be discarded  
 49217 (the SIGCHLD default action is to be ignored), then restore it to its previous setting. This  
 49218 would have the undesirable side effect of discarding all SIGCHLD signals pending to the  
 49219 process.

#### 49220 FUTURE DIRECTIONS

49221 None.

#### 49222 SEE ALSO

49223 *exec*, *exit()*, *fork()*, *waitid()*, the Base Definitions volume of IEEE Std 1003.1-200x, <sys/types.h>,  
 49224 <sys/wait.h>

#### 49225 CHANGE HISTORY

49226 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 49227 Issue 5

49228 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

#### 49229 Issue 6

49230 In the SYNOPSIS, the optional include of the <sys/types.h> header is removed.

49231 The following new requirements on POSIX implementations derive from alignment with the  
 49232 Single UNIX Specification:

- 49233 • The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was  
 49234 required for conforming implementations of previous POSIX specifications, it was not  
 49235 required for UNIX applications.

49236 The following changes were made to align with the IEEE P1003.1a draft standard:

- 49237 • The processing of the SIGCHLD signal and the [ECHILD] error is clarified.

49238 The semantics of WIFSTOPPED(*stat\_val*), WIFEXITED(*stat\_val*), and WIFSIGNALED(*stat\_val*)  
 49239 are defined with respect to *posix\_spawn()* or *posix\_spawnnp()* for alignment with  
 49240 IEEE Std 1003.1d-1999.

49241 The DESCRIPTION is updated for alignment with the ISO/IEC 9899:1999 standard.

## 49242 NAME

49243 waitid — wait for a child process to change state

## 49244 SYNOPSIS

49245 XSI #include &lt;sys/wait.h&gt;

49246 int waitid(idtype\_t idtype, id\_t id, siginfo\_t \*infop, int options);

49247

## 49248 DESCRIPTION

49249 The *waitid()* function shall suspend the calling thread until one child of the process containing  
 49250 the calling thread changes state. It records the current state of a child in the structure pointed to  
 49251 by *infop*. If a child process changed state prior to the call to *waitid()*, *waitid()* shall return  
 49252 immediately. If more than one thread is suspended in *wait()* or *waitpid()* waiting termination of  
 49253 the same process, exactly one thread shall return the process status at the time of the target  
 49254 process termination.

49255 The *idtype* and *id* arguments are used to specify which children *waitid()* waits for.

49256 If *idtype* is P\_PID, *waitid()* shall wait for the child with a process ID equal to (**pid\_t**)*id*.

49257 If *idtype* is P\_PGID, *waitid()* shall wait for any child with a process group ID equal to (**pid\_t**)*id*.

49258 If *idtype* is P\_ALL, *waitid()* shall wait for any children and *id* is ignored.

49259 The *options* argument is used to specify which state changes *waitid()* shall wait for. It is formed  
 49260 by OR'ing together one or more of the following flags:

49261 WEXITED Wait for processes that have exited.

49262 WSTOPPED Status shall be returned for any child that has stopped upon receipt of a signal.

49263 WCONTINUED Status shall be returned for any child that was stopped and has been  
 49264 continued.

49265 WNOHANG Return immediately if there are no children to wait for.

49266 WNOWAIT Keep the process whose status is returned in *infop* in a waitable state. This  
 49267 shall not affect the state of the process; the process may be waited for again  
 49268 after this call completes.

49269 The application shall ensure that the *infop* argument points to a **siginfo\_t** structure. If *waitid()*  
 49270 returns because a child process was found that satisfied the conditions indicated by the  
 49271 arguments *idtype* and *options*, then the structure pointed to by *infop* shall be filled in by the  
 49272 system with the status of the process. The *si\_signo* member shall always be equal to SIGCHLD.

## 49273 RETURN VALUE

49274 If WNOHANG was specified and there are no children to wait for, 0 shall be returned. If *waitid()* |  
 49275 returns due to the change of state of one of its children, 0 shall be returned. Otherwise, -1 shall  
 49276 be returned and *errno* set to indicate the error.

## 49277 ERRORS

49278 The *waitid()* function shall fail if:

49279 [ECHILD] The calling process has no existing unwaited-for child processes.

49280 [EINTR] The *waitid()* function was interrupted by a signal.

49281 [EINVAL] An invalid value was specified for *options*, or *idtype* and *id* specify an invalid  
 49282 set of processes.

49283 **EXAMPLES**

49284 None.

49285 **APPLICATION USAGE**

49286 None.

49287 **RATIONALE**

49288 None.

49289 **FUTURE DIRECTIONS**

49290 None.

49291 **SEE ALSO**49292 *exec*, *exit()*, *wait()*, the Base Definitions volume of IEEE Std 1003.1-200x, <sys/wait.h>49293 **CHANGE HISTORY**

49294 First released in Issue 4, Version 2.

49295 **Issue 5**

49296 Moved from X/OPEN UNIX extension to BASE.

49297 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

49298 **Issue 6**

49299 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

49300 **NAME**

49301           waitpid — wait for a child process to stop or terminate

49302 **SYNOPSIS**

49303           #include <sys/wait.h>

49304           pid\_t waitpid(pid\_t *pid*, int *\*stat\_loc*, int *options*);

49305 **DESCRIPTION**

49306           Refer to *wait()*.

49307 **NAME**49308 `wrtomb` — convert a wide-character code to a character (restartable)49309 **SYNOPSIS**49310 `#include <stdio.h>`49311 `size_t wrtomb(char *restrict s, wchar_t wc, mbstate_t *restrict ps);`49312 **DESCRIPTION**

49313 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
 49314 conflict between the requirements described here and the ISO C standard is unintentional. This  
 49315 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

49316 If *s* is a null pointer, the `wrtomb()` function shall be equivalent to the call:49317 `wrtomb(buf, L'\0', ps)`49318 where *buf* is an internal buffer.

49319 If *s* is not a null pointer, the `wrtomb()` function shall determine the number of bytes needed to  
 49320 represent the character that corresponds to the wide character given by *wc* (including any shift  
 49321 sequences), and store the resulting bytes in the array whose first element is pointed to by *s*. At  
 49322 most {MB\_CUR\_MAX} bytes are stored. If *wc* is a null wide character, a null byte shall be stored,  
 49323 preceded by any shift sequence needed to restore the initial shift state. The resulting state  
 49324 described shall be the initial conversion state.

49325 If *ps* is a null pointer, the `wrtomb()` function shall use its own internal `mbstate_t` object, which is  
 49326 initialized at program start-up to the initial conversion state. Otherwise, the `mbstate_t` object  
 49327 pointed to by *ps* shall be used to completely describe the current conversion state of the  
 49328 associated character sequence. The implementation shall behave as if no function defined in this  
 49329 volume of IEEE Std 1003.1-200x calls `wrtomb()`.

49330 cx If the application uses any of the `_POSIX_THREAD_SAFE_FUNCTIONS` or `_POSIX_THREADS`  
 49331 functions, the application shall ensure that the `wrtomb()` function is called with a non-NULL *ps*  
 49332 argument.

49333 The behavior of this function shall be affected by the `LC_CTYPE` category of the current locale.49334 **RETURN VALUE**

49335 The `wrtomb()` function shall return the number of bytes stored in the array object (including any  
 49336 shift sequences). When *wc* is not a valid wide character, an encoding error shall occur. In this  
 49337 case, the function shall store the value of the macros [EILSEQ] in *errno* and shall return  
 49338 (`size_t`)-1; the conversion state shall be undefined.

49339 **ERRORS**49340 The `wrtomb()` function may fail if:49341 cx [EINVAL] *ps* points to an object that contains an invalid conversion state.

49342 [EILSEQ] Invalid wide-character code is detected.

49343 **EXAMPLES**

49344 None.

49345 **APPLICATION USAGE**

49346 None.

49347 **RATIONALE**

49348 None.

49349 **FUTURE DIRECTIONS**

49350 None.

49351 **SEE ALSO**49352 *mbstinit()*, the Base Definitions volume of IEEE Std 1003.1-200x, <**wchar.h**>49353 **CHANGE HISTORY**49354 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995  
49355 (E).49356 **Issue 6**

49357 In the DESCRIPTION, a note on using this function in a threaded application is added.

49358 Extensions beyond the ISO C standard are now marked.

49359 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

49360 The *wcrtomb()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

49361 **NAME**49362 `wscat` — concatenate two wide-character strings49363 **SYNOPSIS**49364 `#include <wchar.h>`49365 `wchar_t *wscat(wchar_t *restrict ws1, const wchar_t *restrict ws2);`49366 **DESCRIPTION**49367 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
49368 conflict between the requirements described here and the ISO C standard is unintentional. This  
49369 volume of IEEE Std 1003.1-200x defers to the ISO C standard.49370 The `wscat()` function shall append a copy of the wide-character string pointed to by `ws2`  
49371 (including the terminating null wide-character code) to the end of the wide-character string  
49372 pointed to by `ws1`. The initial wide-character code of `ws2` shall overwrite the null wide-character  
49373 code at the end of `ws1`. If copying takes place between objects that overlap, the behavior is  
49374 undefined.49375 **RETURN VALUE**49376 The `wscat()` function shall return `ws1`; no return value is reserved to indicate an error.49377 **ERRORS**

49378 No errors are defined.

49379 **EXAMPLES**

49380 None.

49381 **APPLICATION USAGE**

49382 None.

49383 **RATIONALE**

49384 None.

49385 **FUTURE DIRECTIONS**

49386 None.

49387 **SEE ALSO**49388 `wscncat()`, the Base Definitions volume of IEEE Std 1003.1-200x, `<wchar.h>`49389 **CHANGE HISTORY**

49390 First released in Issue 4. Derived from the MSE working draft.

49391 **Issue 6**49392 The Open Group Corrigendum U040/2 is applied. In the RETURN VALUE section, `s1` is changed  
49393 to `ws1`.49394 The `wscat()` prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

49395 **NAME**49396 `wchr` — wide-character string scanning operation49397 **SYNOPSIS**49398 `#include <wchar.h>`49399 `wchar_t *wchr(const wchar_t *ws, wchar_t wc);`49400 **DESCRIPTION**

49401 `cx` The functionality described on this reference page is aligned with the ISO C standard. Any  
49402 conflict between the requirements described here and the ISO C standard is unintentional. This  
49403 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

49404 The `wchr()` function shall locate the first occurrence of `wc` in the wide-character string pointed  
49405 to by `ws`. The application shall ensure that the value of `wc` is a character representable as a type  
49406 `wchar_t` and a wide-character code corresponding to a valid character in the current locale. The  
49407 terminating null wide-character code is considered to be part of the wide-character string.

49408 **RETURN VALUE**

49409 Upon completion, `wchr()` shall return a pointer to the wide-character code, or a null pointer if  
49410 the wide-character code is not found.

49411 **ERRORS**

49412 No errors are defined.

49413 **EXAMPLES**

49414 None.

49415 **APPLICATION USAGE**

49416 None.

49417 **RATIONALE**

49418 None.

49419 **FUTURE DIRECTIONS**

49420 None.

49421 **SEE ALSO**49422 `wchr()`, the Base Definitions volume of IEEE Std 1003.1-200x, `<wchar.h>`49423 **CHANGE HISTORY**

49424 First released in Issue 4. Derived from the MSE working draft.

49425 **Issue 6**

49426 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

49427 **NAME**49428        `wcsncmp` — compare two wide-character strings49429 **SYNOPSIS**49430        `#include <wchar.h>`49431        `int wcsncmp(const wchar_t *ws1, const wchar_t *ws2);`49432 **DESCRIPTION**49433 **CX**        The functionality described on this reference page is aligned with the ISO C standard. Any  
49434        conflict between the requirements described here and the ISO C standard is unintentional. This  
49435        volume of IEEE Std 1003.1-200x defers to the ISO C standard.49436        The `wcsncmp()` function shall compare the wide-character string pointed to by `ws1` to the wide-  
49437        character string pointed to by `ws2`.49438        The sign of a non-zero return value shall be determined by the sign of the difference between the  
49439        values of the first pair of wide-character codes that differ in the objects being compared. |49440 **RETURN VALUE**49441        Upon completion, `wcsncmp()` shall return an integer greater than, equal to, or less than 0, if the  
49442        wide-character string pointed to by `ws1` is greater than, equal to, or less than the wide-character  
49443        string pointed to by `ws2`, respectively.49444 **ERRORS**

49445        No errors are defined.

49446 **EXAMPLES**

49447        None.

49448 **APPLICATION USAGE**

49449        None.

49450 **RATIONALE**

49451        None.

49452 **FUTURE DIRECTIONS**

49453        None.

49454 **SEE ALSO**49455        `wcsncmp()`, the Base Definitions volume of IEEE Std 1003.1-200x, `<wchar.h>`49456 **CHANGE HISTORY**

49457        First released in Issue 4. Derived from the MSE working draft.

49458 **NAME**

49459 wscoll — wide-character string comparison using collating information

49460 **SYNOPSIS**

49461 #include &lt;wchar.h&gt;

49462 int wscoll(const wchar\_t \*ws1, const wchar\_t \*ws2);

49463 **DESCRIPTION**49464 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
49465 conflict between the requirements described here and the ISO C standard is unintentional. This  
49466 volume of IEEE Std 1003.1-200x defers to the ISO C standard.49467 The *wscoll()* function shall compare the wide-character string pointed to by *ws1* to the wide-  
49468 character string pointed to by *ws2*, both interpreted as appropriate to the *LC\_COLLATE* category  
49469 of the current locale.49470 CX The *wscoll()* function shall not change the setting of *errno* if successful.49471 An application wishing to check for error situations should set *errno* to 0 before calling *wscoll()*.  
49472 If *errno* is non-zero on return, an error has occurred.49473 **RETURN VALUE**49474 Upon successful completion, *wscoll()* shall return an integer greater than, equal to, or less than  
49475 0, according to whether the wide-character string pointed to by *ws1* is greater than, equal to, or  
49476 less than the wide-character string pointed to by *ws2*, when both are interpreted as appropriate  
49477 CX to the current locale. On error, *wscoll()* shall set *errno*, but no return value is reserved to  
49478 indicate an error.49479 **ERRORS**49480 The *wscoll()* function may fail if:49481 CX [EINVAL] The *ws1* or *ws2* arguments contain wide-character codes outside the domain of  
49482 the collating sequence.49483 **EXAMPLES**

49484 None.

49485 **APPLICATION USAGE**49486 The *wcsxfrm()* and *wscmp()* functions should be used for sorting large lists.49487 **RATIONALE**

49488 None.

49489 **FUTURE DIRECTIONS**

49490 None.

49491 **SEE ALSO**49492 *wscmp()*, *wcsxfrm()*, the Base Definitions volume of IEEE Std 1003.1-200x, <wchar.h>49493 **CHANGE HISTORY**

49494 First released in Issue 4. Derived from the MSE working draft.

49495 **Issue 5**

49496 Moved from ENHANCED I18N to BASE and the [ENOSYS] error is removed.

49497 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.

49498 **NAME**

49499           wcscpy — copy a wide-character string

49500 **SYNOPSIS**

49501           #include &lt;wchar.h&gt;

49502           wchar\_t \*wcscpy(wchar\_t \*restrict ws1, const wchar\_t \*restrict ws2);

49503 **DESCRIPTION**49504 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
49505           conflict between the requirements described here and the ISO C standard is unintentional. This  
49506           volume of IEEE Std 1003.1-200x defers to the ISO C standard.49507           The *wcscpy()* function shall copy the wide-character string pointed to by *ws2* (including the  
49508           terminating null wide-character code) into the array pointed to by *ws1*. If copying takes place  
49509           between objects that overlap, the behavior is undefined.49510 **RETURN VALUE**49511           The *wcscpy()* function shall return *ws1*; no return value is reserved to indicate an error.49512 **ERRORS**

49513           No errors are defined.

49514 **EXAMPLES**

49515           None.

49516 **APPLICATION USAGE**

49517           None.

49518 **RATIONALE**

49519           None.

49520 **FUTURE DIRECTIONS**

49521           None.

49522 **SEE ALSO**49523           *wscncpy()*, the Base Definitions volume of IEEE Std 1003.1-200x, <**wchar.h**>49524 **CHANGE HISTORY**

49525           First released in Issue 4. Derived from the MSE working draft.

49526 **Issue 6**49527           The *wcscpy()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

49528 **NAME**

49529 wscspn — get length of a complementary wide substring

49530 **SYNOPSIS**

49531 #include <wchar.h>

49532 size\_t wscspn(const wchar\_t \*ws1, const wchar\_t \*ws2);

49533 **DESCRIPTION**

49534 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
49535 conflict between the requirements described here and the ISO C standard is unintentional. This  
49536 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

49537 The *wscspn()* function shall compute the length (in wide characters) of the maximum initial |  
49538 segment of the wide-character string pointed to by *ws1* which consists entirely of wide-character |  
49539 codes *not* from the wide-character string pointed to by *ws2*.

49540 **RETURN VALUE**

49541 The *wscspn()* function shall return the length of the initial substring of *ws1*; no return value is  
49542 reserved to indicate an error.

49543 **ERRORS**

49544 No errors are defined.

49545 **EXAMPLES**

49546 None.

49547 **APPLICATION USAGE**

49548 None.

49549 **RATIONALE**

49550 None.

49551 **FUTURE DIRECTIONS**

49552 None.

49553 **SEE ALSO**

49554 *wcspn()*, the Base Definitions volume of IEEE Std 1003.1-200x, <wchar.h>

49555 **CHANGE HISTORY**

49556 First released in Issue 4. Derived from the MSE working draft.

49557 **Issue 5**

49558 The RETURN VALUE section is updated to indicate that *wscspn()* returns the length of *ws1*,  
49559 rather than *ws1* itself.

49560 **NAME**49561 `wcsftime` — convert date and time to a wide-character string49562 **SYNOPSIS**49563 `#include <wchar.h>`49564 `size_t wcsftime(wchar_t *restrict wcs, size_t maxsize,`  
49565 `const wchar_t *restrict format, const struct tm *restrict timeptr);` |49566 **DESCRIPTION**49567 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
49568 conflict between the requirements described here and the ISO C standard is unintentional. This  
49569 volume of IEEE Std 1003.1-200x defers to the ISO C standard.49570 The `wcsftime()` function shall be equivalent to the `strftime()` function, except that:

- 49571 • The argument `wcs` points to the initial element of an array of wide characters into which the  
49572 generated output is to be placed.
- 49573 • The argument `maxsize` indicates the maximum number of wide characters to be placed in the  
49574 output array.
- 49575 • The argument `format` is a wide-character string and the conversion specifications are replaced  
49576 by corresponding sequences of wide characters.
- 49577 • The return value indicates the number of wide characters placed in the output array.

49578 If copying takes place between objects that overlap, the behavior is undefined.

49579 **RETURN VALUE**49580 If the total number of resulting wide-character codes including the terminating null wide-  
49581 character code is no more than `maxsize`, `wcsftime()` shall return the number of wide-character  
49582 codes placed into the array pointed to by `wcs`, not including the terminating null wide-character  
49583 code. Otherwise, zero is returned and the contents of the array are unspecified. |49584 **ERRORS**

49585 No errors are defined.

49586 **EXAMPLES**

49587 None.

49588 **APPLICATION USAGE**

49589 None.

49590 **RATIONALE**

49591 None.

49592 **FUTURE DIRECTIONS**

49593 None.

49594 **SEE ALSO**49595 `strftime()`, the Base Definitions volume of IEEE Std 1003.1-200x, `<wchar.h>`49596 **CHANGE HISTORY**

49597 First released in Issue 4.

49598 **Issue 5**

49599 Moved from ENHANCED I18N to BASE and the [ENOSYS] error is removed.

49600 Aligned with ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, the type of the `format`  
49601 argument is changed from `const char *` to `const wchar_t *`.

49602 **Issue 6**

49603

The *wcsftime()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

49604 **NAME**

49605           wcslen — get wide-character string length

49606 **SYNOPSIS**

49607           #include &lt;wchar.h&gt;

49608           size\_t wcslen(const wchar\_t \*ws);

49609 **DESCRIPTION**

49610 cx       The functionality described on this reference page is aligned with the ISO C standard. Any  
49611       conflict between the requirements described here and the ISO C standard is unintentional. This  
49612       volume of IEEE Std 1003.1-200x defers to the ISO C standard.

49613       The *wcslen()* function shall compute the number of wide-character codes in the wide-character  
49614       string to which *ws* points, not including the terminating null wide-character code.

49615 **RETURN VALUE**

49616       The *wcslen()* function shall return the length of *ws*; no return value is reserved to indicate an  
49617       error.

49618 **ERRORS**

49619       No errors are defined.

49620 **EXAMPLES**

49621       None.

49622 **APPLICATION USAGE**

49623       None.

49624 **RATIONALE**

49625       None.

49626 **FUTURE DIRECTIONS**

49627       None.

49628 **SEE ALSO**49629       The Base Definitions volume of IEEE Std 1003.1-200x, <**wchar.h**>49630 **CHANGE HISTORY**

49631       First released in Issue 4. Derived from the MSE working draft.

49632 **NAME**

49633       wcsncat — concatenate a wide-character string with part of another

49634 **SYNOPSIS**

49635       #include &lt;wchar.h&gt;

49636       wchar\_t \*wcsncat(wchar\_t \*restrict ws1, const wchar\_t \*restrict ws2,  
49637                       size\_t n);49638 **DESCRIPTION**49639 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
49640 conflict between the requirements described here and the ISO C standard is unintentional. This  
49641 volume of IEEE Std 1003.1-200x defers to the ISO C standard.49642       The *wcsncat()* function shall append not more than *n* wide-character codes (a null wide-  
49643 character code and wide-character codes that follow it are not appended) from the array pointed  
49644 to by *ws2* to the end of the wide-character string pointed to by *ws1*. The initial wide-character  
49645 code of *ws2* shall overwrite the null wide-character code at the end of *ws1*. A terminating null  
49646 wide-character code shall always be appended to the result. If copying takes place between |  
49647 objects that overlap, the behavior is undefined.49648 **RETURN VALUE**49649       The *wcsncat()* function shall return *ws1*; no return value is reserved to indicate an error.49650 **ERRORS**

49651       No errors are defined.

49652 **EXAMPLES**

49653       None.

49654 **APPLICATION USAGE**

49655       None.

49656 **RATIONALE**

49657       None.

49658 **FUTURE DIRECTIONS**

49659       None.

49660 **SEE ALSO**49661       *wscat()*, the Base Definitions volume of IEEE Std 1003.1-200x, <**wchar.h**>49662 **CHANGE HISTORY**

49663       First released in Issue 4. Derived from the MSE working draft.

49664 **Issue 6**49665       The *wcsncat()* prototype is updated for alignment with the ISO/IEC 9899: 1999 standard.

49666 **NAME**49667        `wcsncmp` — compare part of two wide-character strings49668 **SYNOPSIS**49669        `#include <wchar.h>`49670        `int wcsncmp(const wchar_t *ws1, const wchar_t *ws2, size_t n);`49671 **DESCRIPTION**49672 **CX**        The functionality described on this reference page is aligned with the ISO C standard. Any  
49673 conflict between the requirements described here and the ISO C standard is unintentional. This  
49674 volume of IEEE Std 1003.1-200x defers to the ISO C standard.49675        The `wcsncmp()` function shall compare not more than *n* wide-character codes (wide-character  
49676 codes that follow a null wide-character code are not compared) from the array pointed to by *ws1*  
49677 to the array pointed to by *ws2*.49678        The sign of a non-zero return value shall be determined by the sign of the difference between the  
49679 values of the first pair of wide-character codes that differ in the objects being compared. |49680 **RETURN VALUE**49681        Upon successful completion, `wcsncmp()` shall return an integer greater than, equal to, or less  
49682 than 0, if the possibly null-terminated array pointed to by *ws1* is greater than, equal to, or less  
49683 than the possibly null-terminated array pointed to by *ws2*, respectively.49684 **ERRORS**

49685        No errors are defined.

49686 **EXAMPLES**

49687        None.

49688 **APPLICATION USAGE**

49689        None.

49690 **RATIONALE**

49691        None.

49692 **FUTURE DIRECTIONS**

49693        None.

49694 **SEE ALSO**49695        `wscmp()`, the Base Definitions volume of IEEE Std 1003.1-200x, `<wchar.h>`49696 **CHANGE HISTORY**

49697        First released in Issue 4. Derived from the MSE working draft.

49698 **NAME**49699 `wcsncpy` — copy part of a wide-character string49700 **SYNOPSIS**49701 `#include <wchar.h>`49702 `wchar_t *wcsncpy(wchar_t *restrict ws1, const wchar_t *restrict ws2,`  
49703 `size_t n);`49704 **DESCRIPTION**49705 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
49706 conflict between the requirements described here and the ISO C standard is unintentional. This  
49707 volume of IEEE Std 1003.1-200x defers to the ISO C standard.49708 The `wcsncpy()` function shall copy not more than *n* wide-character codes (wide-character codes  
49709 that follow a null wide-character code are not copied) from the array pointed to by *ws2* to the  
49710 array pointed to by *ws1*. If copying takes place between objects that overlap, the behavior is  
49711 undefined.49712 If the array pointed to by *ws2* is a wide-character string that is shorter than *n* wide-character |  
49713 codes, null wide-character codes shall be appended to the copy in the array pointed to by *ws1*, |  
49714 until *n* wide-character codes in all are written.49715 **RETURN VALUE**49716 The `wcsncpy()` function shall return *ws1*; no return value is reserved to indicate an error.49717 **ERRORS**

49718 No errors are defined.

49719 **EXAMPLES**

49720 None.

49721 **APPLICATION USAGE**49722 If there is no null wide-character code in the first *n* wide-character codes of the array pointed to  
49723 by *ws2*, the result is not null-terminated.49724 **RATIONALE**

49725 None.

49726 **FUTURE DIRECTIONS**

49727 None.

49728 **SEE ALSO**49729 `wscpy()`, the Base Definitions volume of IEEE Std 1003.1-200x, `<wchar.h>`49730 **CHANGE HISTORY**

49731 First released in Issue 4. Derived from the MSE working draft.

49732 **Issue 6**49733 The `wcsncpy()` prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

49734 **NAME**

49735 wcpbrk — scan wide-character string for a wide-character code

49736 **SYNOPSIS**

49737 #include &lt;wchar.h&gt;

49738 wchar\_t \*wcpbrk(const wchar\_t \*ws1, const wchar\_t \*ws2);

49739 **DESCRIPTION**49740 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
49741 conflict between the requirements described here and the ISO C standard is unintentional. This  
49742 volume of IEEE Std 1003.1-200x defers to the ISO C standard.49743 The *wcpbrk()* function shall locate the first occurrence in the wide-character string pointed to by  
49744 *ws1* of any wide-character code from the wide-character string pointed to by *ws2*.49745 **RETURN VALUE**49746 Upon successful completion, *wcpbrk()* shall return a pointer to the wide-character code or a null  
49747 pointer if no wide-character code from *ws2* occurs in *ws1*.49748 **ERRORS**

49749 No errors are defined.

49750 **EXAMPLES**

49751 None.

49752 **APPLICATION USAGE**

49753 None.

49754 **RATIONALE**

49755 None.

49756 **FUTURE DIRECTIONS**

49757 None.

49758 **SEE ALSO**49759 *wchr()*, *wchr()*, the Base Definitions volume of IEEE Std 1003.1-200x, <wchar.h>49760 **CHANGE HISTORY**

49761 First released in Issue 4. Derived from the MSE working draft.

49762 **NAME**

49763       wcsrchr — wide-character string scanning operation

49764 **SYNOPSIS**

49765       #include &lt;wchar.h&gt;

49766       wchar\_t \*wcsrchr(const wchar\_t \*ws, wchar\_t wc);

49767 **DESCRIPTION**49768 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
49769 conflict between the requirements described here and the ISO C standard is unintentional. This  
49770 volume of IEEE Std 1003.1-200x defers to the ISO C standard.49771       The *wcsrchr()* function shall locate the last occurrence of *wc* in the wide-character string pointed  
49772 to by *ws*. The application shall ensure that the value of *wc* is a character representable as a type  
49773 **wchar\_t** and a wide-character code corresponding to a valid character in the current locale. The  
49774 terminating null wide-character code shall be considered to be part of the wide-character string. |49775 **RETURN VALUE**49776       Upon successful completion, *wcsrchr()* shall return a pointer to the wide-character code or a null  
49777 pointer if *wc* does not occur in the wide-character string.49778 **ERRORS**

49779       No errors are defined.

49780 **EXAMPLES**

49781       None.

49782 **APPLICATION USAGE**

49783       None.

49784 **RATIONALE**

49785       None.

49786 **FUTURE DIRECTIONS**

49787       None.

49788 **SEE ALSO**49789       *wchr()*, the Base Definitions volume of IEEE Std 1003.1-200x, <**wchar.h**>49790 **CHANGE HISTORY**

49791       First released in Issue 4. Derived from the MSE working draft.

49792 **Issue 6**

49793       The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

## 49794 NAME

49795       wcsrtoombs — convert a wide-character string to a character string (restartable)

## 49796 SYNOPSIS

49797       #include <wchar.h>

49798       size\_t wcsrtoombs(char \*restrict dst, const wchar\_t \*\*restrict src,  
49799                       size\_t len, mbstate\_t \*restrict ps);

## 49800 DESCRIPTION

49801 cx     The functionality described on this reference page is aligned with the ISO C standard. Any  
49802     conflict between the requirements described here and the ISO C standard is unintentional. This  
49803     volume of IEEE Std 1003.1-200x defers to the ISO C standard.

49804     The *wcsrtoombs()* function shall convert a sequence of wide characters from the array indirectly  
49805     pointed to by *src* into a sequence of corresponding characters, beginning in the conversion state  
49806     described by the object pointed to by *ps*. If *dst* is not a null pointer, the converted characters  
49807     shall then be stored into the array pointed to by *dst*. Conversion continues up to and including a  
49808     terminating null wide character, which shall also be stored. Conversion shall stop earlier in the  
49809     following cases:

- 49810       • When a code is reached that does not correspond to a valid character
- 49811       • When the next character would exceed the limit of *len* total bytes to be stored in the array  
49812       pointed to by *dst* (and *dst* is not a null pointer)

49813     Each conversion shall take place as if by a call to the *wcrtomb()* function.

49814     If *dst* is not a null pointer, the pointer object pointed to by *src* shall be assigned either a null  
49815     pointer (if conversion stopped due to reaching a terminating null wide character) or the address  
49816     just past the last wide character converted (if any). If conversion stopped due to reaching a  
49817     terminating null wide character, the resulting state described shall be the initial conversion state.

49818     If *ps* is a null pointer, the *wcsrtoombs()* function shall use its own internal **mbstate\_t** object, which  
49819     is initialized at program start-up to the initial conversion state. Otherwise, the **mbstate\_t** object  
49820     pointed to by *ps* shall be used to completely describe the current conversion state of the  
49821     associated character sequence. The implementation shall behave as if no function defined in this  
49822     volume of IEEE Std 1003.1-200x calls *wcsrtoombs()*.

49823 cx     If the application uses any of the `_POSIX_THREAD_SAFE_FUNCTIONS` or `_POSIX_THREADS`  
49824     functions, the application shall ensure that the *wcsrtoombs()* function is called with a non-NULL  
49825     *ps* argument.

49826     The behavior of this function shall be affected by the *LC\_CTYPE* category of the current locale.

## 49827 RETURN VALUE

49828     If conversion stops because a code is reached that does not correspond to a valid character, an  
49829     encoding error occurs. In this case, the *wcsrtoombs()* function shall store the value of the macro  
49830     [EILSEQ] in *errno* and return (**size\_t**)–1; the conversion state is undefined. Otherwise, it shall  
49831     return the number of bytes in the resulting character sequence, not including the terminating  
49832     null (if any).

## 49833 ERRORS

49834     The *wcsrtoombs()* function may fail if:

- 49835 cx     [EINVAL]       *ps* points to an object that contains an invalid conversion state.
- 49836     [EILSEQ]       A wide-character code does not correspond to a valid character.

49837 **EXAMPLES**

49838 None.

49839 **APPLICATION USAGE**

49840 None.

49841 **RATIONALE**

49842 None.

49843 **FUTURE DIRECTIONS**

49844 None.

49845 **SEE ALSO**49846 *mbsinit()*, *wcrtomb()*, the Base Definitions volume of IEEE Std 1003.1-200x, <**wchar.h**>49847 **CHANGE HISTORY**49848 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995  
49849 (E).49850 **Issue 6**

49851 In the DESCRIPTION, a note on using this function in a threaded application is added.

49852 Extensions beyond the ISO C standard are now marked.

49853 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

49854 The *wcsrombs()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

49855 **NAME**49856 `wcsspn` — get length of a wide substring49857 **SYNOPSIS**49858 `#include <wchar.h>`49859 `size_t wcsspn(const wchar_t *ws1, const wchar_t *ws2);`49860 **DESCRIPTION**49861 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
49862 conflict between the requirements described here and the ISO C standard is unintentional. This  
49863 volume of IEEE Std 1003.1-200x defers to the ISO C standard.49864 The `wcsspn()` function shall compute the length (in wide characters) of the maximum initial |  
49865 segment of the wide-character string pointed to by `ws1` which consists entirely of wide-character |  
49866 codes from the wide-character string pointed to by `ws2`.49867 **RETURN VALUE**49868 The `wcsspn()` function shall return the length of the initial substring of `ws1`; no return value is  
49869 reserved to indicate an error.49870 **ERRORS**

49871 No errors are defined.

49872 **EXAMPLES**

49873 None.

49874 **APPLICATION USAGE**

49875 None.

49876 **RATIONALE**

49877 None.

49878 **FUTURE DIRECTIONS**

49879 None.

49880 **SEE ALSO**49881 `wcscspn()`, the Base Definitions volume of IEEE Std 1003.1-200x, `<wchar.h>`49882 **CHANGE HISTORY**

49883 First released in Issue 4. Derived from the MSE working draft.

49884 **Issue 5**49885 The RETURN VALUE section is updated to indicate that `wcsspn()` returns the length of `ws1`  
49886 rather than `ws1` itself.

49887 **NAME**49888 `wcsstr` — find a wide-character substring49889 **SYNOPSIS**49890 `#include <wchar.h>`49891 `wchar_t *wcsstr(const wchar_t *restrict ws1, const wchar_t *restrict ws2);`49892 **DESCRIPTION**

49893 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
49894 conflict between the requirements described here and the ISO C standard is unintentional. This  
49895 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

49896 The `wcsstr()` function shall locate the first occurrence in the wide-character string pointed to by  
49897 `ws1` of the sequence of wide characters (excluding the terminating null wide character) in the  
49898 wide-character string pointed to by `ws2`.

49899 **RETURN VALUE**

49900 Upon successful completion, `wcsstr()` shall return a pointer to the located wide-character string,  
49901 or a null pointer if the wide-character string is not found.

49902 If `ws2` points to a wide-character string with zero length, the function shall return `ws1`.

49903 **ERRORS**

49904 No errors are defined.

49905 **EXAMPLES**

49906 None.

49907 **APPLICATION USAGE**

49908 None.

49909 **RATIONALE**

49910 None.

49911 **FUTURE DIRECTIONS**

49912 None.

49913 **SEE ALSO**49914 `wcschr()`, the Base Definitions volume of IEEE Std 1003.1-200x, `<wchar.h>`49915 **CHANGE HISTORY**

49916 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995  
49917 (E).

49918 **Issue 6**49919 The `wcsstr()` prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

## 49920 NAME

49921 `wcstod`, `wcstof`, `wcstold` — convert a wide-character string to a double-precision number

## 49922 SYNOPSIS

49923 `#include <wchar.h>`

49924 `double wcstod(const wchar_t *restrict nptr, wchar_t **restrict endptr);`

49925 `float wcstof(const wchar_t *restrict nptr, wchar_t **restrict endptr);`

49926 `long double wcstold(const wchar_t *restrict nptr,`

49927 `wchar_t **restrict endptr);`

## 49928 DESCRIPTION

49929 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 49930 conflict between the requirements described here and the ISO C standard is unintentional. This  
 49931 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

49932 These functions shall convert the initial portion of the wide-character string pointed to by *nptr* to  
 49933 **double**, **float**, and **long double** representation, respectively. First, they shall decompose the  
 49934 input wide-character string into three parts:

- 49935 1. An initial, possibly empty, sequence of white-space wide-character codes (as specified by  
 49936 `iswspace()`)
- 49937 2. A subject sequence interpreted as a floating-point constant or representing infinity or NaN
- 49938 3. A final wide-character string of one or more unrecognized wide-character codes, including  
 49939 the terminating null wide-character code of the input wide-character string

49940 Then they shall attempt to convert the subject sequence to a floating-point number, and return  
 49941 the result.

49942 The expected form of the subject sequence is an optional plus or minus sign, then one of the  
 49943 following:

- 49944 • A non-empty sequence of decimal digits optionally containing a radix character, then an  
 49945 optional exponent part
- 49946 • A `0x` or `0X`, then a non-empty sequence of hexadecimal digits optionally containing a radix  
 49947 character, then an optional binary exponent part
- 49948 • One of `INF` or `INFINITY`, or any other wide string equivalent except for case
- 49949 • One of `NAN` or `NAN(n-wchar-sequenceopt)`, or any other wide string ignoring case in the `NAN`  
 49950 part, where:

49951 `n-wchar-sequence:`

49952 `digit`

49953 `nondigit`

49954 `n-wchar-sequence digit`

49955 `n-wchar-sequence nondigit`

49956 The subject sequence is defined as the longest initial subsequence of the input wide string,  
 49957 starting with the first non-white-space wide character, that is of the expected form. The subject  
 49958 sequence contains no wide characters if the input wide string is not of the expected form.

49959 If the subject sequence has the expected form for a floating-point number, the sequence of wide  
 49960 characters starting with the first digit or the radix character (whichever occurs first) shall be  
 49961 interpreted as a floating constant according to the rules of the C language, except that the radix  
 49962 character shall be used in place of a period, and that if neither an exponent part nor a radix  
 49963 character appears in a decimal floating-point number, or if a binary exponent part does not

- 49964 appear in a hexadecimal floating-point number, an exponent part of the appropriate type with  
 49965 value zero shall be assumed to follow the last digit in the string. If the subject sequence begins  
 49966 with a minus sign, the sequence shall be interpreted as negated. A wide-character sequence INF  
 49967 or INFINITY shall be interpreted as an infinity, if representable in the return type, else as if it  
 49968 were a floating constant that is too large for the range of the return type. A wide-character  
 49969 sequence NAN or NAN(*n-wchar-sequence<sub>opt</sub>*) shall be interpreted as a quiet NaN, if supported in  
 49970 the return type, else as if it were a subject sequence part that does not have the expected form;  
 49971 the meaning of the *n-wchar* sequences is implementation-defined. A pointer to the final wide  
 49972 string shall be stored in the object pointed to by *endptr*, provided that *endptr* is not a null pointer.
- 49973 If the subject sequence has the hexadecimal form and FLT\_RADIX is a power of 2, the  
 49974 conversion shall be rounded in an implementation-defined manner.
- 49975 CX The radix character shall be as defined in the program's locale (category *LC\_NUMERIC*). In the  
 49976 POSIX locale, or in a locale where the radix character is not defined, the radix character shall  
 49977 default to a period ( ' . ' ).
- 49978 CX In other than the C or POSIX locales, other implementation-defined subject sequences may be  
 49979 accepted.
- 49980 If the subject sequence is empty or does not have the expected form, no conversion shall be  
 49981 performed; the value of *nptr* shall be stored in the object pointed to by *endptr*, provided that  
 49982 *endptr* is not a null pointer.
- 49983 CX The *wcstod()* function shall not change the setting of *errno* if successful.
- 49984 Since 0 is returned on error and is also a valid return on success, an application wishing to check  
 49985 for error situations should set *errno* to 0, then call *wcstod()*, *wcstof()*, or *wcstold()*, then check  
 49986 *errno*.
- 49987 **RETURN VALUE**
- 49988 Upon successful completion, these functions shall return the converted value. If no conversion  
 49989 CX could be performed, 0 shall be returned and *errno* may be set to [EINVAL].
- 49990 If the correct value is outside the range of representable values, plus or minus HUGE\_VAL,  
 49991 HUGE\_VALF, or HUGE\_VALL shall be returned (according to the sign of the value), and *errno*  
 49992 shall be set to [ERANGE].
- 49993 If the correct value would cause underflow, a value whose magnitude is no greater than the  
 49994 smallest normalized positive number in the return type shall be returned and *errno* set to  
 49995 [ERANGE].
- 49996 **ERRORS**
- 49997 The *wcstod()* function shall fail if:
- 49998 [ERANGE] The value to be returned would cause overflow or underflow.
- 49999 The *wcstod()* function may fail if:
- 50000 CX [EINVAL] No conversion could be performed.

50001 **EXAMPLES**

50002 None.

50003 **APPLICATION USAGE**

50004 If the subject sequence has the hexadecimal form and FLT\_RADIX is not a power of 2, and the  
 50005 result is not exactly representable, the result should be one of the two numbers in the  
 50006 appropriate internal format that are adjacent to the hexadecimal floating source value, with the  
 50007 extra stipulation that the error should have a correct sign for the current rounding direction.

50008 If the subject sequence has the decimal form and at most DECIMAL\_DIG (defined in <float.h>)  
 50009 significant digits, the result should be correctly rounded. If the subject sequence *D* has the  
 50010 decimal form and more than DECIMAL\_DIG significant digits, consider the two bounding,  
 50011 adjacent decimal strings *L* and *U*, both having DECIMAL\_DIG significant digits, such that the  
 50012 values of *L*, *D*, and *U* satisfy " $L \leq D \leq U$ ". The result should be one of the (equal or  
 50013 adjacent) values that would be obtained by correctly rounding *L* and *U* according to the current  
 50014 rounding direction, with the extra stipulation that the error with respect to *D* should have a  
 50015 correct sign for the current rounding direction.

50016 **RATIONALE**

50017 None.

50018 **FUTURE DIRECTIONS**

50019 None.

50020 **SEE ALSO**

50021 *iswspace()*, *localeconv()*, *scanf()*, *setlocale()*, *wcstol()*, the Base Definitions volume of  
 50022 IEEE Std 1003.1-200x, <float.h>, <wchar.h>, the Base Definitions volume of  
 50023 IEEE Std 1003.1-200x, Chapter 7, Locale

50024 **CHANGE HISTORY**

50025 First released in Issue 4. Derived from the MSE working draft.

50026 **Issue 5**50027 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.50028 **Issue 6**

50029 Extensions beyond the ISO C standard are now marked.

50030 The following new requirements on POSIX implementations derive from alignment with the  
 50031 Single UNIX Specification:

50032 • In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is  
 50033 added if no conversion could be performed.

50034 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

50035 • The *wcstod()* prototype is updated.50036 • The *wcstof()* and *wcstold()* functions are added.

50037 • If the correct value for *wcstod()* would cause underflow, the return value changed from 0 (as  
 50038 specified in Issue 5) to the smallest normalized positive number.

50039 • The DESCRIPTION, RETURN VALUE, and APPLICATION USAGE sections are extensively  
 50040 updated.

50041 ISO/IEC 9899:1999 standard, Technical Corrigendum No. 1 is incorporated.

50042 **NAME**

50043           wcstof — convert a wide-character string to a double-precision number

50044 **SYNOPSIS**

50045           #include &lt;wchar.h&gt;

50046           float wcstof(const wchar\_t \*restrict *nptr*, wchar\_t \*\*restrict *endptr*);50047 **DESCRIPTION**50048           Refer to *wcstod()*.

50049 **NAME**50050 `wcstoimax`, `wcstoumax` — convert wide-character string to integer type50051 **SYNOPSIS**50052 `#include <stddef.h>`50053 `#include <inttypes.h>`50054 `intmax_t wcstoimax(const wchar_t *restrict nptr,`50055 `wchar_t **restrict endpnr, int base);`50056 `uintmax_t wcstoumax(const wchar_t *restrict nptr,`50057 `wchar_t **restrict endpnr, int base);`50058 **DESCRIPTION**50059 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
50060 conflict between the requirements described here and the ISO C standard is unintentional. This  
50061 volume of IEEE Std 1003.1-200x defers to the ISO C standard.50062 These functions shall be equivalent to the `wcstol()`, `wcstoll()`, `wcstoul()`, and `wcstoull()` functions,  
50063 respectively, except that the initial portion of the wide string shall be converted to **intmax\_t** and  
50064 **uintmax\_t** representation, respectively.50065 **RETURN VALUE**

50066 These functions shall return the converted value, if any.

50067 If no conversion could be performed, zero shall be returned. If the correct value is outside the  
50068 range of representable values, {INTMAX\_MAX}, {INTMAX\_MIN}, or {UINTMAX\_MAX} shall  
50069 be returned (according to the return type and sign of the value, if any), and `errno` shall be set to  
50070 [ERANGE].50071 **ERRORS**

50072 These functions shall fail if:

50073 [EINVAL] The value of *base* is not supported.

50074 [ERANGE] The value to be returned is not representable.

50075 These functions may fail if:

50076 [EINVAL] No conversion could be performed.

50077 **EXAMPLES**

50078 None.

50079 **APPLICATION USAGE**

50080 None.

50081 **RATIONALE**

50082 None.

50083 **FUTURE DIRECTIONS**

50084 None.

50085 **SEE ALSO**50086 `wcstol()`, `wcstoul()`, the Base Definitions volume of IEEE Std 1003.1-200x, `<inttypes.h>`,50087 `<stddef.h>`50088 **CHANGE HISTORY**

50089 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

50090 **NAME**

50091 wcstok — split wide-character string into tokens

50092 **SYNOPSIS**

50093 #include &lt;wchar.h&gt;

50094 wchar\_t \*wcstok(wchar\_t \*restrict ws1, const wchar\_t \*restrict ws2,  
50095 wchar\_t \*\*restrict ptr);50096 **DESCRIPTION**50097 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
50098 conflict between the requirements described here and the ISO C standard is unintentional. This  
50099 volume of IEEE Std 1003.1-200x defers to the ISO C standard.50100 A sequence of calls to *wcstok()* shall break the wide-character string pointed to by *ws1* into a |  
50101 sequence of tokens, each of which shall be delimited by a wide-character code from the wide- |  
50102 character string pointed to by *ws2*. The *ptr* argument points to a caller-provided **wchar\_t** pointer |  
50103 into which the *wcstok()* function shall store information necessary for it to continue scanning the |  
50104 same wide-character string. |50105 The first call in the sequence has *ws1* as its first argument, and is followed by calls with a null |  
50106 pointer as their first argument. The separator string pointed to by *ws2* may be different from call |  
50107 to call.50108 The first call in the sequence shall search the wide-character string pointed to by *ws1* for the first |  
50109 wide-character code that is *not* contained in the current separator string pointed to by *ws2*. If no |  
50110 such wide-character code is found, then there are no tokens in the wide-character string pointed |  
50111 to by *ws1* and *wcstok()* shall return a null pointer. If such a wide-character code is found, it shall |  
50112 be the start of the first token. |50113 The *wcstok()* function shall then search from there for a wide-character code that *is* contained in |  
50114 the current separator string. If no such wide-character code is found, the current token extends |  
50115 to the end of the wide-character string pointed to by *ws1*, and subsequent searches for a token |  
50116 shall return a null pointer. If such a wide-character code is found, it shall be overwritten by a |  
50117 null wide character, which terminates the current token. The *wcstok()* function shall save a |  
50118 pointer to the following wide-character code, from which the next search for a token shall start. |50119 Each subsequent call, with a null pointer as the value of the first argument, shall start searching |  
50120 from the saved pointer and behave as described above. |50121 The implementation shall behave as if no function calls *wcstok()*.50122 **RETURN VALUE**50123 Upon successful completion, the *wcstok()* function shall return a pointer to the first wide- |  
50124 character code of a token. Otherwise, if there is no token, *wcstok()* shall return a null pointer.50125 **ERRORS**

50126 No errors are defined.

50127 **EXAMPLES**

50128 None.

50129 **APPLICATION USAGE**

50130 None.

50131 **RATIONALE**

50132 None.

50133 **FUTURE DIRECTIONS**

50134 None.

50135 **SEE ALSO**

50136 The Base Definitions volume of IEEE Std 1003.1-200x, &lt;wchar.h&gt;

50137 **CHANGE HISTORY**

50138 First released in Issue 4.

50139 **Issue 5**50140 Aligned with ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, a third argument is  
50141 added to the definition of this function in the SYNOPSIS.50142 **Issue 6**50143 The *wcstok()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

## 50144 NAME

50145 wcstol, wcstoll — convert a wide-character string to a long integer

## 50146 SYNOPSIS

50147 #include &lt;wchar.h&gt;

50148 long wcstol(const wchar\_t \*restrict *nptr*, wchar\_t \*\*restrict *endptr*,  
50149 int *base*);50150 long long wcstoll(const wchar\_t \*restrict *nptr*,  
50151 wchar\_t \*\*restrict *endptr*, int *base*);

## 50152 DESCRIPTION

50153 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
50154 conflict between the requirements described here and the ISO C standard is unintentional. This  
50155 volume of IEEE Std 1003.1-200x defers to the ISO C standard.50156 These functions shall convert the initial portion of the wide-character string pointed to by *nptr* to  
50157 **long**, **long long**, **unsigned long**, and **unsigned long long** representation, respectively. First, they  
50158 shall decompose the input string into three parts: |

- 50159 1. An initial, possibly empty, sequence of white-space wide-character codes (as specified by  
50160 *iswspace()*)
- 50161 2. A subject sequence interpreted as an integer represented in some radix determined by the  
50162 value of *base*
- 50163 3. A final wide-character string of one or more unrecognized wide-character codes, including  
50164 the terminating null wide-character code of the input wide-character string

50165 Then they shall attempt to convert the subject sequence to an integer, and return the result. |

50166 If *base* is 0, the expected form of the subject sequence is that of a decimal constant, octal constant,  
50167 or hexadecimal constant, any of which may be preceded by a '+' or '-' sign. A decimal  
50168 constant begins with a non-zero digit, and consists of a sequence of decimal digits. An octal  
50169 constant consists of the prefix '0' optionally followed by a sequence of the digits '0' to '7'  
50170 only. A hexadecimal constant consists of the prefix 0x or 0X followed by a sequence of the  
50171 decimal digits and letters 'a' (or 'A') to 'f' (or 'F') with values 10 to 15 respectively.50172 If the value of *base* is between 2 and 36, the expected form of the subject sequence is a sequence  
50173 of letters and digits representing an integer with the radix specified by *base*, optionally preceded  
50174 by a '+' or '-' sign, but not including an integer suffix. The letters from 'a' (or 'A') to 'z'  
50175 (or 'Z') inclusive are ascribed the values 10 to 35; only letters whose ascribed values are less  
50176 than that of *base* shall be permitted. If the value of *base* is 16, the wide-character code  
50177 representations of 0x or 0X may optionally precede the sequence of letters and digits, following  
50178 the sign if present. |50179 The subject sequence is defined as the longest initial subsequence of the input wide-character  
50180 string, starting with the first non-white-space wide-character code that is of the expected form.  
50181 The subject sequence contains no wide-character codes if the input wide-character string is  
50182 empty or consists entirely of white-space wide-character code, or if the first non-white-space  
50183 wide-character code is other than a sign or a permissible letter or digit.50184 If the subject sequence has the expected form and *base* is 0, the sequence of wide-character codes  
50185 starting with the first digit shall be interpreted as an integer constant. If the subject sequence has  
50186 the expected form and the value of *base* is between 2 and 36, it shall be used as the base for  
50187 conversion, ascribing to each letter its value as given above. If the subject sequence begins with a  
50188 minus sign, the value resulting from the conversion shall be negated. A pointer to the final  
50189 wide-character string shall be stored in the object pointed to by *endptr*, provided that *endptr* is |

- 50190 not a null pointer.
- 50191 CX In other than the C or POSIX locales, other implementation-defined subject sequences may be  
50192 accepted.
- 50193 If the subject sequence is empty or does not have the expected form, no conversion shall be |  
50194 performed; the value of *nptr* shall be stored in the object pointed to by *endptr*, provided that |  
50195 *endptr* is not a null pointer.
- 50196 CX These functions shall not change the setting of *errno* if successful.
- 50197 Since 0, {LONG\_MIN} or {LLONG\_MIN} and {LONG\_MAX} or {LLONG\_MAX} are returned on |  
50198 error and are also valid returns on success, an application wishing to check for error situations  
50199 should set *errno* to 0, then call *wcstol()* or *wcstoll()*, then check *errno*.
- 50200 **RETURN VALUE**
- 50201 Upon successful completion, these functions shall return the converted value, if any. If no  
50202 CX conversion could be performed, 0 shall be returned and *errno* may be set to indicate the error. If  
50203 the correct value is outside the range of representable values, {LONG\_MIN}, {LONG\_MAX},  
50204 {LLONG\_MIN}, or {LLONG\_MAX} shall be returned (according to the sign of the value), and  
50205 *errno* set to [ERANGE].
- 50206 **ERRORS**
- 50207 These functions shall fail if:
- 50208 CX [EINVAL] The value of *base* is not supported.
- 50209 [ERANGE] The value to be returned is not representable.
- 50210 These functions may fail if:
- 50211 CX [EINVAL] No conversion could be performed.
- 50212 **EXAMPLES**
- 50213 None.
- 50214 **APPLICATION USAGE**
- 50215 None.
- 50216 **RATIONALE**
- 50217 None.
- 50218 **FUTURE DIRECTIONS**
- 50219 None.
- 50220 **SEE ALSO**
- 50221 *iswalpha()*, *scanf()*, *wcstod()*, the Base Definitions volume of IEEE Std 1003.1-200x, <**wchar.h**>
- 50222 **CHANGE HISTORY**
- 50223 First released in Issue 4. Derived from the MSE working draft.
- 50224 **Issue 5**
- 50225 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.
- 50226 **Issue 6**
- 50227 Extensions beyond the ISO C standard are now marked.
- 50228 The following new requirements on POSIX implementations derive from alignment with the  
50229 Single UNIX Specification:
- 50230 • In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is  
50231 added if no conversion could be performed.

50232 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

50233 • The *wcstol()* prototype is updated.

50234 • The *wcstoll()* function is added.

50235 **NAME**

50236           wcstold — convert a wide-character string to a double-precision number

50237 **SYNOPSIS**

50238           #include <wchar.h>

50239           long double wcstold(const wchar\_t \*restrict *nptr*,  
50240                               wchar\_t \*\*restrict *endptr*);

50241 **DESCRIPTION**

50242           Refer to *wcstod*().

50243 **NAME**

50244           wcstoll — convert a wide-character string to a long integer

50245 **SYNOPSIS**

50246           #include <wchar.h>

50247           long long wcstoll(const wchar\_t \*restrict *nptr*,

50248                    wchar\_t \*\*restrict *endptr*, int *base*);

50249 **DESCRIPTION**

50250           Refer to *wcstol*().

50251 **NAME**50252 `wcstombs` — convert a wide-character string to a character string50253 **SYNOPSIS**50254 `#include <stdlib.h>`50255 `size_t wcstombs(char *restrict s, const wchar_t *restrict pwcs,`  
50256 `size_t n);`50257 **DESCRIPTION**50258 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
50259 conflict between the requirements described here and the ISO C standard is unintentional. This  
50260 volume of IEEE Std 1003.1-200x defers to the ISO C standard.50261 The `wcstombs()` function shall convert the sequence of wide-character codes that are in the array  
50262 pointed to by `pwcs` into a sequence of characters that begins in the initial shift state and store  
50263 these characters into the array pointed to by `s`, stopping if a character would exceed the limit of `n`  
50264 total bytes or if a null byte is stored. Each wide-character code shall be converted as if by a call to  
50265 `wctomb()`, except that the shift state of `wctomb()` shall not be affected.50266 The behavior of this function shall be affected by the `LC_CTYPE` category of the current locale.50267 No more than `n` bytes shall be modified in the array pointed to by `s`. If copying takes place  
50268 **CX** between objects that overlap, the behavior is undefined. If `s` is a null pointer, `wcstombs()` shall  
50269 return the length required to convert the entire array regardless of the value of `n`, but no values  
50270 are stored.50271 The `wcstombs()` function need not be reentrant. A function that is not required to be reentrant is  
50272 not required to be thread-safe.50273 **RETURN VALUE**50274 If a wide-character code is encountered that does not correspond to a valid character (of one or  
50275 more bytes each), `wcstombs()` shall return `(size_t)-1`. Otherwise, `wcstombs()` shall return the  
50276 number of bytes stored in the character array, not including any terminating null byte. The array  
50277 shall not be null-terminated if the value returned is `n`.50278 **ERRORS**50279 The `wcstombs()` function may fail if:50280 **CX** `[EILSEQ]` A wide-character code does not correspond to a valid character.50281 **EXAMPLES**

50282 None.

50283 **APPLICATION USAGE**

50284 None.

50285 **RATIONALE**

50286 None.

50287 **FUTURE DIRECTIONS**

50288 None.

50289 **SEE ALSO**50290 `mblen()`, `mbtowc()`, `mbstowcs()`, `wctomb()`, the Base Definitions volume of IEEE Std 1003.1-200x,  
50291 `<stdlib.h>`

50292 **CHANGE HISTORY**

50293 First released in Issue 4. Derived from the ISO C standard.

50294 **Issue 6**

50295 The following new requirements on POSIX implementations derive from alignment with the  
50296 Single UNIX Specification:

- 50297 • The DESCRIPTION states the effect of when *s* is a null pointer.
- 50298 • The [EILSEQ] error condition is added.

50299 The *wcstombs()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

## 50300 NAME

50301 wcstoul, wcstoull — convert a wide-character string to an unsigned long

## 50302 SYNOPSIS

50303 #include <wchar.h>

50304 unsigned long wcstoul(const wchar\_t \*restrict *nptr*,  
50305 wchar\_t \*\*restrict *endptr*, int *base*);

50306 unsigned long long wcstoull(const wchar\_t \*restrict *nptr*,  
50307 wchar\_t \*\*restrict *endptr*, int *base*);

## 50308 DESCRIPTION

50309 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
50310 conflict between the requirements described here and the ISO C standard is unintentional. This  
50311 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

50312 The *wcstoul()* and *wcstoull()* functions shall convert the initial portion of the wide-character  
50313 string pointed to by *nptr* to **unsigned long** and **unsigned long long** representation, respectively. |  
50314 First, they shall decompose the input wide-character string into three parts: |

- 50315 1. An initial, possibly empty, sequence of white-space wide-character codes (as specified by  
50316 *iswspace()*)
- 50317 2. A subject sequence interpreted as an integer represented in some radix determined by the  
50318 value of *base*
- 50319 3. A final wide-character string of one or more unrecognized wide-character codes, including  
50320 the terminating null wide-character code of the input wide-character string

50321 Then they shall attempt to convert the subject sequence to an unsigned integer, and return the |  
50322 result. |

50323 If *base* is 0, the expected form of the subject sequence is that of a decimal constant, octal constant,  
50324 or hexadecimal constant, any of which may be preceded by a '+' or '-' sign. A decimal  
50325 constant begins with a non-zero digit, and consists of a sequence of decimal digits. An octal  
50326 constant consists of the prefix '0' optionally followed by a sequence of the digits '0' to '7'  
50327 only. A hexadecimal constant consists of the prefix 0x or 0X followed by a sequence of the  
50328 decimal digits and letters 'a' (or 'A') to 'f' (or 'F') with values 10 to 15 respectively.

50329 If the value of *base* is between 2 and 36, the expected form of the subject sequence is a sequence  
50330 of letters and digits representing an integer with the radix specified by *base*, optionally preceded  
50331 by a '+' or '-' sign, but not including an integer suffix. The letters from 'a' (or 'A') to 'z'  
50332 (or 'Z') inclusive are ascribed the values 10 to 35; only letters whose ascribed values are less  
50333 than that of *base* shall be permitted. If the value of *base* is 16, the wide-character codes 0x or 0X |  
50334 may optionally precede the sequence of letters and digits, following the sign if present.

50335 The subject sequence is defined as the longest initial subsequence of the input wide-character  
50336 string, starting with the first wide-character code that is not white space and is of the expected  
50337 form. The subject sequence contains no wide-character codes if the input wide-character string is  
50338 empty or consists entirely of white-space wide-character codes, or if the first wide-character  
50339 code that is not white space is other than a sign or a permissible letter or digit.

50340 If the subject sequence has the expected form and *base* is 0, the sequence of wide-character codes  
50341 starting with the first digit shall be interpreted as an integer constant. If the subject sequence has |  
50342 the expected form and the value of *base* is between 2 and 36, it shall be used as the base for |  
50343 conversion, ascribing to each letter its value as given above. If the subject sequence begins with a |  
50344 minus sign, the value resulting from the conversion shall be negated. A pointer to the final |  
50345 wide-character string shall be stored in the object pointed to by *endptr*, provided that *endptr* is |

- 50346 not a null pointer.
- 50347 CX In other than the C or POSIX locales, other implementation-defined subject sequences may be  
50348 accepted.
- 50349 If the subject sequence is empty or does not have the expected form, no conversion shall be |  
50350 performed; the value of *nptr* shall be stored in the object pointed to by *endptr*, provided that |  
50351 *endptr* is not a null pointer.
- 50352 CX The *wcstoul()* function shall not change the setting of *errno* if successful.
- 50353 Since 0, {ULONG\_MAX}, and {ULLONG\_MAX} are returned on error and 0 is also a valid return |  
50354 on success, an application wishing to check for error situations should set *errno* to 0, then call |  
50355 *wcstoul()* or *wcstoull()*, then check *errno*.
- 50356 **RETURN VALUE**
- 50357 Upon successful completion, the *wcstoul()* and *wcstoull()* functions shall return the converted  
50358 CX value, if any. If no conversion could be performed, 0 shall be returned and *errno* may be set to  
50359 indicate the error. If the correct value is outside the range of representable values,  
50360 {ULONG\_MAX} or {ULLONG\_MAX} respectively shall be returned and *errno* set to [ERANGE].
- 50361 **ERRORS**
- 50362 These functions shall fail if:
- 50363 CX [EINVAL] The value of *base* is not supported.
- 50364 [ERANGE] The value to be returned is not representable.
- 50365 These functions may fail if:
- 50366 CX [EINVAL] No conversion could be performed.
- 50367 **EXAMPLES**
- 50368 None.
- 50369 **APPLICATION USAGE**
- 50370 None.
- 50371 **RATIONALE**
- 50372 None.
- 50373 **FUTURE DIRECTIONS**
- 50374 None.
- 50375 **SEE ALSO**
- 50376 *iswalph()*, *scanf()*, *wcstod()*, *wcstol()*, the Base Definitions volume of IEEE Std 1003.1-200x,  
50377 <wchar.h>
- 50378 **CHANGE HISTORY**
- 50379 First released in Issue 4. Derived from the MSE working draft.
- 50380 **Issue 5**
- 50381 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.
- 50382 **Issue 6**
- 50383 Extensions beyond the ISO C standard are now marked.
- 50384 The following new requirements on POSIX implementations derive from alignment with the  
50385 Single UNIX Specification:
- 50386
- The [EINVAL] error condition is added for when the value of *base* is not supported.

50387            In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is  
50388            added if no conversion could be performed.

50389            The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 50390            • The *wcstoul()* prototype is updated.
- 50391            • The *wcstoull()* function is added.

50392 **NAME**

50393           wcstoull — convert a wide-character string to an unsigned long

50394 **SYNOPSIS**

50395           #include &lt;wchar.h&gt;

50396           unsigned long long wcstoull(const wchar\_t \*restrict *nptr*,50397                    wchar\_t \*\*restrict *endptr*, int *base*);50398 **DESCRIPTION**50399           Refer to *wcstoul()*.

50400 **NAME**

50401           wcstoumax — convert wide-character string to integer type

50402 **SYNOPSIS**

50403           #include <stddef.h>

50404           #include <inttypes.h>

50405           uintmax\_t wcstoumax(const wchar\_t \*restrict *nptr*,

50406                            wchar\_t \*\*restrict *endptr*, int *base*);

50407 **DESCRIPTION**

50408           Refer to *wcstoimax()*.

50409 **NAME**

50410           wcswcs — find a wide substring (**LEGACY**)

50411 **SYNOPSIS**

```
50412 xSI       #include <wchar.h>
```

```
50413           wchar_t *wcswcs(const wchar_t *ws1, const wchar_t *ws2);
```

50414

50415 **DESCRIPTION**

50416           The `wcswcs()` function shall locate the first occurrence in the wide-character string pointed to by  
50417           `ws1` of the sequence of wide-character codes (excluding the terminating null wide-character  
50418           code) in the wide-character string pointed to by `ws2`.

50419 **RETURN VALUE**

50420           Upon successful completion, `wcswcs()` shall return a pointer to the located wide-character string  
50421           or a null pointer if the wide-character string is not found.

50422           If `ws2` points to a wide-character string with zero length, the function shall return `ws1`.

50423 **ERRORS**

50424           No errors are defined.

50425 **EXAMPLES**

50426           None.

50427 **APPLICATION USAGE**

50428           This function was not included in the final ISO/IEC 9899:1990/Amendment 1:1995 (E).  
50429           Application developers are strongly encouraged to use the `wcsstr()` function instead.

50430 **RATIONALE**

50431           None.

50432 **FUTURE DIRECTIONS**

50433           This function may be withdrawn in a future version.

50434 **SEE ALSO**

50435           `wcschr()`, `wcsstr()`, the Base Definitions volume of IEEE Std 1003.1-200x, `<wchar.h>`

50436 **CHANGE HISTORY**

50437           First released in Issue 4. Derived from the MSE working draft.

50438 **Issue 5**

50439           Marked EX.

50440 **Issue 6**

50441           This function is marked LEGACY.

50442 **NAME**50443        `wcswidth` — number of column positions of a wide-character string50444 **SYNOPSIS**50445 XSI        `#include <wchar.h>`50446        `int wcswidth(const wchar_t *pwcs, size_t n);`

50447

50448 **DESCRIPTION**

50449        The `wcswidth()` function shall determine the number of column positions required for *n* wide-character codes (or fewer than *n* wide-character codes if a null wide-character code is encountered before *n* wide-character codes are exhausted) in the string pointed to by *pwcs*.

50452 **RETURN VALUE**

50453        The `wcswidth()` function either shall return 0 (if *pwcs* points to a null wide-character code), or return the number of column positions to be occupied by the wide-character string pointed to by *pwcs*, or return -1 (if any of the first *n* wide-character codes in the wide-character string pointed to by *pwcs* is not a printable wide-character code).

50457 **ERRORS**

50458        No errors are defined.

50459 **EXAMPLES**

50460        None.

50461 **APPLICATION USAGE**

50462        This function was removed from the final ISO/IEC 9899:1990/Amendment 1:1995 (E), and the return value for a non-printable wide character is not specified.

50464 **RATIONALE**

50465        None.

50466 **FUTURE DIRECTIONS**

50467        None.

50468 **SEE ALSO**

50469        `wcwidth()`, the Base Definitions volume of IEEE Std 1003.1-200x, `<wchar.h>`, the Base Definitions volume of IEEE Std 1003.1-200x, Section 3.103, Column Position

50471 **CHANGE HISTORY**

50472        First released in Issue 4. Derived from the MSE working draft.

50473 **Issue 6**

50474        The Open Group Corrigendum U021/11 is applied. The function is marked as an extension.

50475 **NAME**50476 `wcsxfrm` — wide-character string transformation50477 **SYNOPSIS**50478 `#include <wchar.h>`50479 `size_t wcsxfrm(wchar_t *restrict ws1, const wchar_t *restrict ws2,`  
50480 `size_t n);`50481 **DESCRIPTION**50482 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
50483 conflict between the requirements described here and the ISO C standard is unintentional. This  
50484 volume of IEEE Std 1003.1-200x defers to the ISO C standard.50485 The `wcsxfrm()` function shall transform the wide-character string pointed to by `ws2` and place the  
50486 resulting wide-character string into the array pointed to by `ws1`. The transformation shall be  
50487 such that if `wscmp()` is applied to two transformed wide strings, it shall return a value greater  
50488 than, equal to, or less than 0, corresponding to the result of `wscoll()` applied to the same two  
50489 original wide-character strings. No more than `n` wide-character codes shall be placed into the  
50490 resulting array pointed to by `ws1`, including the terminating null wide-character code. If `n` is 0,  
50491 `ws1` is permitted to be a null pointer. If copying takes place between objects that overlap, the  
50492 behavior is undefined.50493 **CX** The `wcsxfrm()` function shall not change the setting of `errno` if successful.50494 Since no return value is reserved to indicate an error, an application wishing to check for error  
50495 situations should set `errno` to 0, then call `wcsxfrm()`, then check `errno`.50496 **RETURN VALUE**50497 The `wcsxfrm()` function shall return the length of the transformed wide-character string (not  
50498 including the terminating null wide-character code). If the value returned is `n` or more, the  
50499 contents of the array pointed to by `ws1` are unspecified.50500 **CX** On error, the `wcsxfrm()` function may set `errno`, but no return value is reserved to indicate an  
50501 error.50502 **ERRORS**50503 The `wcsxfrm()` function may fail if:50504 **CX** [EINVAL] The wide-character string pointed to by `ws2` contains wide-character codes  
50505 outside the domain of the collating sequence.50506 **EXAMPLES**

50507 None.

50508 **APPLICATION USAGE**50509 The transformation function is such that two transformed wide-character strings can be ordered  
50510 by `wscmp()` as appropriate to collating sequence information in the program's locale (category  
50511 `LC_COLLATE`).50512 The fact that when `n` is 0 `ws1` is permitted to be a null pointer is useful to determine the size of  
50513 the `ws1` array prior to making the transformation.50514 **RATIONALE**

50515 None.

50516 **FUTURE DIRECTIONS**

50517 None.

50518 **SEE ALSO**50519 `wscmp()`, `wscoll()`, the Base Definitions volume of IEEE Std 1003.1-200x, <**wchar.h**>50520 **CHANGE HISTORY**

50521 First released in Issue 4. Derived from the MSE working draft.

50522 **Issue 5**

50523 Moved from ENHANCED I18N to BASE and the [ENOSYS] error is removed.

50524 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.50525 **Issue 6**

50526 In previous versions, this function was required to return -1 on error.

50527 Extensions beyond the ISO C standard are now marked.

50528 The following new requirements on POSIX implementations derive from alignment with the  
50529 Single UNIX Specification:

- 50530
- In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is
- 50531 added if no conversion could be performed.

50532 The `wcsxfrm()` prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

50533 **NAME**50534 `wctob` — wide-character to single-byte conversion50535 **SYNOPSIS**50536 `#include <stdio.h>`50537 `#include <wchar.h>`50538 `int wctob(wint_t c);`50539 **DESCRIPTION**

50540 `cx` The functionality described on this reference page is aligned with the ISO C standard. Any  
50541 conflict between the requirements described here and the ISO C standard is unintentional. This  
50542 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

50543 The `wctob()` function shall determine whether `c` corresponds to a member of the extended  
50544 character set whose character representation is a single byte when in the initial shift state.

50545 The behavior of this function shall be affected by the `LC_CTYPE` category of the current locale.

50546 **RETURN VALUE**

50547 The `wctob()` function shall return EOF if `c` does not correspond to a character with length one in  
50548 the initial shift state. Otherwise, it shall return the single-byte representation of that character as  
50549 an **unsigned char** converted to **int**.

50550 **ERRORS**

50551 No errors are defined.

50552 **EXAMPLES**

50553 None.

50554 **APPLICATION USAGE**

50555 None.

50556 **RATIONALE**

50557 None.

50558 **FUTURE DIRECTIONS**

50559 None.

50560 **SEE ALSO**50561 `btowc()`, the Base Definitions volume of IEEE Std 1003.1-200x, `<wchar.h>`50562 **CHANGE HISTORY**

50563 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995  
50564 (E).

50565 **NAME**

50566 wctomb — convert a wide-character code to a character

50567 **SYNOPSIS**

50568 #include &lt;stdlib.h&gt;

50569 int wctomb(char \*s, wchar\_t wchar);

50570 **DESCRIPTION**

50571 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
50572 conflict between the requirements described here and the ISO C standard is unintentional. This  
50573 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

50574 The *wctomb()* function shall determine the number of bytes needed to represent the character  
50575 corresponding to the wide-character code whose value is *wchar* (including any change in the  
50576 shift state). It shall store the character representation (possibly multiple bytes and any special  
50577 bytes to change shift state) in the array object pointed to by *s* (if *s* is not a null pointer). At most  
50578 {MB\_CUR\_MAX} bytes shall be stored. If *wchar* is 0, a null byte shall be stored, preceded by any  
50579 shift sequence needed to restore the initial shift state, and *wctomb()* shall be left in the initial shift  
50580 state.

50581 cx The behavior of this function is affected by the *LC\_CTYPE* category of the current locale. For a  
50582 state-dependent encoding, this function shall be placed into its initial state by a call for which its  
50583 character pointer argument, *s*, is a null pointer. Subsequent calls with *s* as other than a null  
50584 pointer shall cause the internal state of the function to be altered as necessary. A call with *s* as a  
50585 null pointer shall cause this function to return a non-zero value if encodings have state  
50586 dependency, and 0 otherwise. Changing the *LC\_CTYPE* category causes the shift state of this  
50587 function to be unspecified.

50588 The *wctomb()* function need not be reentrant. A function that is not required to be reentrant is  
50589 not required to be thread-safe.

50590 The implementation shall behave as if no function defined in this volume of  
50591 IEEE Std 1003.1-200x calls *wctomb()*.

50592 **RETURN VALUE**

50593 If *s* is a null pointer, *wctomb()* shall return a non-zero or 0 value, if character encodings,  
50594 respectively, do or do not have state-dependent encodings. If *s* is not a null pointer, *wctomb()*  
50595 shall return -1 if the value of *wchar* does not correspond to a valid character, or return the  
50596 number of bytes that constitute the character corresponding to the value of *wchar*.

50597 In no case shall the value returned be greater than the value of the {MB\_CUR\_MAX} macro.

50598 **ERRORS**

50599 No errors are defined.

50600 **EXAMPLES**

50601 None.

50602 **APPLICATION USAGE**

50603 None.

50604 **RATIONALE**

50605 None.

50606 **FUTURE DIRECTIONS**

50607 None.

50608 **SEE ALSO**

50609            *mblen()*, *mbtowc()*, *mbstowcs()*, *wcstombs()*, the Base Definitions volume of IEEE Std 1003.1-200x,  
50610            <**stdlib.h**>

50611 **CHANGE HISTORY**

50612            First released in Issue 4. Derived from the ANSI C standard.

50613 **Issue 6**

50614            Extensions beyond the ISO C standard are now marked.

50615            In the DESCRIPTION, a note about reentrancy and thread-safety is added.

50616 **NAME**

50617 wctrans — define character mapping

50618 **SYNOPSIS**

50619 #include &lt;wctype.h&gt;

50620 wctrans\_t wctrans(const char \*charclass);

50621 **DESCRIPTION**

50622 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
 50623 conflict between the requirements described here and the ISO C standard is unintentional. This  
 50624 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

50625 The *wctrans()* function is defined for valid character mapping names identified in the current  
 50626 locale. The *charclass* is a string identifying a generic character mapping name for which codeset-  
 50627 specific information is required. The following character mapping names are defined in all  
 50628 locales: **tolower** and **toupper**.

50629 The function shall return a value of type **wctrans\_t**, which can be used as the second argument  
 50630 to subsequent calls of *towctrans()*. The *wctrans()* function shall determine values of **wctrans\_t**  
 50631 according to the rules of the coded character set defined by character mapping information in  
 50632 the program's locale (category *LC\_CTYPE*). The values returned by *wctrans()* shall be valid until  
 50633 a call to *setlocale()* that modifies the category *LC\_CTYPE*.

50634 **RETURN VALUE**

50635 cx The *wctrans()* function shall return 0 and may set *errno* to indicate the error if the given  
 50636 character mapping name is not valid for the current locale (category *LC\_CTYPE*); otherwise, it  
 50637 shall return a non-zero object of type **wctrans\_t** that can be used in calls to *towctrans()*.

50638 **ERRORS**50639 The *wctrans()* function may fail if:

50640 cx [EINVAL] The character mapping name pointed to by *charclass* is not valid in the current  
 50641 locale.

50642 **EXAMPLES**

50643 None.

50644 **APPLICATION USAGE**

50645 None.

50646 **RATIONALE**

50647 None.

50648 **FUTURE DIRECTIONS**

50649 None.

50650 **SEE ALSO**50651 *towctrans()*, the Base Definitions volume of IEEE Std 1003.1-200x, <wctype.h>50652 **CHANGE HISTORY**

50653 First released in Issue 5. Derived from ISO/IEC 9899:1990/Amendment 1:1995 (E).

50654 **NAME**

50655 wctype — define character class

50656 **SYNOPSIS**

50657 #include &lt;wctype.h&gt;

50658 wctype\_t wctype(const char \*property);

50659 **DESCRIPTION**

50660 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
 50661 conflict between the requirements described here and the ISO C standard is unintentional. This  
 50662 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

50663 The *wctype()* function is defined for valid character class names as defined in the current locale.  
 50664 The *property* argument is a string identifying a generic character class for which codeset-specific  
 50665 type information is required. The following character class names shall be defined in all locales:

50666	<b>alnum</b>	<b>digit</b>	<b>punct</b>
50667	<b>alpha</b>	<b>graph</b>	<b>space</b>
50668	<b>blank</b>	<b>lower</b>	<b>upper</b>
50669	<b>cntrl</b>	<b>print</b>	<b>xdigit</b>

50670 Additional character class names defined in the locale definition file (category *LC\_CTYPE*) can  
 50671 also be specified.

50672 The function shall return a value of type **wctype\_t**, which can be used as the second argument to  
 50673 subsequent calls of *iswctype()*. The *wctype()* function shall determine values of **wctype\_t**  
 50674 according to the rules of the coded character set defined by character type information in the  
 50675 program's locale (category *LC\_CTYPE*). The values returned by *wctype()* shall be valid until a  
 50676 call to *setlocale()* that modifies the category *LC\_CTYPE*.

50677 **RETURN VALUE**

50678 The *wctype()* function shall return 0 if the given character class name is not valid for the current  
 50679 locale (category *LC\_CTYPE*); otherwise, it shall return an object of type **wctype\_t** that can be  
 50680 used in calls to *iswctype()*.

50681 **ERRORS**

50682 No errors are defined.

50683 **EXAMPLES**

50684 None.

50685 **APPLICATION USAGE**

50686 None.

50687 **RATIONALE**

50688 None.

50689 **FUTURE DIRECTIONS**

50690 None.

50691 **SEE ALSO**50692 *iswctype()*, the Base Definitions volume of IEEE Std 1003.1-200x, <wctype.h>, <wchar.h>50693 **CHANGE HISTORY**

50694 First released in Issue 4.

50695 **Issue 5**

50696 The following change has been made in this issue for alignment with  
50697 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 50698 • The SYNOPSIS has been changed to indicate that this function and associated data types are  
50699 now made visible by inclusion of the `<wctype.h>` header rather than `<wchar.h>`.

50700 **NAME**

50701 `wcwidth` — number of column positions of a wide-character code

50702 **SYNOPSIS**

50703 XSI `#include <wchar.h>`

50704 `int wcwidth(wchar_t wc);`

50705

50706 **DESCRIPTION**

50707 The `wcwidth()` function shall determine the number of column positions required for the wide  
50708 character `wc`. The application shall ensure that the value of `wc` is a character representable as a  
50709 **wchar\_t**, and is a wide-character code corresponding to a valid character in the current locale. |

50710 **RETURN VALUE**

50711 The `wcwidth()` function shall either return 0 (if `wc` is a null wide-character code), or return the  
50712 number of column positions to be occupied by the wide-character code `wc`, or return -1 (if `wc`  
50713 does not correspond to a printable wide-character code). |

50714 **ERRORS**

50715 No errors are defined.

50716 **EXAMPLES**

50717 None.

50718 **APPLICATION USAGE**

50719 This function was removed from the final ISO/IEC 9899:1990/Amendment 1:1995 (E), and the  
50720 return value for a non-printable wide character is not specified.

50721 **RATIONALE**

50722 None.

50723 **FUTURE DIRECTIONS**

50724 None.

50725 **SEE ALSO**

50726 `wcswidth()`, the Base Definitions volume of IEEE Std 1003.1-200x, `<wchar.h>`

50727 **CHANGE HISTORY**

50728 First released as a World-wide Portability Interface in Issue 4. Derived from MSE working draft.

50729 **Issue 6**

50730 The Open Group Corrigendum U021/12 is applied. This function is marked as an extension.

50731 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

50732 **NAME**

50733 wmemchr — find a wide character in memory

50734 **SYNOPSIS**

50735 #include &lt;wchar.h&gt;

50736 wchar\_t \*wmemchr(const wchar\_t \*ws, wchar\_t wc, size\_t n);

50737 **DESCRIPTION**

50738 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
50739 conflict between the requirements described here and the ISO C standard is unintentional. This  
50740 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

50741 The *wmemchr()* function shall locate the first occurrence of *wc* in the initial *n* wide characters of  
50742 the object pointed to by *ws*. This function shall not be affected by locale and all **wchar\_t** values  
50743 shall be treated identically. The null wide character and **wchar\_t** values not corresponding to  
50744 valid characters shall not be treated specially.

50745 If *n* is zero, the application shall ensure that *ws* is a valid pointer and the function behaves as if  
50746 no valid occurrence of *wc* is found.

50747 **RETURN VALUE**

50748 The *wmemchr()* function shall return a pointer to the located wide character, or a null pointer if  
50749 the wide character does not occur in the object.

50750 **ERRORS**

50751 No errors are defined.

50752 **EXAMPLES**

50753 None.

50754 **APPLICATION USAGE**

50755 None.

50756 **RATIONALE**

50757 None.

50758 **FUTURE DIRECTIONS**

50759 None.

50760 **SEE ALSO**

50761 *wmemcmp()*, *wmemcpy()*, *wmemmove()*, *wmemset()*, the Base Definitions volume of  
50762 IEEE Std 1003.1-200x, <wchar.h>

50763 **CHANGE HISTORY**

50764 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995  
50765 (E).

50766 **Issue 6**

50767 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

50768 **NAME**

50769 wmemcmp — compare wide characters in memory

50770 **SYNOPSIS**

50771 #include &lt;wchar.h&gt;

50772 int wmemcmp(const wchar\_t \*ws1, const wchar\_t \*ws2, size\_t n);

50773 **DESCRIPTION**

50774 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
50775 conflict between the requirements described here and the ISO C standard is unintentional. This  
50776 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

50777 The *wmemcmp()* function shall compare the first *n* wide characters of the object pointed to by  
50778 *ws1* to the first *n* wide characters of the object pointed to by *ws2*. This function shall not be  
50779 affected by locale and all **wchar\_t** values shall be treated identically. The null wide character and  
50780 **wchar\_t** values not corresponding to valid characters shall not be treated specially.

50781 If *n* is zero, the application shall ensure that *ws1* and *ws2* are valid pointers, and the function  
50782 shall behave as if the two objects compare equal.

50783 **RETURN VALUE**

50784 The *wmemcmp()* function shall return an integer greater than, equal to, or less than zero,  
50785 respectively, as the object pointed to by *ws1* is greater than, equal to, or less than the object  
50786 pointed to by *ws2*.

50787 **ERRORS**

50788 No errors are defined.

50789 **EXAMPLES**

50790 None.

50791 **APPLICATION USAGE**

50792 None.

50793 **RATIONALE**

50794 None.

50795 **FUTURE DIRECTIONS**

50796 None.

50797 **SEE ALSO**

50798 *wmemchr()*, *wmemcpy()*, *wmemmove()*, *wmemset()*, the Base Definitions volume of  
50799 IEEE Std 1003.1-200x, <**wchar.h**>

50800 **CHANGE HISTORY**

50801 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995  
50802 (E).

50803 **Issue 6**

50804 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

50805 **NAME**

50806 wmemcpy — copy wide characters in memory

50807 **SYNOPSIS**

50808 #include &lt;wchar.h&gt;

50809 wchar\_t \*wmemcpy(wchar\_t \*restrict ws1, const wchar\_t \*restrict ws2,  
50810 size\_t n);50811 **DESCRIPTION**50812 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
50813 conflict between the requirements described here and the ISO C standard is unintentional. This  
50814 volume of IEEE Std 1003.1-200x defers to the ISO C standard.50815 The *wmemcpy()* function shall copy *n* wide characters from the object pointed to by *ws2* to the  
50816 object pointed to by *ws1*. This function shall not be affected by locale and all **wchar\_t** values  
50817 shall be treated identically. The null wide character and **wchar\_t** values not corresponding to  
50818 valid characters shall not be treated specially. |50819 If *n* is zero, the application shall ensure that *ws1* and *ws2* are valid pointers, and the function  
50820 shall copy zero wide characters. |50821 **RETURN VALUE**50822 The *wmemcpy()* function shall return the value of *ws1*.50823 **ERRORS**

50824 No errors are defined.

50825 **EXAMPLES**

50826 None.

50827 **APPLICATION USAGE**

50828 None.

50829 **RATIONALE**

50830 None.

50831 **FUTURE DIRECTIONS**

50832 None.

50833 **SEE ALSO**50834 *wmemchr()*, *wmemcmp()*, *wmemmove()*, *wmemset()*, the Base Definitions volume of  
50835 IEEE Std 1003.1-200x, <**wchar.h**>50836 **CHANGE HISTORY**50837 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995  
50838 (E).50839 **Issue 6**

50840 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

50841 The *wmemcpy()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

50842 **NAME**

50843 wmemmove — copy wide characters in memory with overlapping areas

50844 **SYNOPSIS**

50845 #include &lt;wchar.h&gt;

50846 wchar\_t \*wmemmove(wchar\_t \*ws1, const wchar\_t \*ws2, size\_t n);

50847 **DESCRIPTION**

50848 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
50849 conflict between the requirements described here and the ISO C standard is unintentional. This  
50850 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

50851 The *wmemmove()* function shall copy *n* wide characters from the object pointed to by *ws2* to the  
50852 object pointed to by *ws1*. Copying shall take place as if the *n* wide characters from the object  
50853 pointed to by *ws2* are first copied into a temporary array of *n* wide characters that does not  
50854 overlap the objects pointed to by *ws1* or *ws2*, and then the *n* wide characters from the temporary  
50855 array are copied into the object pointed to by *ws1*.

50856 This function shall not be affected by locale and all **wchar\_t** values shall be treated identically.  
50857 The null wide character and **wchar\_t** values not corresponding to valid characters shall not be  
50858 treated specially.

50859 If *n* is zero, the application shall ensure that *ws1* and *ws2* are valid pointers, and the function  
50860 shall copy zero wide characters.

50861 **RETURN VALUE**50862 The *wmemmove()* function shall return the value of *ws1*.50863 **ERRORS**

50864 No errors are defined

50865 **EXAMPLES**

50866 None.

50867 **APPLICATION USAGE**

50868 None.

50869 **RATIONALE**

50870 None.

50871 **FUTURE DIRECTIONS**

50872 None.

50873 **SEE ALSO**

50874 *wmemchr()*, *wmemcmp()*, *wmemcpy()*, *wmemset()*, the Base Definitions volume of  
50875 IEEE Std 1003.1-200x, <**wchar.h**>

50876 **CHANGE HISTORY**

50877 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995  
50878 (E).

50879 **Issue 6**

50880 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

50881 **NAME**

50882 wmemset — set wide characters in memory

50883 **SYNOPSIS**

50884 #include &lt;wchar.h&gt;

50885 wchar\_t \*wmemset(wchar\_t \*ws, wchar\_t wc, size\_t n);

50886 **DESCRIPTION**

50887 cx The functionality described on this reference page is aligned with the ISO C standard. Any  
 50888 conflict between the requirements described here and the ISO C standard is unintentional. This  
 50889 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

50890 The *wmemset()* function shall copy the value of *wc* into each of the first *n* wide characters of the  
 50891 object pointed to by *ws*. This function shall not be affected by locale and all **wchar\_t** values shall  
 50892 be treated identically. The null wide character and **wchar\_t** values not corresponding to valid  
 50893 characters shall not be treated specially.

50894 If *n* is zero, the application shall ensure that *ws* is a valid pointer, and the function shall copy  
 50895 zero wide characters.

50896 **RETURN VALUE**50897 The *wmemset()* functions shall return the value of *ws*.50898 **ERRORS**

50899 No errors are defined.

50900 **EXAMPLES**

50901 None.

50902 **APPLICATION USAGE**

50903 None.

50904 **RATIONALE**

50905 None.

50906 **FUTURE DIRECTIONS**

50907 None.

50908 **SEE ALSO**

50909 *wmemchr()*, *wmemcmp()*, *wmemcpy()*, *wmemmove()*, the Base Definitions volume of  
 50910 IEEE Std 1003.1-200x, <**wchar.h**>

50911 **CHANGE HISTORY**

50912 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995  
 50913 (E).

50914 **Issue 6**

50915 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

## 50916 NAME

50917 wordexp, wordfree — perform word expansions

## 50918 SYNOPSIS

50919 #include &lt;wordexp.h&gt;

50920 int wordexp(const char \*restrict words, wordexp\_t \*restrict pwordexp,  
50921 int flags);

50922 void wordfree(wordexp\_t \*pwordexp);

## 50923 DESCRIPTION

50924 The *wordexp()* function shall perform word expansions as described in the Shell and Utilities |  
 50925 volume of IEEE Std 1003.1-200x, Section 2.6, Word Expansions, subject to quoting as in the Shell |  
 50926 and Utilities volume of IEEE Std 1003.1-200x, Section 2.2, Quoting, and place the list of expanded |  
 50927 words into the structure pointed to by *pwordexp*. |

50928 The *words* argument is a pointer to a string containing one or more words to be expanded. The |  
 50929 expansions shall be the same as would be performed by the command line interpreter if *words* |  
 50930 were the part of a command line representing the arguments to a utility. Therefore, the |  
 50931 application shall ensure that *words* does not contain an unquoted <newline> or any of the |  
 50932 unquoted shell special characters `'|', '&', ';', '<', '>'` except in the context of command |  
 50933 substitution as specified in the Shell and Utilities volume of IEEE Std 1003.1-200x, Section 2.6.3, |  
 50934 Command Substitution. It also shall not contain unquoted parentheses or braces, except in the |  
 50935 context of command or variable substitution. The application shall ensure that every member of |  
 50936 *words* which it expects to have expanded by *wordexp()* does not contain an unquoted initial |  
 50937 comment character. The application shall also ensure that any words which it intends to be |  
 50938 ignored (because they begin or continue a comment) are deleted from *words*. If the argument |  
 50939 *words* contains an unquoted comment character (number sign) that is the beginning of a token, |  
 50940 *wordexp()* shall either treat the comment character as a regular character, or interpret it as a |  
 50941 comment indicator and ignore the remainder of *words*.

50942 The structure type **wordexp\_t** is defined in the **<wordexp.h>** header and includes at least the |  
 50943 following members: |

50944

50945

Member Type	Member Name	Description
size_t	<i>we_wordc</i>	Count of words matched by <i>words</i> .
char **	<i>we_wordv</i>	Pointer to list of expanded words.
size_t	<i>we_offs</i>	Slots to reserve at the beginning of <i>pwordexp-&gt;we_wordv</i> .

50946

50947

50948

50949 The *wordexp()* function shall store the number of generated words into *pwordexp->we\_wordc* and |  
 50950 a pointer to a list of pointers to words in *pwordexp->we\_wordv*. Each individual field created |  
 50951 during field splitting (see the Shell and Utilities volume of IEEE Std 1003.1-200x, Section 2.6.5, |  
 50952 Field Splitting) or pathname expansion (see the Shell and Utilities volume of |  
 50953 IEEE Std 1003.1-200x, Section 2.6.6, Pathname Expansion) shall be a separate word in the |  
 50954 *pwordexp->we\_wordv* list. The words shall be in order as described in the Shell and Utilities |  
 50955 volume of IEEE Std 1003.1-200x, Section 2.6, Word Expansions. The first pointer after the last |  
 50956 word pointer shall be a null pointer. The expansion of special parameters described in the Shell |  
 50957 and Utilities volume of IEEE Std 1003.1-200x, Section 2.5.2, Special Parameters is unspecified. |

50958 It is the caller's responsibility to allocate the storage pointed to by *pwordexp*. The *wordexp()* |  
 50959 function shall allocate other space as needed, including memory pointed to by *pwordexp-* |  
 50960 *>we\_wordv*. The *wordfree()* function frees any memory associated with *pwordexp* from a previous |  
 50961 call to *wordexp()*.

50962 The *flags* argument is used to control the behavior of *wordexp()*. The value of *flags* is the  
50963 bitwise-inclusive OR of zero or more of the following constants, which are defined in  
50964 **<wordexp.h>**:

50965 **WRDE\_APPEND** Append words generated to the ones from a previous call to *wordexp()*.

50966 **WRDE\_DOOFFS** Make use of *pwordexp->we\_offs*. If this flag is set, *pwordexp->we\_offs* is used  
50967 to specify how many null pointers to add to the beginning of *pwordexp->we\_wordv*. In other words, *pwordexp->we\_wordv*  
50968 shall point to *pwordexp->we\_offs* null pointers, followed by *pwordexp->we\_wordc* word pointers,  
50970 followed by a null pointer.

50971 **WRDE\_NOCMD** If the implementation supports the utilities defined in the Shell and  
50972 Utilities volume of IEEE Std 1003.1-200x, fail if command substitution, as  
50973 specified in the Shell and Utilities volume of IEEE Std 1003.1-200x,  
50974 Section 2.6.3, Command Substitution, is requested.

50975 **WRDE\_REUSE** The *pwordexp* argument was passed to a previous successful call to  
50976 *wordexp()*, and has not been passed to *wordfree()*. The result shall be the  
50977 same as if the application had called *wordfree()* and then called *wordexp()*  
50978 without **WRDE\_REUSE**.

50979 **WRDE\_SHOWERR** Do not redirect *stderr* to **/dev/null**.

50980 **WRDE\_UNDEF** Report error on an attempt to expand an undefined shell variable.

50981 The **WRDE\_APPEND** flag can be used to append a new set of words to those generated by a  
50982 previous call to *wordexp()*. The following rules apply to applications when two or more calls to  
50983 *wordexp()* are made with the same value of *pwordexp* and without intervening calls to *wordfree()*:

- 50984 1. The first such call shall not set **WRDE\_APPEND**. All subsequent calls shall set it.
- 50985 2. All of the calls shall set **WRDE\_DOOFFS**, or all shall not set it.
- 50986 3. After the second and each subsequent call, *pwordexp->we\_wordv* shall point to a list  
50987 containing the following:
  - 50988 a. Zero or more null pointers, as specified by **WRDE\_DOOFFS** and *pwordexp->we\_offs*
  - 50989 b. Pointers to the words that were in the *pwordexp->we\_wordv* list before the call, in the  
50990 same order as before
  - 50991 c. Pointers to the new words generated by the latest call, in the specified order
- 50992 4. The count returned in *pwordexp->we\_wordc* shall be the total number of words from all of  
50993 the calls.
- 50994 5. The application can change any of the fields after a call to *wordexp()*, but if it does it shall  
50995 reset them to the original value before a subsequent call, using the same *pwordexp* value, to  
50996 *wordfree()* or *wordexp()* with the **WRDE\_APPEND** or **WRDE\_REUSE** flag.

50997 If the implementation supports the utilities defined in the Shell and Utilities volume of  
50998 IEEE Std 1003.1-200x, and *words* contains an unquoted character—<newline>, '|', '&', ';',  
50999 '<', '>', '(', ')', '{', '}'—in an inappropriate context, *wordexp()* shall fail, and the number  
51000 of expanded words shall be 0.

51001 Unless **WRDE\_SHOWERR** is set in *flags*, *wordexp()* shall redirect *stderr* to **/dev/null** for any  
51002 utilities executed as a result of command substitution while expanding *words*. If  
51003 **WRDE\_SHOWERR** is set, *wordexp()* may write messages to *stderr* if syntax errors are detected  
51004 while expanding *words*.

51005 The application shall ensure that if WRDE\_DOOFFS is set, then *pwordexp->we\_offs* has the same  
51006 value for each *wordexp()* call and *wordfree()* call using a given *pwordexp*.

51007 The following constants are defined as error return values:

51008 WRDE\_BADCHAR One of the unquoted characters—<newline>, '|', '&', ';', '<', '>',  
51009 '(', ')', '{', '}'—appears in *words* in an inappropriate context.

51010 WRDE\_BADVAL Reference to undefined shell variable when WRDE\_UNDEF is set in *flags*.

51011 WRDE\_CMDSUB Command substitution requested when WRDE\_NOCMD was set in *flags*.

51012 WRDE\_NOSPACE Attempt to allocate memory failed.

51013 WRDE\_SYNTAX Shell syntax error, such as unbalanced parentheses or unterminated  
51014 string.

#### 51015 RETURN VALUE

51016 Upon successful completion, *wordexp()* shall return 0. Otherwise, a non-zero value, as described  
51017 in <**wordexp.h**>, shall be returned to indicate an error. If *wordexp()* returns the value  
51018 WRDE\_NOSPACE, then *pwordexp->we\_wordc* and *pwordexp->we\_wordv* shall be updated to  
51019 reflect any words that were successfully expanded. In other cases, they shall not be modified.

51020 The *wordfree()* function shall not return a value.

#### 51021 ERRORS

51022 No errors are defined.

#### 51023 EXAMPLES

51024 None.

#### 51025 APPLICATION USAGE

51026 The *wordexp()* function is intended to be used by an application that wants to do all of the shell's  
51027 expansions on a word or words obtained from a user. For example, if the application prompts  
51028 for a filename (or list of filenames) and then uses *wordexp()* to process the input, the user could  
51029 respond with anything that would be valid as input to the shell.

51030 The WRDE\_NOCMD flag is provided for applications that, for security or other reasons, want to  
51031 prevent a user from executing shell commands. Disallowing unquoted shell special characters  
51032 also prevents unwanted side effects, such as executing a command or writing a file.

#### 51033 RATIONALE

51034 This function was included as an alternative to *glob()*. There had been continuing controversy  
51035 over exactly what features should be included in *glob()*. It is hoped that by providing *wordexp()*  
51036 (which provides all of the shell word expansions, but which may be slow to execute) and *glob()* |  
51037 (which is faster, but which only performs pathname expansion, without tilde or parameter |  
51038 expansion) this will satisfy the majority of applications.

51039 While *wordexp()* could be implemented entirely as a library routine, it is expected that most  
51040 implementations run a shell in a subprocess to do the expansion.

51041 Two different approaches have been proposed for how the required information might be  
51042 presented to the shell and the results returned. They are presented here as examples.

51043 One proposal is to extend the *echo* utility by adding a **-q** option. This option would cause *echo*  
51044 to add a backslash before each backslash and <blank> that occurs within an argument. The  
51045 *wordexp()* function could then invoke the shell as follows:

```
51046 (void) strcpy(buffer, "echo -q");
51047 (void) strcat(buffer, words);
51048 if ((flags & WRDE_SHOWERR) == 0)
```

```
51049         (void) strcat(buffer, "2>/dev/null");
51050         f = popen(buffer, "r");
```

51051 The *wordexp()* function would read the resulting output, remove unquoted backslashes, and  
 51052 break into words at unquoted <blank>s. If the WRDE\_NOCMD flag was set, *wordexp()* would  
 51053 have to scan *words* before starting the subshell to make sure that there would be no command  
 51054 substitution. In any case, it would have to scan *words* for unquoted special characters.

51055 Another proposal is to add the following options to *sh*:

51056 **-w wordlist**

51057 This option provides a wordlist expansion service to applications. The words in *wordlist*  
 51058 shall be expanded and the following written to standard output:

- 51059 1. The count of the number of words after expansion, in decimal, followed by a null byte
- 51060 2. The number of bytes needed to represent the expanded words (not including null  
 51061 separators), in decimal, followed by a null byte
- 51062 3. The expanded words, each terminated by a null byte

51063 If an error is encountered during word expansion, *sh* exits with a non-zero status after  
 51064 writing the former to report any words successfully expanded

51065 **-P** Run in “protected” mode. If specified with the **-w** option, no command substitution shall  
 51066 be performed.

51067 With these options, *wordexp()* could be implemented fairly simply by creating a subprocess  
 51068 using *fork()* and executing *sh* using the line:

```
51069 execl(<shell path>, "sh", "-P", "-w", words, (char *)0);
```

51070 after directing standard error to **/dev/null**.

51071 It seemed objectionable for a library routine to write messages to standard error, unless  
 51072 explicitly requested, so *wordexp()* is required to redirect standard error to **/dev/null** to ensure  
 51073 that no messages are generated, even for commands executed for command substitution. The  
 51074 WRDE\_SHOWERR flag can be specified to request that error messages be written.

51075 The WRDE\_REUSE flag allows the implementation to avoid the expense of freeing and  
 51076 reallocating memory, if that is possible. A minimal implementation can call *wordfree()* when  
 51077 WRDE\_REUSE is set.

#### 51078 FUTURE DIRECTIONS

51079 None.

#### 51080 SEE ALSO

51081 *fnmatch()*, *glob()*, the Base Definitions volume of IEEE Std 1003.1-200x, **<wordexp.h>**, the Shell  
 51082 and Utilities volume of IEEE Std 1003.1-200x

#### 51083 CHANGE HISTORY

51084 First released in Issue 4. Derived from the ISO POSIX-2 standard.

#### 51085 Issue 5

51086 Moved from POSIX2 C-language Binding to BASE.

#### 51087 Issue 6

51088 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

51089 The **restrict** keyword is added to the *wordexp()* prototype for alignment with the  
 51090 ISO/IEC 9899:1999 standard.

51091 **NAME**

51092        `wprintf` — print formatted wide-character output

51093 **SYNOPSIS**

51094        `#include <stdio.h>`

51095        `#include <wchar.h>`

51096        `int wprintf(const wchar_t *restrict format, ...);` |

51097 **DESCRIPTION** |

51098        Refer to *fwprintf()*.

## 51099 NAME

51100 pwrite, write — write on a file

## 51101 SYNOPSIS

51102 #include &lt;unistd.h&gt;

51103 XSI ssize\_t pwrite(int *fildev*, const void \**buf*, size\_t *nbyte*,  
51104 off\_t *offset*);51105 ssize\_t write(int *fildev*, const void \**buf*, size\_t *nbyte*);

## 51106 DESCRIPTION

51107 The *write()* function shall attempt to write *nbyte* bytes from the buffer pointed to by *buf* to the  
51108 file associated with the open file descriptor, *fildev*.51109 Before any action described below is taken, and if *nbyte* is zero and the file is a regular file, the  
51110 *write()* function may detect and return errors as described below. In the absence of errors, or if  
51111 error detection is not performed, the *write()* function shall return zero and have no other results.  
51112 If *nbyte* is zero and the file is not a regular file, the results are unspecified.51113 On a regular file or other file capable of seeking, the actual writing of data shall proceed from the  
51114 position in the file indicated by the file offset associated with *fildev*. Before successful return  
51115 from *write()*, the file offset shall be incremented by the number of bytes actually written. On a  
51116 regular file, if this incremented file offset is greater than the length of the file, the length of the  
51117 file shall be set to this file offset.51118 On a file not capable of seeking, writing shall always take place starting at the current position.  
51119 The value of a file offset associated with such a device is undefined.51120 If the O\_APPEND flag of the file status flags is set, the file offset shall be set to the end of the file  
51121 prior to each write and no intervening file modification operation shall occur between changing  
51122 the file offset and the write operation.51123 XSI If a *write()* requests that more bytes be written than there is room for (for example, the process'  
51124 file size limit or the physical end of a medium), only as many bytes as there is room for shall be  
51125 written. For example, suppose there is space for 20 bytes more in a file before reaching a limit. A  
51126 write of 512 bytes will return 20. The next write of a non-zero number of bytes would give a  
51127 failure return (except as noted below).51128 XSI If the request would cause the file size to exceed the soft file size limit for the process and there  
51129 is no room for any bytes to be written, the request shall fail and the implementation shall  
51130 generate the SIGXFSZ signal for the thread.51131 If *write()* is interrupted by a signal before it writes any data, it shall return  $-1$  with *errno* set to  
51132 [EINTR].51133 If *write()* is interrupted by a signal after it successfully writes some data, it shall return the  
51134 number of bytes written.51135 If the value of *nbyte* is greater than {SSIZE\_MAX}, the result is implementation-defined.51136 After a *write()* to a regular file has successfully returned:

- 51137
- Any successful *read()* from each byte position in the file that was modified by that write shall  
51138 return the data specified by the *write()* for that position until such byte positions are again  
51139 modified.
  - Any subsequent successful *write()* to the same byte position in the file shall overwrite that  
51140 file data.  
51141

51142	Write requests to a pipe or FIFO shall be handled in the same way as a regular file with the	
51143	following exceptions:	
51144	• There is no file offset associated with a pipe, hence each write request shall append to the	
51145	end of the pipe.	
51146	• Write requests of {PIPE_BUF} bytes or less shall not be interleaved with data from other	
51147	processes doing writes on the same pipe. Writes of greater than {PIPE_BUF} bytes may have	
51148	data interleaved, on arbitrary boundaries, with writes by other processes, whether or not the	
51149	O_NONBLOCK flag of the file status flags is set.	
51150	• If the O_NONBLOCK flag is clear, a write request may cause the thread to block, but on	
51151	normal completion it shall return <i>nbyte</i> .	
51152	• If the O_NONBLOCK flag is set, <i>write()</i> requests shall be handled differently, in the	
51153	following ways:	
51154	— The <i>write()</i> function shall not block the thread.	
51155	— A write request for {PIPE_BUF} or fewer bytes shall have the following effect: if there is	
51156	sufficient space available in the pipe, <i>write()</i> shall transfer all the data and return the	
51157	number of bytes requested. Otherwise, <i>write()</i> shall transfer no data and return $-1$ with	
51158	<i>errno</i> set to [EAGAIN].	
51159	— A write request for more than {PIPE_BUF} bytes shall cause one of the following:	
51160	— When at least one byte can be written, transfer what it can and return the number of	
51161	bytes written. When all data previously written to the pipe is read, it shall transfer at	
51162	least {PIPE_BUF} bytes.	
51163	— When no data can be written, transfer no data, and return $-1$ with <i>errno</i> set to	
51164	[EAGAIN].	
51165	When attempting to write to a file descriptor (other than a pipe or FIFO) that supports non-	
51166	blocking writes and cannot accept the data immediately:	
51167	• If the O_NONBLOCK flag is clear, <i>write()</i> shall block the calling thread until the data can be	
51168	accepted.	
51169	• If the O_NONBLOCK flag is set, <i>write()</i> shall not block the thread. If some data can be	
51170	written without blocking the thread, <i>write()</i> shall write what it can and return the number of	
51171	bytes written. Otherwise, it shall return $-1$ and set <i>errno</i> to [EAGAIN].	
51172	Upon successful completion, where <i>nbyte</i> is greater than 0, <i>write()</i> shall mark for update the	
51173	<i>st_ctime</i> and <i>st_mtime</i> fields of the file, and if the file is a regular file, the S_ISUID and S_ISGID	
51174	bits of the file mode may be cleared.	
51175	For regular files, no data transfer shall occur past the offset maximum established in the open	
51176	file description associated with <i>fildes</i> .	
51177	If <i>fildes</i> refers to a socket, <i>write()</i> shall be equivalent to <i>send()</i> with no flags set.	
51178 SIO	If the O_DSYNC bit has been set, write I/O operations on the file descriptor shall complete as	
51179	defined by synchronized I/O data integrity completion.	
51180	If the O_SYNC bit has been set, write I/O operations on the file descriptor shall complete as	
51181	defined by synchronized I/O file integrity completion.	
51182 SHM	If <i>fildes</i> refers to a shared memory object, the result of the <i>write()</i> function is unspecified.	
51183 TYM	If <i>fildes</i> refers to a typed memory object, the result of the <i>write()</i> function is unspecified.	

51184 XSR If *fdes* refers to a STREAM, the operation of *write()* shall be determined by the values of the  
 51185 minimum and maximum *nbyte* range (packet size) accepted by the STREAM. These values are  
 51186 determined by the topmost STREAM module. If *nbyte* falls within the packet size range, *nbyte*  
 51187 bytes shall be written. If *nbyte* does not fall within the range and the minimum packet size value  
 51188 is 0, *write()* shall break the buffer into maximum packet size segments prior to sending the data  
 51189 downstream (the last segment may contain less than the maximum packet size). If *nbyte* does not  
 51190 fall within the range and the minimum value is non-zero, *write()* shall fail with *errno* set to  
 51191 [ERANGE]. Writing a zero-length buffer (*nbyte* is 0) to a STREAMS device sends 0 bytes with 0  
 51192 returned. However, writing a zero-length buffer to a STREAMS-based pipe or FIFO sends no  
 51193 message and 0 is returned. The process may issue *I\_SWROPT ioctl()* to enable zero-length  
 51194 messages to be sent across the pipe or FIFO.

51195 When writing to a STREAM, data messages are created with a priority band of 0. When writing  
 51196 to a STREAM that is not a pipe or FIFO:

- If O\_NONBLOCK is clear, and the STREAM cannot accept data (the STREAM write queue is full due to internal flow control conditions), *write()* shall block until data can be accepted.

- If O\_NONBLOCK is set and the STREAM cannot accept data, *write()* shall return -1 and set *errno* to [EAGAIN].

- If O\_NONBLOCK is set and part of the buffer has been written while a condition in which the STREAM cannot accept additional data occurs, *write()* shall terminate and return the number of bytes written.

51204 In addition, *write()* shall fail if the STREAM head has processed an asynchronous error before  
 51205 the call. In this case, the value of *errno* does not reflect the result of *write()*, but reflects the prior  
 51206 error.

51207 XSI The *pwrite()* function shall be equivalent to *write()*, except that it writes into a given position  
 51208 without changing the file pointer. The first three arguments to *pwrite()* are the same as *write()*  
 51209 with the addition of a fourth argument offset for the desired position inside the file.

#### 51210 RETURN VALUE

51211 XSI Upon successful completion, *write()* and *pwrite()* shall return the number of bytes actually  
 51212 written to the file associated with *fdes*. This number shall never be greater than *nbyte*.  
 51213 Otherwise, -1 shall be returned and *errno* set to indicate the error.

#### 51214 ERRORS

51215 XSI The *write()* and *pwrite()* functions shall fail if:

51216 [EAGAIN] The O\_NONBLOCK flag is set for the file descriptor and the thread would be  
 51217 delayed in the *write()* operation.

51218 [EBADF] The *fdes* argument is not a valid file descriptor open for writing.

51219 [EFBIG] An attempt was made to write a file that exceeds the implementation-defined  
 51220 XSI maximum file size or the process' file size limit, and there was no room for  
 51221 any bytes to be written.

51222 [EFBIG] The file is a regular file, *nbyte* is greater than 0, and the starting position is  
 51223 greater than or equal to the offset maximum established in the open file  
 51224 description associated with *fdes*.

51225 [EINTR] The write operation was terminated due to the receipt of a signal, and no data  
 51226 was transferred.

51227 [EIO] The process is a member of a background process group attempting to write  
 51228 to its controlling terminal, TOSTOP is set, the process is neither ignoring nor

51229		blocking SIGTTOU, and the process group of the process is orphaned. This error may also be returned under implementation-defined conditions.
51230		
51231	[ENOSPC]	There was no free space remaining on the device containing the file.
51232	[EPIPE]	An attempt is made to write to a pipe or FIFO that is not open for reading by any process, or that only has one end open. A SIGPIPE signal shall also be sent to the thread.
51233		
51234		
51235 XSR	[ERANGE]	The transfer request size was outside the range supported by the STREAMS file associated with <i>fildev</i> .
51236		
51237		The <i>write()</i> function shall fail if:
51238	[EAGAIN] or [EWOULDBLOCK]	
51239		The file descriptor is for a socket, is marked O_NONBLOCK, and write would block.
51240		
51241	[ECONNRESET]	A write was attempted on a socket that is not connected.
51242	[EPIPE]	A write was attempted on a socket that is shut down for writing, or is no longer connected. In the latter case, if the socket is of type SOCK_STREAM, the SIGPIPE signal is generated to the calling process.
51243		
51244		
51245 XSI		The <i>write()</i> and <i>pwrite()</i> functions may fail if:
51246 XSR	[EINVAL]	The STREAM or multiplexer referenced by <i>fildev</i> is linked (directly or indirectly) downstream from a multiplexer.
51247		
51248	[EIO]	A physical I/O error has occurred.
51249	[ENOBUFS]	Insufficient resources were available in the system to perform the operation.
51250	[ENXIO]	A request was made of a nonexistent device, or the request was outside the capabilities of the device.
51251		
51252 XSR	[ENXIO]	A hangup occurred on the STREAM being written to.
51253 XSR		A write to a STREAMS file may fail if an error message has been received at the STREAM head. In this case, <i>errno</i> is set to the value included in the error message.
51254		
51255		The <i>write()</i> function may fail if:
51256	[EACCES]	A write was attempted on a socket and the calling process does not have appropriate privileges.
51257		
51258	[ENETDOWN]	A write was attempted on a socket and the local network interface used to reach the destination is down.
51259		
51260	[ENETUNREACH]	A write was attempted on a socket and no route to the network is present.
51261 XSI		The <i>pwrite()</i> function shall fail and the file pointer remain unchanged if:
51262 XSI	[EINVAL]	The <i>offset</i> argument is invalid. The value is negative.
51263 XSI	[ESPIPE]	<i>fildev</i> is associated with a pipe or FIFO.

51264 **EXAMPLES**51265 **Writing from a Buffer**

51266 The following example writes data from the buffer pointed to by *buf* to the file associated with  
51267 the file descriptor *fd*.

```
51268 #include <sys/types.h>
51269 #include <string.h>
51270 ...
51271 char buf[20];
51272 size_t nbytes;
51273 ssize_t bytes_written;
51274 int fd;
51275 ...
51276 strcpy(buf, "This is a test\n");
51277 nbytes = strlen(buf);

51278 bytes_written = write(fd, buf, nbytes);
51279 ...
```

51280 **APPLICATION USAGE**

51281 None.

51282 **RATIONALE**

51283 See also the RATIONALE section in *read()*.

51284 An attempt to write to a pipe or FIFO has several major characteristics:

- 51285 • *Atomic/non-atomic*: A write is atomic if the whole amount written in one operation is not  
51286 interleaved with data from any other process. This is useful when there are multiple writers  
51287 sending data to a single reader. Applications need to know how large a write request can be  
51288 expected to be performed atomically. This maximum is called {PIPE\_BUF}. This volume of  
51289 IEEE Std 1003.1-200x does not say whether write requests for more than {PIPE\_BUF} bytes  
51290 are atomic, but requires that writes of {PIPE\_BUF} or fewer bytes shall be atomic.
- 51291 • *Blocking/immediate*: Blocking is only possible with O\_NONBLOCK clear. If there is enough  
51292 space for all the data requested to be written immediately, the implementation should do so.  
51293 Otherwise, the process may block; that is, pause until enough space is available for writing.  
51294 The effective size of a pipe or FIFO (the maximum amount that can be written in one  
51295 operation without blocking) may vary dynamically, depending on the implementation, so it  
51296 is not possible to specify a fixed value for it.

- 51297 • *Complete/partial/deferred*: A write request:

```
51298 int fildes;
51299 size_t nbyte;
51300 ssize_t ret;
51301 char *buf;

51302 ret = write(fildes, buf, nbyte);
```

51303 may return:

51304 complete     *ret*=*nbyte*

51305 partial       *ret*<*nbyte*

51306 This shall never happen if *nbyte*≤{PIPE\_BUF}. If it does happen (with  
51307 *nbyte*>{PIPE\_BUF}), this volume of IEEE Std 1003.1-200x does not guarantee

51308 atomicity, even if  $ret \leq \{PIPE\_BUF\}$ , because atomicity is guaranteed according  
51309 to the amount *requested*, not the amount *written*.

51310 deferred:  $ret = -1$ ,  $errno = [EAGAIN]$

51311 This error indicates that a later request may succeed. It does not indicate that it  
51312 *shall* succeed, even if  $nbyte \leq \{PIPE\_BUF\}$ , because if no process reads from the  
51313 pipe or FIFO, the write never succeeds. An application could usefully count the  
51314 number of times `[EAGAIN]` is caused by a particular value of  
51315  $nbyte > \{PIPE\_BUF\}$  and perhaps do later writes with a smaller value, on the  
51316 assumption that the effective size of the pipe may have decreased.

51317 Partial and deferred writes are only possible with `O_NONBLOCK` set.

51318 The relations of these properties are shown in the following tables:

51319  
51320

Write to a Pipe or FIFO with <code>O_NONBLOCK</code> clear			
Immediately Writable:	None	Some	<i>nbyte</i>
$nbyte \leq \{PIPE\_BUF\}$	Atomic blocking <i>nbyte</i>	Atomic blocking <i>nbyte</i>	Atomic immediate <i>nbyte</i>
$nbyte > \{PIPE\_BUF\}$	Blocking <i>nbyte</i>	Blocking <i>nbyte</i>	Blocking <i>nbyte</i>

51325 If the `O_NONBLOCK` flag is clear, a write request shall block if the amount writable  
51326 immediately is less than that requested. If the flag is set (by `fcntl()`), a write request shall never  
51327 block.

51328  
51329

Write to a Pipe or FIFO with <code>O_NONBLOCK</code> set			
Immediately Writable:	None	Some	<i>nbyte</i>
$nbyte \leq \{PIPE\_BUF\}$	-1, <code>[EAGAIN]</code>	-1, <code>[EAGAIN]</code>	Atomic <i>nbyte</i>
$nbyte > \{PIPE\_BUF\}$	-1, <code>[EAGAIN]</code>	< <i>nbyte</i> or -1, <code>[EAGAIN]</code>	≤ <i>nbyte</i> or -1, <code>[EAGAIN]</code>

51334 There is no exception regarding partial writes when `O_NONBLOCK` is set. With the exception  
51335 of writing to an empty pipe, this volume of IEEE Std 1003.1-200x does not specify exactly when a  
51336 partial write is performed since that would require specifying internal details of the  
51337 implementation. Every application should be prepared to handle partial writes when  
51338 `O_NONBLOCK` is set and the requested amount is greater than `{PIPE_BUF}`, just as every  
51339 application should be prepared to handle partial writes on other kinds of file descriptors.

51340 The intent of forcing writing at least one byte if any can be written is to assure that each write  
51341 makes progress if there is any room in the pipe. If the pipe is empty, `{PIPE_BUF}` bytes must be  
51342 written; if not, at least some progress must have been made.

51343 Where this volume of IEEE Std 1003.1-200x requires -1 to be returned and *errno* set to  
51344 `[EAGAIN]`, most historical implementations return zero (with the `O_NDELAY` flag set, which is  
51345 the historical predecessor of `O_NONBLOCK`, but is not itself in this volume of  
51346 IEEE Std 1003.1-200x). The error indications in this volume of IEEE Std 1003.1-200x were chosen  
51347 so that an application can distinguish these cases from end-of-file. While `write()` cannot receive  
51348 an indication of end-of-file, `read()` can, and the two functions have similar return values. Also,  
51349 some existing systems (for example, Eighth Edition) permit a write of zero bytes to mean that  
51350 the reader should get an end-of-file indication; for those systems, a return value of zero from  
51351 `write()` indicates a successful write of an end-of-file indication.

51352 Implementations are allowed, but not required, to perform error checking for *write()* requests of  
51353 zero bytes.

51354 The concept of a {PIPE\_MAX} limit (indicating the maximum number of bytes that can be  
51355 written to a pipe in a single operation) was considered, but rejected, because this concept would  
51356 unnecessarily limit application writing.

51357 See also the discussion of O\_NONBLOCK in *read()*.

51358 Writes can be serialized with respect to other reads and writes. If a *read()* of file data can be  
51359 proven (by any means) to occur after a *write()* of the data, it must reflect that *write()*, even if the  
51360 calls are made by different processes. A similar requirement applies to multiple write operations  
51361 to the same file position. This is needed to guarantee the propagation of data from *write()* calls  
51362 to subsequent *read()* calls. This requirement is particularly significant for networked file  
51363 systems, where some caching schemes violate these semantics.

51364 Note that this is specified in terms of *read()* and *write()*. The XSI extensions *readv()* and *writv()* |  
51365 also obey these semantics. A new “high-performance” write analog that did not follow these |  
51366 serialization requirements would also be permitted by this wording. This volume of  
51367 IEEE Std 1003.1-200x is also silent about any effects of application-level caching (such as that  
51368 done by *stdio*).

51369 This volume of IEEE Std 1003.1-200x does not specify the value of the file offset after an error is  
51370 returned; there are too many cases. For programming errors, such as [EBADF], the concept is  
51371 meaningless since no file is involved. For errors that are detected immediately, such as  
51372 [EAGAIN], clearly the pointer should not change. After an interrupt or hardware error, however,  
51373 an updated value would be very useful and is the behavior of many implementations.

51374 This volume of IEEE Std 1003.1-200x does not specify behavior of concurrent writes to a file  
51375 from multiple processes. Applications should use some form of concurrency control.

#### 51376 FUTURE DIRECTIONS

51377 None.

#### 51378 SEE ALSO

51379 *chmod()*, *creat()*, *dup()*, *fcntl()*, *getrlimit()*, *lseek()*, *open()*, *pipe()*, *ulimit()*, *writv()*, the Base |  
51380 Definitions volume of IEEE Std 1003.1-200x, <limits.h>, <stropts.h>, <sys/uio.h>, <unistd.h>

#### 51381 CHANGE HISTORY

51382 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 51383 Issue 5

51384 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX  
51385 Threads Extension.

51386 Large File Summit extensions are added.

51387 The *pwrite()* function is added.

#### 51388 Issue 6

51389 The DESCRIPTION states that the *write()* function does not block the thread. Previously this  
51390 said “process” rather than “thread”.

51391 The DESCRIPTION and ERRORS sections are updated so that references to STREAMS are  
51392 marked as part of the XSI STREAMS Option Group.

51393 The following new requirements on POSIX implementations derive from alignment with the  
51394 Single UNIX Specification:

- 51395 • The DESCRIPTION now states that if *write()* is interrupted by a signal after it has  
51396 successfully written some data, it returns the number of bytes written. In earlier versions of  
51397 this volume of IEEE Std 1003.1-200x, it was optional whether *write()* returned the number of  
51398 bytes written, or whether it returned  $-1$  with *errno* set to [EINTR]. This is a FIPS requirement.
- 51399 • The following changes are made to support large files:
- 51400 — For regular files, no data transfer occurs past the offset maximum established in the open  
51401 file description associated with the *files*.
  - 51402 — A second [EFBIG] error condition is added.
- 51403 • The [EIO] error condition is added.
- 51404 • The [EPIPE] error condition is added for when a pipe has only one end open.
- 51405 • The [ENXIO] optional error condition is added.
- 51406 Text referring to sockets is added to the DESCRIPTION.
- 51407 The following changes were made to align with the IEEE P1003.1a draft standard:
- 51408 • The effect of reading zero bytes is clarified.
- 51409 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that  
51410 *write()* results are unspecified for typed memory objects.
- 51411 The following error conditions are added for operations on sockets: [EAGAIN],  
51412 [EWOULDBLOCK], [ECONNRESET], [ENOTCONN], and [EPIPE].
- 51413 The [EIO] error is changed to “may fail”.
- 51414 The [ENOBUFFS] error is added for sockets.
- 51415 The following error conditions are added for operations on sockets: [EACCES], [ENETDOWN],  
51416 and [ENETUNREACH].
- 51417 The *writenv()* function is split out into a separate reference page.

51418 **NAME**

51419 writev — write a vector

51420 **SYNOPSIS**51421 XSI `#include <sys/uio.h>`51422 `ssize_t writev(int fildev, const struct iovec *iov, int iovcnt);`

51423

51424 **DESCRIPTION**

51425 The `writev()` function shall be equivalent to `write()`, except as described below. The `writev()`  
 51426 function shall gather output data from the `iovcnt` buffers specified by the members of the `iov`  
 51427 array: `iov[0]`, `iov[1]`, ..., `iov[iovcnt-1]`. The `iovcnt` argument is valid if greater than 0 and less than  
 51428 or equal to `{IOV_MAX}`, defined in `<limits.h>`.

51429 Each `iovec` entry specifies the base address and length of an area in memory from which data  
 51430 should be written. The `writev()` function shall always write a complete area before proceeding to  
 51431 the next.

51432 If `fildev` refers to a regular file and all of the `iov_len` members in the array pointed to by `iov` are 0,  
 51433 `writev()` shall return 0 and have no other effect. For other file types, the behavior is unspecified.

51434 If the sum of the `iov_len` values is greater than `{SSIZE_MAX}`, the operation shall fail and no data  
 51435 shall be transferred.

51436 **RETURN VALUE**

51437 Upon successful completion, `writev()` shall return the number of bytes actually written.  
 51438 Otherwise, it shall return a value of `-1`, the file-pointer shall remain unchanged, and `errno` shall  
 51439 be set to indicate an error.

51440 **ERRORS**51441 Refer to `write()`.51442 In addition, the `writev()` function shall fail if:51443 [EINVAL] The sum of the `iov_len` values in the `iov` array would overflow an `ssize_t`.51444 The `writev()` function may fail and set `errno` to:51445 [EINVAL] The `iovcnt` argument was less than or equal to 0, or greater than `{IOV_MAX}`.51446 **EXAMPLES**51447 **Writing Data from an Array**

51448 The following example writes data from the buffers specified by members of the `iov` array to the  
 51449 file associated with the file descriptor `fd`.

51450 `#include <sys/types.h>`51451 `#include <sys/uio.h>`51452 `#include <unistd.h>`51453 `...`51454 `ssize_t bytes_written;`51455 `int fd;`51456 `char *buf0 = "short string\n";`51457 `char *buf1 = "This is a longer string\n";`51458 `char *buf2 = "This is the longest string in this example\n";`51459 `int iovcnt;`51460 `struct iovec iov[3];`

```
51461     iov[0].iov_base = buf0;
51462     iov[0].iov_len = strlen(buf0);
51463     iov[1].iov_base = buf1;
51464     iov[1].iov_len = strlen(buf1);
51465     iov[2].iov_base = buf2;
51466     iov[2].iov_len = strlen(buf2);
51467     ...
51468     iovcnt = sizeof(iov) / sizeof(struct iovec);

51469     bytes_written = writev(fd, iov, iovcnt);
51470     ...
```

**51471 APPLICATION USAGE**

51472 None.

**51473 RATIONALE**

51474 Refer to *write()*.

**51475 FUTURE DIRECTIONS**

51476 None.

**51477 SEE ALSO**

51478 *readv()*, *write()*, the Base Definitions volume of IEEE Std 1003.1-200x, <limits.h>, <sys/uio.h>

**51479 CHANGE HISTORY**

51480 First released in Issue 4, Version 2.

**51481 Issue 6**

51482 Split out from the *write()* reference page.

51483 **NAME**

51484           wscanf — convert formatted wide-character input

51485 **SYNOPSIS**

51486           #include &lt;stdio.h&gt;

51487           #include &lt;wchar.h&gt;

51488           int wscanf(const wchar\_t \*restrict *format*, ... );51489 **DESCRIPTION**51490           Refer to *fwscanf()*.

51491 **NAME**

51492        y0, y1, yn — Bessel functions of the second kind

51493 **SYNOPSIS**

```
51494 xSI      #include <math.h>
51495          double y0(double x);
51496          double y1(double x);
51497          double yn(int n, double x);
51498
```

51499 **DESCRIPTION**

51500        The *y0()*, *y1()*, and *yn()* functions shall compute Bessel functions of *x* of the second kind of  
 51501        orders 0, 1, and *n*, respectively.

51502        An application wishing to check for error situations should set *errno* to zero and call  
 51503        *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 51504        *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 51505        zero, an error has occurred.

51506 **RETURN VALUE**

51507        Upon successful completion, these functions shall return the relevant Bessel value of *x* of the  
 51508        second kind.

51509        If *x* is NaN, NaN shall be returned.

51510        If the *x* argument to these functions is negative, *-HUGE\_VAL* or NaN shall be returned, and a  
 51511        domain error may occur.

51512        If *x* is 0.0, *-HUGE\_VAL* shall be returned and a range error may occur.

51513        If the correct result would cause underflow, 0.0 shall be returned and a range error may occur.

51514        If the correct result would cause overflow, *-HUGE\_VAL* or 0.0 shall be returned and a range  
 51515        error may occur.

51516 **ERRORS**

51517        These functions may fail if:

51518        Domain Error    The value of *x* is negative.

51519                        If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero, |  
 51520                        then *errno* shall be set to [EDOM]. If the integer expression (*math\_errhandling* |  
 51521                        & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception |  
 51522                        shall be raised. |

51523        Range Error    The value of *x* is 0.0, or the correct result would cause overflow.

51524                        If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero, |  
 51525                        then *errno* shall be set to [ERANGE]. If the integer expression |  
 51526                        (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the overflow |  
 51527                        floating-point exception shall be raised. |

51528        Range Error    The value of *x* is too large in magnitude, or the correct result would cause  
 51529        underflow.

51530                        If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero, |  
 51531                        then *errno* shall be set to [ERANGE]. If the integer expression |  
 51532                        (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the underflow |  
 51533                        floating-point exception shall be raised. |

51534 **EXAMPLES**

51535 None.

51536 **APPLICATION USAGE**

51537 On error, the expressions (`math_errhandling & MATH_ERRNO`) and (`math_errhandling &`  
51538 `MATH_ERREXCEPT`) are independent of each other, but at least one of them must be non-zero.

51539 **RATIONALE**

51540 None.

51541 **FUTURE DIRECTIONS**

51542 None.

51543 **SEE ALSO**

51544 *feclearexcept()*, *fetetestexcept()*, *isnan()*, *j0()*, the Base Definitions volume of IEEE Std 1003.1-200x, |  
51545 Section 4.18, Treatment of Error Conditions for Mathematical Functions, <**math.h**> |

51546 **CHANGE HISTORY**

51547 First released in Issue 1. Derived from Issue 1 of the SVID.

51548 **Issue 5**

51549 The DESCRIPTION is updated to indicate how an application should check for an error. This  
51550 text was previously published in the APPLICATION USAGE section.

51551 **Issue 6**

51552 The DESCRIPTION is updated to avoid use of the term “must” for application requirements.

51553 The RETURN VALUE and ERRORS sections are reworked for alignment of the error handling |  
51554 with the ISO/IEC 9899:1999 standard.



# Index

1

2	<code>_CS_PATH</code> .....	670	<code>_PC_REC_XFER_ALIGN</code> .....	856
3	<code>_CS_XBS5_ILP32_OFF32_CFLAGS</code> .....	670	<code>_PC_SYMLINK_MAX</code> .....	856
4	<code>_CS_XBS5_ILP32_OFF32_LDFLAGS</code> .....	670	<code>_PC_SYNC_IO</code> .....	856
5	<code>_CS_XBS5_ILP32_OFF32_LIBS</code> .....	670	<code>_PC_VDISABLE</code> .....	856
6	<code>_CS_XBS5_ILP32_OFF32_LINTFLAGS</code> .....	670	<code>_POSIX2 constants</code>	
7	<code>_CS_XBS5_ILP32_OFFBIG_CFLAGS</code> .....	670	in <code>sysconf</code> .....	1969
8	<code>_CS_XBS5_ILP32_OFFBIG_LDFLAGS</code> .....	670	<code>_POSIX2_CHAR_TERM</code> .....	1971
9	<code>_CS_XBS5_ILP32_OFFBIG_LIBS</code> .....	670	<code>_POSIX2_C_BIND</code> .....	1971
10	<code>_CS_XBS5_ILP32_OFFBIG_LINTFLAGS</code> .....	670	<code>_POSIX2_C_DEV</code> .....	1971
11	<code>_CS_XBS5_LP64_OFF64_CFLAGS</code> .....	670	<code>_POSIX2_C_VERSION</code> .....	1971
12	<code>_CS_XBS5_LP64_OFF64_LDFLAGS</code> .....	670	<code>_POSIX2_FORT_DEV</code> .....	1971
13	<code>_CS_XBS5_LP64_OFF64_LIBS</code> .....	670	<code>_POSIX2_FORT_RUN</code> .....	1971
14	<code>_CS_XBS5_LP64_OFF64_LINTFLAGS</code> .....	670	<code>_POSIX2_LOCALEDEF</code> .....	1971
15	<code>_CS_XBS5_LPBIG_OFFBIG_CFLAGS</code> .....	670	<code>_POSIX2_PBS</code> .....	1971
16	<code>_CS_XBS5_LPBIG_OFFBIG_LDFLAGS</code> .....	670	<code>_POSIX2_PBS_ACCOUNTING</code> .....	1971
17	<code>_CS_XBS5_LPBIG_OFFBIG_LIBS</code> .....	670	<code>_POSIX2_PBS_LOCATE</code> .....	1971
18	<code>_CS_XBS5_LPBIG_OFFBIG_LINTFLAGS</code> .....	670	<code>_POSIX2_PBS_MESSAGE</code> .....	1971
19	<code>_exit</code> .....	535	<code>_POSIX2_PBS_TRACK</code> .....	1971
20	<code>_Exit</code> .....	764	<code>_POSIX2_SW_DEV</code> .....	1971
21	<code>_exit</code> .....	764, 2091	<code>_POSIX2_UPE</code> .....	1971
22	<code>_Exit()</code> .....	<b>535</b>	<code>_POSIX2_VERSION</code> .....	1971
23	<code>_FILE</code> .....	581	<code>_POSIX</code> .....	465
24	<code>_IOFBF</code> .....	1779, 1821	<code>_POSIX_ADVISORY_INFO</code> .....	1970
25	<code>_IOLBF</code> .....	833, 1821	<code>_POSIX_ASYNCHRONOUS_IO</code> .....	1970
26	<code>_IONBF</code> .....	1779, 1821	<code>_POSIX_ASYNC_IO</code> .....	856
27	<code>_LINE</code> .....	581	<code>_POSIX_BARRIERS</code> .....	1970
28	<code>_longjmp()</code> .....	<b>536</b>	<code>_POSIX_BASE</code> .....	1970
29	<code>_LVL</code> .....	467	<code>_POSIX_CHOWN_RESTRICTED</code> .....	646, 856, 858
30	<code>_MAX</code> .....	466	<code>_POSIX_CLOCK_SELECTION</code> .....	1970
31	<code>_PC constants</code>		<code>_POSIX_CPUTIME</code> .....	1970
32	used in <code>pathconf</code> .....	856	<code>_POSIX_C_LANG_SUPPORT</code> .....	1970
33	<code>_PC_ALLOC_SIZE_MIN</code> .....	856	<code>_POSIX_C_LANG_SUPPORT_R</code> .....	1970
34	<code>_PC_ASYNC_IO</code> .....	856	<code>_POSIX_C_SOURCE</code> .....	464
35	<code>_PC_CHOWN_RESTRICTED</code> .....	856	<code>_POSIX_DEVICE_IO</code> .....	1970
36	<code>_PC_FILESIZEBITS</code> .....	856	<code>_POSIX_DEVICE_SPECIFIC</code> .....	1970
37	<code>_PC_LINK_MAX</code> .....	856	<code>_POSIX_DEVICE_SPECIFIC_R</code> .....	1970
38	<code>_PC_MAX_CANON</code> .....	856	<code>_POSIX_FD_MGMT</code> .....	1970
39	<code>_PC_MAX_INPUT</code> .....	856	<code>_POSIX_FIFO</code> .....	1970
40	<code>_PC_NAME_MAX</code> .....	856	<code>_POSIX_FILE_ATTRIBUTES</code> .....	1970
41	<code>_PC_NO_TRUNC</code> .....	856	<code>_POSIX_FILE_LOCKING</code> .....	1970
42	<code>_PC_PATH_MAX</code> .....	856	<code>_POSIX_FILE_SYSTEM</code> .....	1970
43	<code>_PC_PIPE_BUF</code> .....	856	<code>_POSIX_FSYNC</code> .....	1970
44	<code>_PC_PRIO_IO</code> .....	856	<code>_POSIX_JOB_CONTROL</code> .....	1970
45	<code>_PC_REC_INCR_XFER_SIZE</code> .....	856	<code>_POSIX_MAPPED_FILES</code> .....	1970
46	<code>_PC_REC_MAX_XFER_SIZE</code> .....	856	<code>_POSIX_MEMLOCK</code> .....	1970
47	<code>_PC_REC_MIN_XFER_SIZE</code> .....	856	<code>_POSIX_MEMLOCK_RANGE</code> .....	1970

48	_POSIX_MEMORY_PROTECTION.....	1970	_POSIX_USER_GROUPS_R.....	1971
49	_POSIX_MESSAGE_PASSING.....	1970	_POSIX_V6_ILP32_OFF32.....	1971
50	_POSIX_MONOTONIC_CLOCK.....	1970	_POSIX_V6_ILP32_OFFBIG.....	1971
51	_POSIX_NETWORKING.....	1970	_POSIX_V6_LP64_OFF64.....	1971
52	_POSIX_NO_TRUNC.....	856	_POSIX_V6_LPBIG_OFFBIG.....	1971
53	_POSIX_OPEN_MAX.....	1017	_POSIX_VDISABLE.....	856
54	_POSIX_PIPE.....	1970	_POSIX_VERSION.....	1971, 2056
55	_POSIX_PRIORITIZED_IO.....	492, 1970	_PROCESS.....	467
56	_POSIX_PRIORITY_SCHEDULING.....	492, 1970	_PTHREAD_THREADS_MAX.....	1557
57	_POSIX_PRIO_IO.....	856	_REGEX_VERSION.....	1971
58	_POSIX_READER_WRITER_LOCKS.....	1970	_SC constants	
59	_POSIX_REALTIME_SIGNALS.....	1970	in sysconf.....	1969
60	_POSIX_REGEX.....	1970	_SC_2_CHAR_TERM.....	1971
61	_POSIX_SAVED_IDS.....	1970	_SC_2_C_BIND.....	1971
62	_POSIX_SEMAPHORES.....	1970	_SC_2_C_DEV.....	1971
63	_POSIX_SHARED_MEMORY_OBJECTS.....	1970	_SC_2_C_VERSION.....	1971
64	_POSIX_SHELL.....	1970	_SC_2_FORT_DEV.....	1971
65	_POSIX_SIGNALS.....	1970	_SC_2_FORT_RUN.....	1971
66	_POSIX_SINGLE_PROCESS.....	1970	_SC_2_LOCALEDEF.....	1971
67	_POSIX_SPAWN.....	1970	_SC_2_PBS_ACCOUNTING.....	1971
68	_POSIX_SPIN_LOCKS.....	1970	_SC_2_PBS_LOCATE.....	1971
69	_POSIX_SPORADIC_SERVER.....	1970	_SC_2_PBS_MESSAGE.....	1971
70	_POSIX_SYNCHRONIZED_IO.....	1970	_SC_2_PBS_TRACK.....	1971
71	_POSIX_SYNC_IO.....	856	_SC_2_SW_DEV.....	1971
72	_POSIX_SYSTEM_DATABASE.....	1970	_SC_2_UPE.....	1971
73	_POSIX_SYSTEM_DATABASE_R.....	1970	_SC_2_VERSION.....	1342, 1971
74	_POSIX_THREADS.....	700, 1971, 2031	_SC_ADVISORY_INFO.....	1970
75	_POSIX_THREAD_ATTR_STACKADDR.....	1971	_SC_AIO_LISTIO_MAX.....	1969
76	_POSIX_THREAD_ATTR_STACKSIZE.....	1971	_SC_AIO_MAX.....	1969
77	_POSIX_THREAD_CPUTIME.....	1971	_SC_AIO_PRIO_DELTA_MAX.....	1969
78	_POSIX_THREAD_PRIORITY_SCHEDULING.....	1971	_SC_ARG_MAX.....	1969
79	.....	1971	_SC_ASYNCIO.....	1970
80	_POSIX_THREAD_PRIO_INHERIT.....	1971	_SC_ATEXIT_MAX.....	1969
81	_POSIX_THREAD_PRIO_PROTECT.....	1971	_SC_BARRIERS.....	1970
82	_POSIX_THREAD_PROCESS_SHARED.....	1583	_SC_BASE.....	1970
83	.....	1971	_SC_BC_BASE_MAX.....	1969
84	_POSIX_THREAD_SAFE_FUNCTIONS.....	700	_SC_BC_DIM_MAX.....	1969
85	.....	1971, 2031	_SC_BC_SCALE_MAX.....	1969
86	_POSIX_THREAD_SPORADIC_SERVER.....	1971	_SC_BC_STRING_MAX.....	1969
87	_POSIX_TIMEOUTS.....	1971	_SC_CHILD_MAX.....	1969
88	_POSIX_TIMERS.....	1971	_SC_CLK_TCK.....	1969, 2027
89	_POSIX_TRACE.....	1971	_SC_CLOCK_SELECTION.....	1970
90	_POSIX_TRACE_EVENT_FILTER.....	1971	_SC_COLL_WEIGHTS_MAX.....	1969
91	_POSIX_TRACE_EVENT_NAME_MAX.....	1426	_SC_CPUTIME.....	1970
92	.....	1428	_SC_C_LANG_SUPPORT.....	1970
93	_POSIX_TRACE_INHERIT.....	1971	_SC_C_LANG_SUPPORT_R.....	1970
94	_POSIX_TRACE_LOG.....	1971	_SC_DELAYTIMER_MAX.....	1969
95	_POSIX_TRACE_SYS_MAX.....	1423	_SC_DEVICE_IO.....	1970
96	_POSIX_TRACE_USER_EVENT_MAX.....	1428	_SC_DEVICE_SPECIFIC.....	1970
97	_POSIX_TYPED_MEMORY_OBJECTS.....	1971	_SC_DEVICE_SPECIFIC_R.....	1970
98	_POSIX_USER_GROUPS.....	1971	_SC_EXPR_NEST_MAX.....	1969

## Index

99	_SC_FD_MGMT .....	1970	_SC_THREAD_ATTR_STACKADDR.....	1971
100	_SC_FIFO .....	1970	_SC_THREAD_ATTR_STACKSIZE .....	1971
101	_SC_FILE_ATTRIBUTES.....	1970	_SC_THREAD_CPUTIME.....	1971
102	_SC_FILE_LOCKING.....	1970	_SC_THREAD_DESTRUCTOR_ITERATIONS.....	1972
103	_SC_FILE_SYSTEM.....	1970	.....	1972
104	_SC_FSYNC.....	1970	_SC_THREAD_KEYS_MAX .....	1972
105	_SC_GETGR_R_SIZE_MAX.....	964, 967, 1969	_SC_THREAD_PRIORITY_SCHEDULING...1971	
106	_SC_GETPW_R_SIZE_MAX.....	1009, 1012, 1969	_SC_THREAD_PRIO_INHERIT .....	1971
107	_SC_IOV_MAX.....	1969	_SC_THREAD_PRIO_PROTECT .....	1971
108	_SC_JOB_CONTROL .....	1970	_SC_THREAD_PROCESS_SHARED.....	1971
109	_SC_LINE_MAX.....	1969	_SC_THREAD_SAFE_FUNCTIONS .....	1971
110	_SC_LOGIN_NAME_MAX.....	1969	_SC_THREAD_SPORADIC_SERVER .....	1971
111	_SC_MEMLOCK.....	1970	_SC_THREAD_STACK_MIN .....	1972
112	_SC_MEMLOCK_RANGE .....	1970	_SC_THREAD_THREADS_MAX.....	1972
113	_SC_MEMORY_PROTECTION.....	1970	_SC_TIMEOUTS.....	1971
114	_SC_MESSAGE_PASSING.....	1970	_SC_TIMERS .....	1971
115	_SC_MONOTONIC_CLOCK.....	1970	_SC_TIMER_MAX .....	1972
116	_SC_MQ_OPEN_MAX.....	1969	_SC_TRACE.....	1971
117	_SC_MQ_PRIO_MAX .....	1969	_SC_TRACE_EVENT_FILTER .....	1971
118	_SC_NETWORKING.....	1970	_SC_TRACE_INHERIT .....	1971
119	_SC_NGROUPS_MAX.....	1969	_SC_TRACE_LOG .....	1971
120	_SC_OPEN_MAX.....	1969	_SC_TTY_NAME_MAX .....	1972
121	_SC_PAGESIZE.....	1253, 1348, 1971	_SC_TYPED_MEMORY_OBJECTS .....	1971
122	_SC_PAGE_SIZE.....	1253, 1971	_SC_TZNAME_MAX .....	1972
123	_SC_PIPE.....	1970	_SC_USER_GROUPS.....	1971
124	_SC_PRIORITIZED_IO .....	1970	_SC_USER_GROUPS_R.....	1971
125	_SC_PRIORITY_SCHEDULING.....	1970	_SC_V6_ILP32_OFF32.....	1971
126	_SC_READER_WRITER_LOCKS .....	1970	_SC_V6_ILP32_OFFBIG.....	1971
127	_SC_REALTIME_SIGNALS .....	1970	_SC_V6_LP64_OFF64 .....	1971
128	_SC_REGEX .....	1970	_SC_V6_LPBIG_OFFBIG .....	1971
129	_SC_REGEX_VERSION .....	1971	_SC_VERSION.....	1971
130	_SC_RE_DUP_MAX .....	1972	_SC_XBS5_ILP32_OFF32 .....	1972
131	_SC_RTSIG_MAX .....	1972	_SC_XBS5_ILP32_OFFBIG .....	1972
132	_SC_SAVED_IDS.....	1970	_SC_XBS5_LP64_OFF64.....	1972
133	_SC_SEMAPHORES.....	1970	_SC_XBS5_LPBIG_OFFBIG.....	1972
134	_SC_SEM_NSEMS_MAX.....	1972	_SC_XOPEN_CRYPT .....	1972
135	_SC_SEM_VALUE_MAX.....	1972	_SC_XOPEN_ENH_I18N .....	1972
136	_SC_SHARED_MEMORY_OBJECTS .....	1970	_SC_XOPEN_LEGACY.....	1972
137	_SC_SHELL .....	1970	_SC_XOPEN_REALTIME.....	1972
138	_SC_SIGNALS .....	1970	_SC_XOPEN_REALTIME_THREADS.....	1972
139	_SC_SIGQUEUE_MAX .....	1972	_SC_XOPEN_SHM .....	1972
140	_SC_SINGLE_PROCESS.....	1970	_SC_XOPEN_UNIX.....	1972
141	_SC_SPAWN .....	1970	_SC_XOPEN_VERSION .....	1972
142	_SC_SPIN_LOCKS.....	1970	_SC_XOPEN_XCU_VERSION .....	1972
143	_SC_SPORADIC_SERVER.....	1970	_setjmp.....	536
144	_SC_STREAM_MAX .....	1972	_setjmp() .....	538
145	_SC_SYMLOOP_MAX.....	1972	_TIME.....	467
146	_SC_SYNCHRONIZED_IO .....	1970	_tolower() .....	539
147	_SC_SYSTEM_DATABASE .....	1970	_toupper().....	540
148	_SC_SYSTEM_DATABASE_R.....	1970	_XBS5_ILP32_OFF32 .....	1972
149	_SC_THREADS.....	1971	_XBS5_ILP32_OFFBIG.....	1972

150	_XBS5_LP64_OFF64.....	1972	aio_read().....	561
151	_XBS5_LPBIG_OFFBIG.....	1972	aio_return().....	564
152	_XOPEN_CRYPT.....	1972	aio_suspend().....	565
153	_XOPEN_ENH_I18N.....	1972	aio_write().....	567
154	_XOPEN_LEGACY.....	1972	alarm().....	570
155	_XOPEN_REALTIME.....	794, 1972	anycast.....	517
156	_XOPEN_REALTIME_THREADS.....	1972	ANYMARK.....	1087
157	_XOPEN_SHM.....	1972	appropriate privileges.....	548, 857
158	_XOPEN_SOURCE.....	464	argc.....	760
159	_XOPEN_UNIX.....	1972	ARG_MAX.....	472, 755, 758, 761, 1969
160	_XOPEN_VERSION.....	1972	asctime().....	572
161	_XOPEN_XCU_VERSION.....	1972	asctime_r.....	572
162	a64l().....	541	asin().....	574
163	ABDAY_1.....	1312	asinf.....	574
164	abort().....	543	asinf().....	576
165	abs().....	544	asinh.....	578
166	accept().....	545	asinhf.....	577
167	access().....	547	asinhf().....	577-578
168	acos().....	550	asinl.....	578
169	acosf.....	550	asinl.....	574
170	acosf().....	552	asinl().....	580
171	acosh().....	553	assert().....	581
172	acoshf.....	553	async-signal-safe.....	1468
173	acoshl.....	553	atan().....	582
174	acosl.....	550	atan2().....	584
175	acosl().....	555	atan2f.....	584
176	ACTION.....	1047	atan2l.....	584
177	address information.....	883	atanf.....	582
178	address string.....	883	atanf().....	586
179	addrinfo structure.....	883	atanh().....	587
180	ADV.....	453	atanhf.....	587
181	ADVANCED REALTIME.....	652, 656, 1277-1278	atanhl.....	587
182	.....	1344, 1346, 1348, 1350, 1352, 1355, 1363	atanl.....	582
183	.....	1366, 1368-1371, 1373, 1375, 1377, 1379	atanl().....	589
184	.....	1381, 1383, 1385-1392, 1453, 1455, 1534	atexit().....	590
185	.....	1579, 1753	ATEXIT_MAX.....	590, 1969
186	ADVANCED REALTIME THREADS.....	1500, 1502-1503	atof().....	591
187	.....	1505, 1507, 1509-1510, 1551, 1633, 1635-1636	atoi().....	592
188	.....	1638-1639	atol().....	594
189	AF.....	468	atoll.....	594
190	AIO.....	453	attributes, clock-resolution.....	526, 1395
191	AIO.....	466	attributes, creation-time.....	526, 1395
192	AIO_ALLDONE.....	556	attributes, generation-version.....	525, 1395
193	aio_cancel().....	556	attributes, inheritance.....	526, 1399
194	AIO_CANCELED.....	556	attributes, log-full-policy.....	524, 526, 1399, 1403
195	aio_error().....	558	attributes, log-max-size.....	526, 1400, 1403
196	aio_fsync().....	559	attributes, max-data-size.....	526, 1403-1404
197	AIO_LISTIO_MAX.....	1163, 1969	attributes, stream-full-policy.....	522-523, 526, 1400
198	AIO_MAX.....	1163, 1969	attributes, stream-min-size.....	526, 1404
199	AIO_NOTCANCELED.....	556	attributes, trace-name.....	526, 1395
200	AIO_PRIO_DELTA_MAX.....	492, 1969	attributes, truncation-status.....	1426

## Index

201	background.....	1799	carg()	614
202	background process.....	2002	cargf.....	614
203	BAR.....	453	cargl.....	614
204	basename().....	595	casin().....	615
205	baud rate functions.....	636	casinf.....	615
206	bcmp().....	597	casinf().....	616
207	bcopy().....	598	casinh().....	617
208	BC_constants		casinhf.....	617
209	in sysconf.....	1969	casinhl.....	617
210	BC_BASE_MAX.....	1969	casinl.....	615
211	BC_DIM_MAX.....	1969	casinl().....	618
212	BC_SCALE_MAX.....	1969	catan().....	619
213	BC_STRING_MAX.....	1969	catanf.....	619
214	BE.....	454	catanf().....	620
215	bind().....	599	catanh().....	621
216	BOOT_TIME.....	742-743	catanhf.....	621
217	broadcasting a condition.....	1519	catanhl.....	621
218	BSD.....	570, 647, 767, 791, 800, 858	catanl.....	619
219	.....	998, 1145, 1233, 1300, 1670, 1710, 1718	catanl().....	622
220	.....	1799, 1842, 1868, 1994, 2017, 2056, 2091	catclose().....	623
221	bsd_signal().....	601	catgets().....	624
222	bsearch().....	603	catopen().....	626
223	btowc().....	605	cbirt().....	628
224	buffer cache.....	909	cbirtf.....	628
225	BUFSIZ.....	1779	cbirtl.....	628
226	BUS_.....	468	ccos().....	629
227	byte-oriented stream.....	486	ccosf.....	629
228	byte-stream mode.....	1667	ccosf().....	630
229	bzero().....	606	ccosh().....	631
230	cabs().....	607	ccoshf.....	631
231	cabsf.....	607	ccoshl.....	631
232	cabsl.....	607	ccosl.....	629
233	acos().....	608	ccosl().....	632
234	acosf.....	608	CD.....	454
235	acosf().....	609	ceil().....	633
236	cacosh().....	610	ceilf.....	633
237	cacoshf.....	610	ceill.....	633
238	cacoshl.....	610	cexp().....	635
239	acosl.....	608	cexpf.....	635
240	acosl().....	611	cexpl.....	635
241	calloc().....	612	cfgetispeed().....	636
242	can.....	451	cfgetospeed().....	638
243	cancel-safe.....	1625	cfsetispeed().....	639
244	cancelability state.....	1558, 1625	cfsetospeed().....	640
245	cancelability states.....	504	change current working directory.....	642
246	cancelability type.....	1558, 1625	change file modes.....	645
247	cancelation cleanup handler.....	1516	change owner and group of file.....	647
248	.....	1528, 1547, 1562	CHAR_MAX.....	1173, 1175
249	cancelation points.....	505	chdir().....	641
250	canceling execution of a thread.....	1511	CHILD_MAX.....	852, 1969
251	canonical name.....	884	chmod().....	643



## Index

303	ctanl	696	dup2	723
304	ctanl()	<b>699</b>	dynamic package initialization	1600
305	ctermid()	<b>700</b>	E2BIG	<b>472</b>
306	ctime()	<b>702</b>	EACCES	<b>472</b>
307	ctime_r	702	EADDRINUSE	<b>472</b>
308	CX	<b>454</b>	EADDRNOTAVAIL	<b>472</b>
309	data key creation	1562	EAFNOSUPPORT	<b>472</b>
310	data messages	488	EAGAIN	<b>472, 478</b>
311	data type	<b>530</b>	EALREADY	<b>472</b>
312	DATEMSK	952	EBADF	<b>472</b>
313	daylight	<b>704, 2048</b>	EBADMSG	<b>472</b>
314	DBL_MANT_DIG	633, 834	EBUSY	<b>473</b>
315	DBL_MAX_EXP	633, 834	ECANCELED	<b>473</b>
316	DBM	705-706	ECHILD	<b>473</b>
317	DBM_	468	ECONNABORTED	<b>473</b>
318	dbm_clearerr()	<b>705</b>	ECONNREFUSED	<b>473</b>
319	dbm_close	705	ECONNRESET	<b>473</b>
320	dbm_delete	705	ecvt()	<b>726</b>
321	dbm_error	705	EDEADLK	<b>473</b>
322	dbm_fetch	705	EDESTADDRREQ	<b>473</b>
323	dbm_firstkey	705	EDOM	<b>473</b>
324	DBM_INSERT	705	EDQUOT	<b>473</b>
325	dbm_nextkey	705	EEXIST	<b>473</b>
326	dbm_open	705	EFAULT	<b>473</b>
327	DBM_REPLACE	705	EFBIG	<b>473</b>
328	dbm_store	705	effective group ID	647, 761, 970
329	DEAD_PROCESS	742-743	effective user ID	548, 761, 1145
330	deferred cancelability	1558	EHOSTUNREACH	473
331	delay process execution	1885	EIDRM	<b>473</b>
332	DELAYTIMER_MAX	1969	Eighth Edition UNIX	2162
333	dependency ordering	717	EILSEQ	<b>474, 488</b>
334	descriptive name	883	EINPROGRESS	<b>474, 492</b>
335	destroying a mutex	1568	EINTR	<b>474, 507, 1420</b>
336	destroying condition variables	1523	EINVAL	<b>474, 1404, 1420</b>
337	destructor functions	1562	EIO	474
338	detaching a thread	1544	EISCONN	474
339	difftime()	<b>708</b>	EISDIR	474
340	DIR	530, 664, 1324, 1673, 1675, 1713, 1740, 2010	ELOOP	474
341	directive	861, 892, 925, 934, 1936	ELSIZE	1203
342	directory operations	1325	EMFILE	474
343	dirent structure	1325	EMLINK	474
344	dirname()	<b>709</b>	EMPTY	743
345	div()	<b>711</b>	EMSGSIZE	474
346	dlclose()	<b>712</b>	EMULTIHOP	474
347	dlopen()	<b>714</b>	ENAMETOOLONG	<b>475</b>
348	dlopen()	<b>716</b>	encrypt()	<b>728</b>
349	dlsym()	<b>719</b>	endgrent()	<b>730</b>
350	dot	1325, 1710	endhostent()	<b>732</b>
351	dot-dot	1325, 1710	endnetent()	<b>734</b>
352	drand48()	<b>721</b>	endprotoent()	<b>736</b>
353	dup()	<b>723</b>	endpwent()	<b>738</b>

354	endservent()	740	additional	478
355	enduxent()	742	ESPIPE	477
356	ENETDOWN	475	ESRCH	477
357	ENETRESET	475	establishing cancelation handlers	1516
358	ENETUNREACH	475	ESTALE	477
359	ENFILE	475	ETIME	477
360	ENOBUFS	475	ETIMEDOUT	477
361	ENODATA	475	ETXTBSY	477
362	ENODEV	475	EWOLDBLOCK	477
363	ENOENT	475	examine and change blocked signals	1632
364	ENOEXEC	475	examine and change signal action	1842
365	ENOLCK	475	EXDEV	478
366	ENOLINK	475	exec	754
367	ENOMEM	475	of shell scripts	760
368	ENOMSG	475	exec family	548, 663, 790, 831, 854
369	ENOPROTOPT	475	.....	1468, 1800, 2091
370	ENOSPC	476	execl	754
371	ENOSR	476	execle	754
372	ENOSTR	476	execlp	754
373	ENOSYS	476	execute a file	760
374	ENOTCONN	476	execution time monitoring	500
375	ENOTDIR	476	execv	754
376	ENOTEMPTY	476	execve	754
377	ENOTSOCK	476	execvp	754
378	ENOTSUP	476	exit()	764
379	ENOTTY	476	EXIT_FAILURE	764
380	ENTRY	1047	EXIT_SUCCESS	764, 767
381	environ	745, 760	exp()	769
382	envp	760	exp2()	771
383	ENXIO	476	exp2f	771
384	EOPNOTSUPP	476	exp2l	771
385	E_OVERFLOW	476	expf	769
386	EPERM	476	expl	769
387	EPIPE	476	expm1()	773
388	EPROTO	477	expm1f	773
389	EPROTONOSUPPORT	477	expm1l	773
390	EPROTOTYPE	477	EXPR_NEST_MAX	1969
391	erand48	721	extension	
392	erand48()	746	CX	454
393	ERANGE	477	OH	456
394	erf()	747	XSI	460
395	erfc()	749	extensions to setlocale	1794
396	erfcf	749	fabs()	775
397	erfcl	749	fabsf	775
398	erff	747	fabsl	775
399	erff()	751	fattach()	776
400	erfl	747, 751	fchdir()	779
401	EROFS	477	fchmod()	780
402	errno	752	fchown()	782
403	error descriptions	940	fclose()	784
404	error numbers	471	fcntl()	786

## Index

405	fcvt.....	726	file	
406	fcvt().....	<b>793</b>	locking.....	789
407	FD.....	<b>454</b>	file accessibility.....	548
408	fdatasync().....	<b>794</b>	file control.....	789
409	fdetach().....	<b>795</b>	FILE object.....	484
410	fdim().....	<b>797</b>	file permission bits.....	548
411	fdimf.....	797	file permissions.....	548, 858, 1901
412	fdiml.....	797	file position indicator.....	484
413	fdopen().....	<b>799</b>	fileno().....	<b>831</b>
414	FD_CLOEXEC.....	485, 626, 755, 786, 1059	FILESIZEBITS.....	856
415	.....	1317, 1325, 1335, 1356, 1363, 1823	FIND.....	1047
416	FD_CLR.....	1463	find string token.....	1950
417	FD_CLR().....	<b>534</b>	flockfile().....	<b>832</b>
418	FD_ISSET.....	534, 1463	floor().....	<b>834</b>
419	FD_SET.....	534, 1463	floorf.....	834
420	FD_ZERO.....	534, 1463	floorl.....	834
421	feature test macro.....	463, 945	FLT_RADIX.....	1192
422	_POSIX_C_SOURCE.....	464	FLT_ROUNDS.....	836
423	_XOPEN_SOURCE.....	464	FLUSH.....	468
424	feclearexcept().....	<b>801</b>	FLUSHR.....	1081
425	fegetenv().....	<b>802</b>	FLUSHRW.....	1081
426	fegetexceptflag().....	<b>803</b>	FLUSHW.....	1081
427	fegetround().....	<b>804</b>	fma().....	<b>836</b>
428	feholdexcept().....	<b>806</b>	fmaf.....	836
429	feof().....	<b>807</b>	fmal.....	836
430	feraiseexcept().....	<b>808</b>	fmax().....	<b>838</b>
431	ferror().....	<b>809</b>	fmaxf.....	838
432	fesetenv.....	802	fmaxl.....	838
433	fesetenv().....	<b>810</b>	fmin().....	<b>839</b>
434	fesetexceptflag.....	803	fminf.....	839
435	fesetexceptflag().....	<b>811</b>	fminl.....	839
436	fesetround.....	804	FMNAMESZ.....	1080
437	fesetround().....	<b>812</b>	fmod().....	<b>840</b>
438	fetestexcept().....	<b>813</b>	fmodf.....	840
439	feupdateenv().....	<b>815</b>	fmodl.....	840
440	fflush().....	<b>817</b>	fntmsg().....	<b>842</b>
441	ffs().....	<b>820</b>	fnmatch().....	<b>845</b>
442	fgetc().....	<b>821</b>	FNM.....	468
443	fgetpos().....	<b>823</b>	FNM_NOESCAPE.....	845
444	fgets().....	<b>825</b>	FNM_NOMATCH.....	845
445	fgetwc().....	<b>827</b>	FNM_PATHNAME.....	845
446	fgetws().....	<b>829</b>	FNM_PERIOD.....	845
447	FIFO.....	1235-1236, 1321, 2161	fopen().....	<b>847</b>
448	FILE.....	530, 650, 784, 799, 807, 809	FOPEN_MAX.....	799, 848, 1341, 2029
449	.....	817, 821, 823, 825, 827, 829, 831-832	foreground.....	1799
450	.....	847, 861, 873, 875, 877, 879-880, 887	fork handler.....	1469
451	.....	892, 899, 902, 911, 923-925, 932, 934	fork().....	<b>851</b>
452	.....	943-944, 1036, 1331, 1341, 1642-1643	forkall.....	854
453	.....	1656, 1712, 1779, 1821, 1904, 2029	format of entries.....	<b>461</b>
454	.....	2058-2059, 2076, 2078, 2080, 2082, 2084	fpathconf().....	<b>856</b>
455	.....	2086	fpclassify().....	<b>860</b>

456	FPE_.....	468	fwrite().....	932
457	fprintf().....	861	fwscanf().....	934
458	fputc().....	873	F_DUPFD.....	723, 786, 788-789
459	fputs().....	875	F_GETFD.....	786, 788-789
460	fputwc().....	877	F_GETFL.....	786, 788-789
461	fputws().....	879	F_GETLK.....	787-789
462	FQDN.....	986	F_GETOWN.....	786, 788
463	FR.....	454	F_LOCK.....	1181
464	fread().....	880	F_RDLCK.....	789
465	free().....	882	F_SETFD.....	786, 788-790
466	freeaddrinfo().....	883	F_SETFL.....	786, 788-789
467	freopen().....	887	F_SETLK.....	787-789
468	frexp().....	890	F_SETLKW.....	505, 787-789
469	frexpf.....	890	F_SETOWN.....	786, 789
470	frexpl.....	890	F_TEST.....	1181
471	FSC.....	455	F_TLOCK.....	1181
472	fscanf().....	892	F_ULOCK.....	1181
473	fseek().....	899	F_UNLCK.....	787
474	fseeko.....	899	F_WRLCK.....	789
475	fsetpos().....	902	gai_strerror().....	940
476	fstat().....	904	gcvt.....	726
477	fstatvfs().....	906	gcvt().....	941
478	fsync().....	909	generation-version attribute.....	525, 1395
479	ftell().....	911	get configurable path name variables.....	858
480	ftello.....	911	get configurable system variables.....	1973
481	ftime().....	913	get file status.....	1901
482	ftok().....	915	get process times.....	2027
483	fruncate().....	917	get supplementary group IDs.....	969
484	ftrylockfile.....	832	get system time.....	2017
485	ftrylockfile().....	919	get thread ID.....	1623
486	FTW.....	468, 1307-1308	get user name.....	980
487	ftw().....	920	getaddrinfo.....	883
488	FTW_CHDIR.....	1307	getaddrinfo().....	942
489	FTW_D.....	920, 1307	GETALL.....	1760
490	FTW_DEPTH.....	1307	getc().....	943
491	FTW_DNR.....	920, 1307-1308	getchar().....	946
492	FTW_DP.....	1307	getchar_unlocked.....	944
493	FTW_F.....	920, 1307	getchar_unlocked().....	947
494	FTW_MOUNT.....	1307	getcontext().....	948
495	FTW_NS.....	920, 1307-1308	getcwd().....	950
496	FTW_PHYS.....	1307	getc_unlocked().....	944
497	FTW_SL.....	920, 1307	getdate().....	952
498	FTW_SLN.....	1307	getdate_err.....	952
499	fully-qualified domain name.....	986	getegid().....	957
500	functions.....	463	getenv.....	760
501	implementation.....	463	getenv().....	958
502	use.....	463	geteuid().....	961
503	funlockfile.....	832	getgid().....	962
504	funlockfile().....	923	getgrnt.....	730
505	fwide().....	924	getgrnt().....	963
506	fwprintf().....	925	getgrgid().....	964

## Index

507	getgrgid_r.....	964	getservbyport().....	1022
508	getgrnam().....	967	getservent.....	740
509	getgrnam_r.....	967	getservent().....	1023
510	getgroups().....	969	getsid().....	1024
511	gethostbyaddr().....	971	getsockname().....	1025
512	gethostbyname.....	971	getsockopt().....	1026
513	gethostbyname().....	973	getsubopt().....	1029
514	gethostent.....	732	gettimeofday().....	1033
515	gethostent().....	974	getuid().....	1034
516	gethostid().....	975	getutxent.....	742
517	gethostname().....	976	getutxent().....	1035
518	getitimer().....	977	getutxid.....	742, 1035
519	getlogin().....	979	getutxline.....	742, 1035
520	getlogin_r.....	979	GETVAL.....	1760-1761
521	getmsg().....	982	getwc().....	1036
522	getnameinfo().....	986	getwchar().....	1037
523	GETNCNT.....	1760-1761	getwd.....	951
524	getnetbyaddr.....	734	getwd().....	1038
525	getnetbyaddr().....	988	GETZCNT.....	1760-1761
526	getnetbyname.....	734	glob().....	1039
527	getnetbyname().....	989	globfree.....	1039
528	getnetent.....	734	GLOB.....	468
529	getnetent().....	990	GLOB_constants	
530	getopt().....	991	error returns of glob.....	1041
531	getpeername().....	996	used in glob.....	1039
532	getpgid().....	997	GLOB_ABORTED.....	1041
533	getpgrp().....	998	GLOB_APPEND.....	1039-1040
534	GETPID.....	1760-1761	GLOB_DOOFFS.....	1039-1040
535	getpid().....	999	GLOB_ERR.....	1039, 1041
536	getpmsg.....	982	GLOB_MARK.....	1040
537	getpmsg().....	1000	GLOB_NOCHECK.....	1040-1041
538	getppid().....	1001	GLOB_NOESCAPE.....	1040
539	getpriority().....	1002	GLOB_NOMATCH.....	1041
540	getprotobyname.....	736	GLOB_NOSORT.....	1040
541	getprotobyname().....	1005	GLOB_NOSPACE.....	1041
542	getprotobynumber.....	736	gmtime().....	1043
543	getprotobynumber().....	1006	gmtime_r.....	1043
544	getprotoent.....	736	grantpt().....	1045
545	getprotoent().....	1007	granularity of clock.....	913
546	getpwent.....	738	HALT.....	843
547	getpwent().....	1008	hcreate.....	1047
548	getpwnam().....	1009	hcreate().....	1047
549	getpwnam_r.....	1009	hdestroy.....	1047
550	getpwuid().....	1012	hdestroy().....	1050
551	getpwuid_r.....	1012	high resolution sleep.....	1300
552	getrlimit().....	1015	host name.....	883
553	getrusage().....	1018	hsearch().....	1051
554	gets().....	1020	htonl().....	1052
555	getservbyname.....	740	htons.....	1052
556	getservbyname().....	1021	htons().....	1053
557	getservbyport.....	740	HUGE_VAL.....	587, 633, 681, 769, 771, 773

558	.....797, 834, 1054, 1153, 1157, 1184, 1188	INT_MAX.....1065
559	.....1190, 1302, 1304, 1458, 1715, 1720, 1724	INT_MIN.....544
560	.....1882, 1944, 1982, 2014, 2118	IN_.....468
561	HUGE_VALF.....1944, 2118	ioctl()..... <b>1080</b>
562	HUGE_VALL.....1944, 2118	IOV_.....468
563	hypot()..... <b>1054</b>	IOV_MAX.....1969
564	hypotf.....1054	IP6..... <b>455</b>
565	hypotl.....1054	IPC..... <b>489</b> , 1283, 1285, 1288, 1290
566	h_errno..... <b>1046</b>	.....1765, 1769, 1833, 1835
567	iconv()..... <b>1056</b>	IPC_.....468
568	iconv_close()..... <b>1058</b>	IPC_constants
569	iconv_open()..... <b>1059</b>	used in semctl.....1760
570	IEEE Std 754-1985.....453	used in shmctl.....1831
571	IEEE Std 854-1987.....453	IPC_CREAT.....1284, 1763, 1834
572	IF_.....468	IPC_EXCL.....1284, 1763
573	if_freenameindex()..... <b>1061</b>	IPC_NOWAIT.....1286-1287, 1289-1290, 1766
574	if_indextoname()..... <b>1062</b>	IPC_PRIVATE.....1284, 1763, 1834
575	if_nameindex()..... <b>1063</b>	IPC_RMID.....1282, 1761, 1831
576	if_nametoindex()..... <b>1064</b>	IPC_SET.....1282, 1761, 1831
577	ILL_.....468	IPC_STAT.....1282, 1760, 1831
578	ilogb()..... <b>1065</b>	IPPORT_.....468
579	ilogbf.....1065	IPPROTO_.....468
580	ilogbl.....1065	IPv4.....517
581	imaxabs()..... <b>1067</b>	IPv4-compatible address.....518
582	imaxdiv()..... <b>1068</b>	IPv4-mapped address.....518
583	implementation-defined.....451	IPv6.....517
584	IMPLINK_.....468	compatibility with IPv4.....518
585	INADDR_.....468	interface identification.....518
586	index()..... <b>1069</b>	options.....519
587	inet_addr()..... <b>1070</b>	IPv6 address
588	inet_ntoa.....1070	anycast.....517
589	inet_ntoa()..... <b>1072</b>	loopback.....518
590	inet_ntop()..... <b>1073</b>	multicast.....517
591	inet_pton.....1073	unicast.....517
592	Inf.....574	unspecified.....518
593	INF.....864, 928	IP_.....468
594	INFINITY.....864, 928	isalnum()..... <b>1092</b>
595	INFO.....843	isalpha()..... <b>1093</b>
596	inheritance attribute.....526, 1399	isascii()..... <b>1094</b>
597	init.....767, 1145	isastream()..... <b>1095</b>
598	initialize a named semaphore.....1749	isatty()..... <b>1096</b>
599	initialize an unnamed semaphore.....1746	isblank()..... <b>1097</b>
600	initializing a mutex.....1568	iscntrl()..... <b>1098</b>
601	initializing condition variables.....1523	isdigit()..... <b>1099</b>
602	initstate()..... <b>1075</b>	isfinite()..... <b>1100</b>
603	INIT_PROCESS.....742-743	isgraph()..... <b>1101</b>
604	input and output rationale.....1669	isgreater()..... <b>1102</b>
605	insque()..... <b>1077</b>	isgreaterequal()..... <b>1103</b>
606	INT.....468	isinf()..... <b>1104</b>
607	international environment.....1794	isless()..... <b>1105</b>
608	Internet Protocols.....516	islessequal()..... <b>1106</b>

## Index

609	islessgreater()	1107	I_SENDFD	1085-1086
610	islower()	1108	I_SETCLTIME	661, 1087
611	isnan()	1110	I_SETSIG	1081-1082
612	isnormal()	1111	I_SRDOPT	1082-1083, 1667
613	ISO C standard	453, 570, 760, 789, 945	I_STR	1084
614		1205, 1663, 1710, 1794, 1842, 1868, 2017	I_SWROPT	1085, 2159
615	isprint()	1112	I_UNLINK	1088
616	ispunct()	1113	j0()	1141
617	isspace()	1114	j1	1141
618	isunordered()	1115	jn	1141
619	isupper()	1116	job control	767, 998, 1145, 1799, 1811, 1973, 2091
620	iswalnum()	1117	jrand48	721
621	iswalpha()	1119	jrand48()	1143
622	iswblank()	1121	kill()	1144
623	iswcntrl()	1122	killpg()	1147
624	iswctype()	1124	l64a	541
625	iswdigit()	1126	l64a()	1148
626	iswgraph()	1127	labs()	1149
627	iswlower()	1129	LANG	626
628	iswprint()	1131	last close	1827
629	iswpunct()	1133	LASTMARK	1087
630	iswspace()	1135	lchown()	1150
631	iswupper()	1137	lcong48	721
632	iswxdigit()	1139	lcong48()	1152
633	isxdigit()	1140	LC_ALL	755, 1175, 1312, 1793, 1795
634	ITIMER_PROF	977	LC_COLLATE	1039-1040, 1793-1794, 1911
635	ITIMER_REAL	977		1960, 2102, 2138
636	ITIMER_VIRTUAL	977	LC_CTYPE	605, 1124, 1213, 1215, 1217
637	I_	468		1219-1220, 1222, 1224, 1793-1794, 2034-2038
638	I_ATMARK	1086-1087		2097, 2113, 2129, 2140-2141, 2143-2144
639	I_CANPUT	1087	LC_MESSAGES	626, 1793-1794, 1917
640	I_CKBAND	1087	LC_MONETARY	1175, 1793-1794, 1920
641	I_FDINSERT	1083	LC_NUMERIC	726, 862, 892, 925, 934
642	I_FIND	1082		1175, 1793-1794, 1920, 1944, 2118
643	I_FLUSH	1080	LC_TIME	953, 1312, 1793-1794
644	I_FLUSHBAND	1081	ldexp()	1153
645	I_GETBAND	1087	ldexpf	1153
646	I_GETCLTIME	1087	ldexpl	1153
647	I_GETSIG	1082	ldiv()	1155
648	I_GRDOPT	1083, 1667	legacy	451
649	I_GWROPT	1085	LEGACY	2071
650	I_LINK	1088	lfind	1203
651	I_LIST	1086	lfind()	1156
652	I_LOOK	1080	lgamma()	1157
653	I_NREAD	1083	lgammaf	1157
654	I_PEEK	1082	lgammal	1157
655	I_PLINK	1089	LIFO	507
656	I_POP	1080	LINE_MAX	1969
657	I_PUNLINK	1089	link to a file	1161
658	I_PUSH	1080	link()	1159
659	I_RECVFD	472, 1086	LINK_MAX	474, 856, 1159, 1709

660	LIO_.....	466	logl.....	1184
661	lio_listio().....	<b>1162</b>	logl().....	<b>1195</b>
662	LIO_NOP.....	1162	LOG.....	468
663	LIO_NOWAIT.....	1162	LOG_constants in syslog.....	<b>666</b>
664	LIO_READ.....	1162	LOG_ALERT.....	666
665	LIO_WAIT.....	1162	LOG_CONS.....	667
666	LIO_WRITE.....	1162	LOG_CRIT.....	666
667	list directed I/O.....	1164	LOG_DEBUG.....	666
668	listen().....	<b>1165</b>	LOG_EMERG.....	666
669	llabs.....	1149	LOG_ERR.....	666
670	llabs().....	<b>1167</b>	LOG_INFO.....	666
671	lldiv.....	1155	LOG_LOCAL.....	666
672	lldiv().....	<b>1168</b>	LOG_NDELAY.....	667
673	LLONG_MAX.....	1953, 2125	LOG_NOTICE.....	666
674	LLONG_MIN.....	1953	LOG_NOWAIT.....	667
675	,.....	2125	LOG_ODELAY.....	667
676	llrint().....	<b>1169</b>	LOG_PID.....	667
677	llrintf.....	1169	LOG_USER.....	666-667
678	llrintl.....	1169	LOG_WARNING.....	666
679	llround().....	<b>1171</b>	longjmp().....	<b>1196</b>
680	llroundf.....	1171	LONG_MAX.....	1953, 2125
681	llroundl.....	1171	LONG_MIN.....	1953, 2125
682	load ordering.....	717	rand48.....	721
683	localeconv().....	<b>1173</b>	rand48().....	<b>1198</b>
684	localtime().....	<b>1178</b>	rint().....	<b>1199</b>
685	localtime_r.....	1178	rintf.....	1199
686	lockf().....	<b>1181</b>	rintl.....	1199
687	locking.....	789	rround().....	<b>1201</b>
688	advisory.....	790	rroundf.....	1201
689	mandatory.....	790	rroundl.....	1201
690	locking and unlocking a mutex.....	1576	rsearch().....	<b>1203</b>
691	log().....	<b>1184</b>	rseek().....	<b>1205</b>
692	log-full-policy attribute .524, 526, 1399, 1403, 1424		rstat().....	<b>1207</b>
693	log-max-size attribute.....	526, 1400, 1403	L_ctermid.....	700
694	log10().....	<b>1186</b>	l_sysid.....	790
695	log10f.....	1186	makecontext().....	<b>1209</b>
696	log10l.....	1186	malloc().....	<b>1211</b>
697	log1p().....	<b>1188</b>	manipulate signal sets.....	1848
698	log1pf.....	1188	mappings.....	1256
699	log1pl.....	1188	MAP.....	466, 468
700	log2().....	<b>1190</b>	MAP_FAILED.....	1256
701	log2f.....	1190	MAP_FIXED.....	1252, 1255
702	log2l.....	1190	MAP_PRIVATE.....	851, 1252, 1256, 1260, 1292
703	logb().....	<b>1192</b>	MAP_SHARED.....	854, 1252-1253
704	logbf.....	1192	max-data-size attribute.....	526, 1403-1404
705	logbl.....	1192	MAXPATHLEN.....	709
706	logf.....	1184	MAX_CANON.....	856
707	logf().....	<b>1194</b>	MAX_INPUT.....	856
708	login shell.....	760	may.....	452
709	LOGIN_NAME_MAX.....	979, 1969	mblen().....	<b>1213</b>
710	LOGIN_PROCESS.....	742-743	mbrlen().....	<b>1215</b>

## Index

711	mbrtowc()	1217	MM_NOTOK	843
712	mbsinit()	1219	MM_NRECOV	842
713	mbsrtowcs()	1220	MM_NULLMC	842
714	mbstowcs()	1222	MM_OK	843
715	mbtowc()	1224	MM_OPSYS	842
716	MB_CUR_MAX	1213, 1215, 1217, 1224, 2097, 2141	MM_PRINT	842, 844
717	MC1	455	MM_RECOVER	842
718	MC2	455	MM_SOFT	842
719	MCL	466	MM_UTIL	842
720	MCL_CURRENT	1249	MM_WARNING	843
721	MCL_FUTURE	1249	modf()	1258
722	memccpy()	1226	modff	1258
723	memchr()	1227	modfl	1258
724	memcmp()	1228	MON	456
725	memcpy()	1229	MORECTL	983
726	MEMLOCK_FUTURE	1256	MOREDATA	983
727	memmove()	1230	MPR	456
728	memory management	493	mprotect()	1260
729	memory protection option	1255	mq_close()	1262
730	memset()	1231	mq_getattr()	1263
731	message catalog descriptor	755, 760, 764	mq_notify()	1265
732	message parts	489	mq_open()	1267
733	message priority	488	MQ_OPEN_MAX	1969
734	high-priority	488	MQ_PRIO_MAX	1273-1274, 1969
735	normal	488	mq_receive()	1270
736	priority	488	mq_send()	1273
737	message-discard mode	1667	mq_setattr()	1275
738	message-nondiscard mode	1667	mq_timedreceive	1270
739	MF	455	mq_timedreceive()	1277
740	MINSIGSTKSZ	1845	mq_timedsend	1273
741	mkdir()	1232	mq_timedsend()	1278
742	mkfifo()	1235	mq_unlink()	1279
743	mknod()	1238	mrnd48	721
744	mkstemp()	1241	mrnd48()	1281
745	mktemp()	1243	MSG	456, 468
746	mktime()	1245	msgctl()	1282
747	ML	455	msgget()	1284
748	mlock()	1247	msgrcv()	1286
749	mlockall()	1249	msgsnd()	1289
750	MLR	455	MSGVERB	843-844
751	mmap()	1251	MSG_	468
752	MM_APPL	842	MSG_ANY	982
753	MM_CONSOLE	842	MSG_BAND	982, 1648
754	MM_ERROR	843-844	MSG_EOR	1888, 1890
755	mm_FIRM	842	MSG_HIPRI	982, 1648
756	MM_HALT	843	MSG_NOERROR	1286-1287
757	MM_HARD	842	msg_perm	490
758	MM_INFO	843	msqid	490
759	MM_NOCON	843	msync()	1292
760	MM_NOMSG	843	MS	466, 468
761	MM_NOSEV	843	MS_ASYNC	1253, 1292

762	MS_INVALIDATE	1292-1293	nl_langinfo()	1312
763	MS_SYNC	1253, 1292	nohup utility	761
764	multicast	517	non-local jumps	1868
765	munlock	1247	non-volatile storage	909
766	munlock()	1295	nrand48	721
767	munlockall	1249	nrand48()	1314
768	munlockall()	1296	ntohl	1052
769	munmap()	1297	ntohl()	1315
770	mutex attributes	1583	ntohs	1052
771	mutex initialization attributes	1582	ntohs()	1316
772	mutex performance	1583	NULL	671, 700, 707, 714, 719, 1256, 1675
773	MUXID_ALL	1088-1089	NUM_EMPL	1048
774	MUXID_R	468	NZERO	1002, 1310
775	MX	456	OB	456
776	M_	468	obsolescent	636
777	name information	986	OF	456
778	name space	464	OH	456
779	NAME_MAX	475, 547, 626, 641, 643	OLD_TIME	742-743
780		646, 758, 776, 795, 848, 856, 887, 906	open a file	1321
781		915, 1150, 1159, 1207, 1232, 1235, 1239	open a named semaphore	1749
782		1268, 1279, 1308, 1319, 1324, 1456, 1673	open a shared memory object	1824
783		1677, 1684, 1709, 1717, 1749, 1757, 1824	open()	1317
784		1827, 1899, 1966, 2040, 2061, 2069, 2071	opendir()	1324
785	NaN	574	openlog	666
786	NAN	864	openlog()	1327
787	NaN	864	OPEN_MAX	474, 626, 723, 738, 789
788	NAN	928		848, 887, 920, 964, 967, 979, 1012, 1059
789	NaN	928		1267, 1308, 1319, 1324, 1335, 1363, 1366
790	nan()	1299		1969, 2029
791	nanf	1299	optarg	991, 1328
792	nanl	1299	opterr	991, 1328
793	nanosleep()	1300	optind	991, 1328
794	NDEBUG	471, 581	option	
795	nearbyint()	1302	ADV	453
796	nearbyintf	1302	AIO	453
797	nearbyintl	1302	BAR	453
798	network interfaces	509	BE	454
799	NEW_TIME	742-743	CD	454
800	nextafter()	1304	CPT	454
801	nextafterf	1304	CS	454
802	nextafterl	1304	FD	454
803	nexttoward	1304	FR	454
804	nexttoward()	1306	FSC	455
805	nexttowardf	1304, 1306	IP6	455
806	nexttowardl	1304, 1306	MC1	455
807	nftw()	1307	MC2	455
808	NGROUPS_MAX	970, 1969	MF	455
809	nice()	1310	ML	455
810	NLSPATH	626	MLR	455
811	NL_ARGMAX	861, 892, 925, 934	MON	456
812	NL_CAT_LOCALE	626	MPR	456

## Index

813	MSG.....	456		
814	MX.....	456		
815	PIO.....	457		
816	PS.....	457		
817	RS.....	457		
818	RTS.....	457		
819	SD.....	457		
820	SEM.....	457		
821	SHM.....	457		
822	SIO.....	457		
823	SPI.....	458		
824	SPN.....	458		
825	SS.....	458		
826	TCT.....	458		
827	TEF.....	458		
828	THR.....	458		
829	TMO.....	458		
830	TMR.....	459		
831	TPI.....	459		
832	TPP.....	459		
833	TPS.....	459		
834	TRC.....	459		
835	TRI.....	459		
836	TRL.....	459		
837	TSA.....	459		
838	TSF.....	460		
839	TSH.....	460		
840	TSP.....	460		
841	TSS.....	460		
842	TYM.....	460		
843	UP.....	460		
844	XSR.....	461		
845	optopt.....	991, 994, 1328		
846	optstring.....	994		
847	orphaned process group.....	767		
848	O_ constants			
849	used in open().....	1317		
850	used in posix_openpt().....	1353		
851	O_ACCMODE.....	786		
852	O_APPEND.....	491, 567, 705, 800, 1317, 2157		
853	O_CREAT.....	687, 1267-1268, 1279, 1317-1319		
854	.....	1743, 1748, 1823-1825		
855	O_DSYNC.....	559, 1317-1318, 1667, 2158		
856	O_EXCL.....	1268, 1317-1318, 1748, 1823-1824		
857	O_NDELAY.....	2162		
858	O_NOCTTY.....	1318, 1322, 1353		
859	O_NONBLOCK.....	474, 661, 784, 817, 821, 827		
860	.....	873, 877, 900, 902, 983, 1084, 1086, 1268		
861	.....	1270, 1273, 1275, 1318-1320, 1335, 1338		
862	.....	1649, 1666, 2158, 2161		
863	O_RDONLY.....	709, 1267, 1317-1319, 1322, 1823		
				1825
	O_RDWR.....	779, 1181, 1267, 1317-1321, 1353		
	.....	1823, 1825		
	O_RSYNC.....	1318, 1667		
	O_SYNC.....	559, 1318, 1667, 2158		
	O_TRUNC.....	687, 1318, 1320, 1322, 1824-1825		
	O_WRONLY.....	687, 779, 1181, 1267, 1317-1320		
	.....	1322		
	PAGESIZE.....	493, 1247, 1293, 1297, 1475, 1971		
	PAGE_SIZE.....	1971		
	PATH.....	671		
	PATH environment variable.....	762		
	pathconf.....	856		
	pathconf().....	<b>1329</b>		
	PATH_MAX.....	475, 547, 626, 641, 643, 646		
	.....	758, 776, 795, 848, 856, 887, 906, 915		
	.....	951, 1038, 1150, 1159, 1207, 1232, 1235		
	.....	1239, 1268, 1279, 1308, 1319, 1324, 1456		
	.....	1677, 1684, 1709, 1717, 1749, 1757, 1824		
	.....	1827, 1899, 1966, 1974, 2040, 2061, 2069		
	.....	2071		
	pause().....	<b>1330</b>		
	pclose().....	<b>1331</b>		
	perror().....	<b>1333</b>		
	persistent connection (I_PLINK).....	<b>1089</b>		
	PF_.....	468		
	physical write.....	909		
	PIO.....	457		
	pipe.....	853, 1322, 2161		
	pipe().....	<b>1335</b>		
	PIPE_BUF.....	856, 2158, 2161		
	PIPE_MAX.....	2163		
	plain characters.....	1919		
	POLL.....	468		
	poll().....	<b>1337</b>		
	POLLERR.....	1337		
	POLLHUP.....	1337		
	POLLIN.....	1337		
	POLLNVAL.....	1338		
	POLLOUT.....	1337		
	POLLPRI.....	1337		
	POLLRDBAND.....	1337		
	POLLRDNORM.....	1337		
	POLLWRBAND.....	1337		
	POLLWRNORM.....	1337		
	POLL_.....	468		
	popen().....	<b>1341</b>		
	portability.....	<b>453</b>		
	POSIX.....	726		
	POSIX.1 symbols.....	463		
	POSIX_.....	465		

864	posix	465	posix_spawn_file_actions_destroy()	1369
865	POSIX_ALLOC_SIZE_MIN	856	posix_spawn_file_actions_init	1369
866	posix_fadvise()	1344	posix_spawn_file_actions_init()	1370
867	POSIX_FADV_DONTNEED	1344	POSIX_SPAWN_RESETEIDS	1356, 1373
868	POSIX_FADV_NOREUSE	1344	POSIX_SPAWN_SETPGROUP	1356, 1373
869	POSIX_FADV_NORMAL	1344	POSIX_SPAWN_SETSCHEDPARAM	1373
870	POSIX_FADV_RANDOM	1344	POSIX_SPAWN_SETSCHEDULER	1356, 1373
871	POSIX_FADV_SEQUENTIAL	1344	POSIX_SPAWN_SETSIGDEF	1373
872	POSIX_FADV_WILLNEED	1344	POSIX_SPAWN_SETSIGMASK	1373
873	posix_fallocate()	1346	POSIX_TRACE_ADD_EVENTSET	1438
874	posix_madvise()	1348	POSIX_TRACE_ALL_EVENTS	1432
875	POSIX_MADV_DONTNEED	1348	POSIX_TRACE_APPEND	1400, 1424
876	POSIX_MADV_NORMAL	1348	posix_trace_attr_destroy()	1393
877	POSIX_MADV_RANDOM	1348	posix_trace_attr_getclockres()	1395
878	POSIX_MADV_SEQUENTIAL	1348	posix_trace_attr_getcreatetime	1395
879	POSIX_MADV_WILLNEED	1348	posix_trace_attr_getcreatetime()	1397
880	posix_memalign()	1352	posix_trace_attr_getgenversion	1395
881	posix_mem_offset()	1350	posix_trace_attr_getgenversion()	1398
882	posix_openpt()	1353	posix_trace_attr_getinherited()	1399
883	POSIX_REC_INCR_XFER_SIZE	856	posix_trace_attr_getlogfullpolicy	1399
884	POSIX_REC_MAX_XFER_SIZE	856	posix_trace_attr_getlogfullpolicy()	1402
885	POSIX_REC_MIN_XFER_SIZE	856	posix_trace_attr_getlogsize()	1403
886	POSIX_REC_XFER_ALIGN	856	posix_trace_attr_getmaxdatasize	1403
887	posix_spawn()	1355	posix_trace_attr_getmaxdatasize()	1406
888	posix_spawnattr_destroy()	1371	posix_trace_attr_getmaxsystemevents	1403
889	posix_spawnattr_getflags()	1373	posix_trace_attr_getmaxsystemevents	1406
890	posix_spawnattr_getpgroup()	1375	posix_trace_attr_getmaxuserevents	1403
891	posix_spawnattr_getschedparam()	1377	posix_trace_attr_getname	1406
892	posix_spawnattr_getschedpolicy()	1379	posix_trace_attr_getname()	1407
893	posix_spawnattr_getsigdefault()	1381	posix_trace_attr_getstreamfullpolicy	1399
894	posix_spawnattr_getsigmask()	1383	posix_trace_attr_getstreamfullpolicy()	1408
895	posix_spawnattr_init	1371	posix_trace_attr_getstreamsize	1403
896	posix_spawnattr_init()	1385	posix_trace_attr_getstreamsize()	1409
897	posix_spawnattr_setflags	1373	posix_trace_attr_init	1393
898	posix_spawnattr_setflags()	1386	posix_trace_attr_init()	1410
899	posix_spawnattr_setpgroup	1375	posix_trace_attr_setinherited	1399
900	posix_spawnattr_setpgroup()	1387	posix_trace_attr_setinherited()	1411
901	posix_spawnattr_setschedparam	1377	posix_trace_attr_setlogfullpolicy	1399
902	posix_spawnattr_setschedparam()	1388	posix_trace_attr_setlogfullpolicy()	1412
903	posix_spawnattr_setschedpolicy	1379	posix_trace_attr_setlogsize	1403
904	posix_spawnattr_setschedpolicy()	1389	posix_trace_attr_setlogsize()	1413
905	posix_spawnattr_setsigdefault	1381	posix_trace_attr_setmaxdatasize	1403
906	posix_spawnattr_setsigdefault()	1390	posix_trace_attr_setmaxdatasize()	1414
907	posix_spawnattr_setsigmask	1383	posix_trace_attr_setname	1395
908	posix_spawnattr_setsigmask()	1391	posix_trace_attr_setname()	1415
909	posix_spawnnp	1355	posix_trace_attr_setstreamfullpolicy	1399
910	posix_spawnnp()	1392	posix_trace_attr_setstreamfullpolicy()	1416
911	posix_spawn_file_actions_addclose()	1363	posix_trace_attr_setstreamsize	1403
912	posix_spawn_file_actions_adddup2()	1366	posix_trace_attr_setstreamsize()	1417
913	posix_spawn_file_actions_addopen	1363	posix_trace_clear()	1418
914	posix_spawn_file_actions_addopen()	1368		

## Index

- 915 posix\_trace\_close() .....**1420**
- 916 POSIX\_TRACE\_CLOSE\_FOR\_CHILD .....1399
- 917 posix\_trace\_create() .....**1422**
- 918 posix\_trace\_create\_withlog .....1422
- 919 POSIX\_TRACE\_ERROR trace event .....527
- 920 posix\_trace\_event() .....**1426**
- 921 posix\_trace\_eventid\_equal() .....**1428**
- 922 posix\_trace\_eventid\_get\_name .....1428
- 923 posix\_trace\_eventid\_get\_name() .....**1430**
- 924 posix\_trace\_eventid\_open .....1426
- 925 posix\_trace\_eventid\_open() .....**1431**
- 926 posix\_trace\_eventset\_add() .....**1432**
- 927 posix\_trace\_eventset\_del .....1432
- 928 posix\_trace\_eventset\_empty .....1432
- 929 posix\_trace\_eventset\_fill .....1432
- 930 posix\_trace\_eventset\_ismember .....1432
- 931 posix\_trace\_eventtypelist\_getnext\_id() .....**1434**
- 932 posix\_trace\_eventtypelist\_rewind .....1434
- 933 posix\_trace\_event\_info structure .....524
- 934 POSIX\_TRACE\_FILTER trace event .....527, 1438
- 935 POSIX\_TRACE\_FLUSH .....1400
- 936 posix\_trace\_flush .....1422
- 937 posix\_trace\_flush() .....**1435**
- 938 POSIX\_TRACE\_FLUSHING .....523
- 939 POSIX\_TRACE\_FULL .....522-524
- 940 posix\_trace\_getnext\_event() .....**1441**
- 941 posix\_trace\_get\_attr() .....**1436**
- 942 posix\_trace\_get\_filter() .....**1438**
- 943 posix\_trace\_get\_status .....1436
- 944 posix\_trace\_get\_status() .....**1440**
- 945 POSIX\_TRACE\_INHERITED .....1399
- 946 POSIX\_TRACE\_LOOP .....522, 1399-1400, 1424
- 947 POSIX\_TRACE\_NOT\_FLUSHING .....523
- 948 POSIX\_TRACE\_NOT\_FULL .....522, 524
- 949 POSIX\_TRACE\_NOT\_FULL .....1418
- 950 POSIX\_TRACE\_NOT\_TRUNCATED .....525, 1442
- 951 POSIX\_TRACE\_NO\_OVERRUN .....523-524, 1436
- 952 posix\_trace\_open .....1420
- 953 posix\_trace\_open() .....**1444**
- 954 POSIX\_TRACE\_OVERFLOW trace event .....527
- 955 POSIX\_TRACE\_OVERRUN .....523-524
- 956 POSIX\_TRACE\_RESUME trace event .....527
- 957 posix\_trace\_rewind .....1420
- 958 posix\_trace\_rewind() .....**1445**
- 959 POSIX\_TRACE\_RUNNING .....522, 1448
- 960 POSIX\_TRACE\_SET\_EVENTSET .....1438
- 961 posix\_trace\_set\_filter .....1438
- 962 posix\_trace\_set\_filter() .....**1446**
- 963 posix\_trace\_shutdown .....1422
- 964 posix\_trace\_shutdown() .....**1447**
- 965 POSIX\_TRACE\_START trace event .....527, 1448
- posix\_trace\_start() .....**1448**
- posix\_trace\_status\_info structure .....521
- posix\_trace\_stop .....1448
- POSIX\_TRACE\_STOP trace event .....527, 1448
- POSIX\_TRACE\_SUB\_EVENTSET .....1438
- POSIX\_TRACE\_SUSPENDED .....522-523, 1448
- POSIX\_TRACE\_SYSTEM\_EVENTS .....1432
- posix\_trace\_timedgetnext\_event .....1441
- posix\_trace\_timedgetnext\_event() .....**1450**
- posix\_trace\_trid\_eventid\_open .....1428
- posix\_trace\_trid\_eventid\_open() .....**1451**
- POSIX\_TRACE\_TRUNCATED\_READ .....525, 1442
- POSIX\_TRACE\_TRUNCATED\_RECORD .....525  
.....1442
- posix\_trace\_trygetnext\_event .....1441
- posix\_trace\_trygetnext\_event() .....**1452**
- POSIX\_TRACE\_UNTIL\_FULL .....523  
.....1399-1400, 1424
- POSIX\_TRACE\_USER\_EVENT\_MAX .....1426
- POSIX\_TRACE\_WOPID\_EVENTS .....1432
- POSIX\_TYPED\_MEM\_ALLOCATE .....1251-1252  
.....1350, 1453, 1455
- POSIX\_TYPED\_MEM\_ALLOCATE\_CONTIG .....  
.....1251-1252, 1350, 1453, 1455
- posix\_typed\_mem\_get\_info() .....**1453**
- POSIX\_TYPED\_MEM\_MAP\_ALLOCATABLE .....297  
.....1455
- posix\_typed\_mem\_open() .....**1455**
- pow() .....**1458**
- powf .....1458
- powl .....1458
- pread .....1666
- pread() .....**1461**
- predefined stream
  - standard error .....487
  - standard input .....487
  - standard output .....487
- preempted thread .....1528
- PRI .....468
- printf .....861
- printf() .....**1462**
- priority .....488
- PRIO .....468
- PRIO\_INHERIT .....1579
- PRIO\_PGRP .....1002
- PRIO\_PROCESS .....1002
- PRIO\_USER .....1002
- process
  - concurrent execution .....853
  - setting real and effective user IDs .....1807
  - single-threaded .....853

966	process creation .....	853	pthread_barrierattr_destroy() .....	1505
967	process group		pthread_barrierattr_getpshared() .....	1507
968	orphaned .....	767	pthread_barrierattr_init .....	1505
969	process group ID.....	998, 1799, 1811	pthread_barrierattr_init().....	1509
970	process ID, 1.....	767	pthread_barrierattr_setpshared .....	1507
971	process lifetime .....	1146	pthread_barrierattr_setpshared().....	1510
972	process scheduling.....	494	pthread_barrier_destroy().....	1500
973	process shared memory .....	1583	pthread_barrier_init.....	1500
974	process synchronization .....	1583	pthread_barrier_init() .....	1502
975	process termination.....	766	PTHREAD_BARRIER_SERIAL_THREAD .....	1503
976	PROT .....	466, 468	pthread_barrier_wait() .....	1503
977	PROT_EXEC.....	1251, 1260	pthread_cancel() .....	1511
978	PROT_NONE.....	494, 1251, 1260	PTHREAD_CANCELED.....	507, 1548
979	PROT_READ.....	1251, 1260	PTHREAD_CANCEL_ASYNCHRONOUS .....	504
980	PROT_WRITE.....	1251, 1253, 1255, 1260	.....	1624
981	PS .....	457	PTHREAD_CANCEL_DEFERRED .....	504
982	pselect().....	1463	.....	1526, 1624
983	pseudo-random sequence generation functions....		PTHREAD_CANCEL_DISABLE.....	504, 1624
984	.....	1663	PTHREAD_CANCEL_ENABLE.....	504, 1624
985	PTHREAD .....	466	pthread_cleanup_pop() .....	1513
986	pthread_atfork().....	1468	pthread_cleanup_push.....	1513
987	pthread_attr_destroy() .....	1470	pthread_condattr_destroy() .....	1532
988	pthread_attr_getdetachstate().....	1473	pthread_condattr_getclock().....	1534
989	pthread_attr_getguardsize().....	1475	pthread_condattr_getpshared().....	1536
990	pthread_attr_getinheritsched().....	1477	pthread_condattr_init.....	1532
991	pthread_attr_getschedparam() .....	1479	pthread_condattr_init() .....	1538
992	pthread_attr_getschedpolicy().....	1481	pthread_condattr_setclock.....	1534
993	pthread_attr_getscope() .....	1483	pthread_condattr_setclock() .....	1539
994	pthread_attr_getstack() .....	1485	pthread_condattr_setpshared.....	1536
995	pthread_attr_getstackaddr() .....	1487	pthread_condattr_setpshared() .....	1540
996	pthread_attr_getstacksize() .....	1489	pthread_cond_broadcast().....	1518
997	pthread_attr_init.....	1470	pthread_cond_destroy().....	1521
998	pthread_attr_init() .....	1490	pthread_cond_init.....	1521
999	pthread_attr_setdetachstate.....	1473	pthread_cond_init().....	1524
1000	pthread_attr_setdetachstate() .....	1491	PTHREAD_COND_INITIALIZER.....	1521, 1524
1001	pthread_attr_setguardsize.....	1475	pthread_cond_signal .....	1518
1002	pthread_attr_setguardsize() .....	1492	pthread_cond_signal().....	1525
1003	pthread_attr_setinheritsched.....	1477	pthread_cond_timedwait().....	1526
1004	pthread_attr_setinheritsched() .....	1493	pthread_cond_wait.....	1526
1005	pthread_attr_setschedparam .....	1479	pthread_cond_wait().....	1531
1006	pthread_attr_setschedparam().....	1494	pthread_create().....	1541
1007	pthread_attr_setschedpolicy.....	1481	PTHREAD_CREATE_DETACHED.....	480, 1473
1008	pthread_attr_setschedpolicy() .....	1495	PTHREAD_CREATE_JOINABLE ..	480, 1473, 1558
1009	pthread_attr_setscope .....	1483	PTHREAD_DESTRUCTOR_ITERATIONS .....	1555
1010	pthread_attr_setscope().....	1496	.....	1560, 1972
1011	pthread_attr_setstack .....	1485	pthread_detach().....	1544
1012	pthread_attr_setstack().....	1497	pthread_equal() .....	1546
1013	pthread_attr_setstackaddr .....	1487	pthread_exit() .....	1547
1014	pthread_attr_setstackaddr() .....	1498	PTHREAD_EXPLICIT_SCHED .....	1477
1015	pthread_attr_setstacksize .....	1489	pthread_getconcurrency() .....	1549
1016	pthread_attr_setstacksize().....	1499	pthread_getcpuclid() .....	1551

## Index

1017	pthread_getschedparam()	1552	pthread_rwlockattr_init()	1621
1018	pthread_getspecific()	1555	pthread_rwlockattr_setpshared	1619
1019	PTHREAD_INHERIT_SCHED	1477	pthread_rwlockattr_setpshared()	1622
1020	pthread_join()	1557	pthread_rwlock_destroy()	1602
1021	PTHREAD_KEYS_MAX	1560, 1972	pthread_rwlock_init	1602
1022	pthread_key_create()	1560	pthread_rwlock_init()	1604
1023	pthread_key_delete()	1564	pthread_rwlock_rdlock()	1605
1024	pthread_kill()	1566	pthread_rwlock_timedrdlock()	1607
1025	pthread_mutexattr_destroy()	1582	pthread_rwlock_timedwrlock()	1609
1026	pthread_mutexattr_getprioceiling()	1587	pthread_rwlock_tryrdlock	1605
1027	pthread_mutexattr_getprotocol()	1589	pthread_rwlock_tryrdlock()	1611
1028	pthread_mutexattr_getpshared()	1591	pthread_rwlock_trywrlock()	1612
1029	pthread_mutexattr_gettype()	1593	pthread_rwlock_unlock()	1614
1030	pthread_mutexattr_init	1582	pthread_rwlock_wrllock	1612
1031	pthread_mutexattr_init()	1595	pthread_rwlock_wrllock()	1616
1032	pthread_mutexattr_setprioceiling	1587	PTHREAD_SCOPE_PROCESS	502-503, 1483
1033	pthread_mutexattr_setprioceiling()	1596	PTHREAD_SCOPE_SYSTEM	502-503, 1483
1034	pthread_mutexattr_setprotocol	1589	pthread_self()	1623
1035	pthread_mutexattr_setprotocol()	1597	pthread_setcancelstate()	1624
1036	pthread_mutexattr_setpshared	1591	pthread_setcanceltype	1624
1037	pthread_mutexattr_setpshared()	1598	pthread_setconcurrency	1549
1038	pthread_mutexattr_settype	1593	pthread_setconcurrency()	1626
1039	pthread_mutexattr_settype()	1599	pthread_setschedparam	1552
1040	PTHREAD_MUTEX_DEFAULT	1575, 1593	pthread_setschedparam()	1627
1041	pthread_mutex_destroy()	1567	pthread_setschedprio()	1628
1042	PTHREAD_MUTEX_ERRORCHECK	1575, 1593	pthread_setspecific	1555
1043	pthread_mutex_getprioceiling()	1572	pthread_setspecific()	1630
1044	pthread_mutex_init	1567	pthread_sigmask()	1631
1045	pthread_mutex_init()	1574	pthread_spin_destroy()	1633
1046	PTHREAD_MUTEX_INITIALIZER	1567, 1574	pthread_spin_init	1633
1047	pthread_mutex_lock()	1575	pthread_spin_init()	1635
1048	PTHREAD_MUTEX_NORMAL	1575, 1593	pthread_spin_lock()	1636
1049	PTHREAD_MUTEX_RECURSIVE	1575, 1593-1594	pthread_spin_trylock	1636
1050	pthread_mutex_setprioceiling	1572	pthread_spin_trylock()	1638
1051	pthread_mutex_setprioceiling()	1578	pthread_spin_unlock()	1639
1052	pthread_mutex_timedlock()	1579	PTHREAD_STACK_MIN	1485, 1487, 1489, 1972
1053	pthread_mutex_trylock	1575	pthread_testcancel	1624
1054	pthread_mutex_trylock()	1581	pthread_testcancel()	1640
1055	pthread_mutex_unlock	1575, 1581	PTHREAD_THREADS_MAX	1541, 1972
1056	pthread_once()	1600	ptsname()	1641
1057	PTHREAD_ONCE_INIT	1600	putc()	1642
1058	PTHREAD_PRIO_INHERIT	1589	putchar()	1644
1059	PTHREAD_PRIO_NONE	1589	putchar_unlocked	944
1060	PTHREAD_PRIO_PROTECT	1576, 1589	putchar_unlocked()	1645
1061	PTHREAD_PROCESS_PRIVATE	1507, 1536	putc_unlocked	944
1062	.....	1583, 1591, 1619, 1633	putc_unlocked()	1643
1063	PTHREAD_PROCESS_SHARED	1507, 1536	putenv()	1646
1064	.....	1583, 1591, 1619, 1633	putmsg()	1648
1065	pthread_rwlockattr_destroy()	1617	putpmsg	1648
1066	pthread_rwlockattr_getpshared()	1619	putpmsg()	1652
1067	pthread_rwlockattr_init	1617	puts()	1653

1068	pututxline.....	742	REG_EBRACK.....	1696
1069	pututxline().....	<b>1655</b>	REG_ECOLLATE.....	1696
1070	putwc().....	<b>1656</b>	REG_ECTYPE.....	1696
1071	putwchar().....	<b>1657</b>	REG_EESCAPE.....	1696
1072	pwrite.....	2157	REG_EPAREN.....	1696
1073	pwrite().....	<b>1658</b>	REG_ERANGE.....	1696
1074	P_ALL.....	2094	REG_ESPACE.....	1696
1075	P_PGID.....	2094	REG_ESUBREG.....	1696
1076	P_PID.....	2094	REG_EXTENDED.....	1694
1077	qsort().....	<b>1659</b>	REG_ICASE.....	1694
1078	queue a signal to a process.....	1865	REG_NEWLINE.....	1694
1079	raise().....	<b>1660</b>	REG_NOMATCH.....	1696
1080	rand().....	<b>1662</b>	REG_NOSUB.....	1694
1081	random.....	1075	REG_NOTBOL.....	1695
1082	random().....	<b>1665</b>	REG_NOTEOL.....	1695
1083	RAND_MAX.....	1662	remainder().....	<b>1701</b>
1084	rand_r.....	1662	remainderf.....	1701
1085	read from a file.....	1669	remainderl.....	1701
1086	read().....	<b>1666</b>	remove a directory.....	1718
1087	readdir().....	<b>1673</b>	remove directory entries.....	2063
1088	readdir_r.....	1673	remove().....	<b>1703</b>
1089	readlink().....	<b>1677</b>	remque.....	1077
1090	readv().....	<b>1680</b>	remque().....	<b>1705</b>
1091	real user ID.....	548, 1145	remquo().....	<b>1706</b>
1092	realloc().....	<b>1682</b>	remquof.....	1706
1093	realpath().....	<b>1684</b>	remquol.....	1706
1094	REALTIME.....	556, 558-559, 561, 564-565	rename a file.....	1710
1095	.....	567, 653, 659, 794, 1162, 1247, 1249	rename().....	<b>1708</b>
1096	.....	1262-1263, 1265, 1267, 1270, 1273, 1275	rewind().....	<b>1712</b>
1097	.....	1279, 1300, 1728-1732, 1735, 1743-1746	rewinddir().....	<b>1713</b>
1098	.....	1748, 1751, 1755, 1757, 1759, 1823, 1827	RE_DUP_MAX.....	1972
1099	.....	1864, 1872, 1878, 2019, 2022-2023	rindex().....	<b>1714</b>
1100	REALTIME THREADS.....	1477, 1481, 1483, 1493	rint().....	<b>1715</b>
1101	.....	1495-1496, 1552, 1572, 1578, 1587, 1589	RLIMIT_.....	468
1102	.....	1596-1597, 1627	RLIMIT_AS.....	1016
1103	recv().....	<b>1686</b>	RLIMIT_CORE.....	1015
1104	recvfrom().....	<b>1688</b>	RLIMIT_CPU.....	1015
1105	recvmsg().....	<b>1691</b>	RLIMIT_DATA.....	1015
1106	regcomp().....	<b>1694</b>	RLIMIT_FSIZE.....	1015
1107	regerror.....	1694	RLIMIT_NOFILE.....	1015, 1017
1108	regexec.....	1694	RLIMIT_STACK.....	1015
1109	regfree.....	1694	RLIM.....	468
1110	register fork handlers.....	1468	RLIM_INFINITY.....	1015-1016
1111	REG_.....	468	RLIM_SAVED_CUR.....	1016
1112	REG_constants		RLIM_SAVED_MAX.....	1016
1113	error return values of regcomp.....	1696	rmdir().....	<b>1717</b>
1114	used in regcomp.....	1694	RMSGD.....	1083
1115	REG_BADBR.....	1696	RMSGN.....	1083
1116	REG_BADPAT.....	1696	RNORM.....	1083
1117	REG_BADRPT.....	1696	round robin.....	496
1118	REG_EBRACE.....	1696	round().....	<b>1720</b>

## Index

1119	roundf .....	1720	.....	1733, 1751
1120	roundl.....	1720	sched_rr_get_interval() .....	<b>1731</b>
1121	routing .....	509	sched_setparam().....	<b>1732</b>
1122	RPROTDAT .....	1083	sched_setscheduler().....	<b>1735</b>
1123	RPROTDIS .....	1083	SCHED_SPORADIC.....	492, 495, 497, 1605, 1733, 1751
1124	RPROTNORM.....	1083	sched_yield().....	<b>1738</b>
1125	RS .....	<b>457</b>	SCM .....	468
1126	RS_HIPRI.....	982, 1082, 1648	SCN .....	468
1127	RTLD_GLOBAL .....	712, 716-717, 720	SD.....	<b>457</b>
1128	RTLD_LAZY .....	716, 719	security considerations ..	647, 766, 1145, 1799, 1901
1129	RTLD_LOCAL .....	717	seed48.....	721
1130	RTLD_NEXT .....	719-720	seed48() .....	<b>1739</b>
1131	RTLD_NOW .....	716-717	seekdir() .....	<b>1740</b>
1132	RTS.....	<b>457</b>	SEEK_CUR .....	787, 899, 1205
1133	RTSIG_MAX.....	1972	SEEK_END .....	787, 899, 1205
1134	RUSAGE_.....	468	SEEK_GET .....	1712
1135	RUSAGE_CHILDREN .....	1018	SEEK_SET .....	492, 561, 567, 787, 899, 1205
1136	RUSAGE_SELF .....	1018	SEGV .....	468
1137	SA.....	468	select .....	1463
1138	SA_NOCLDSTOP.....	481, 1838-1839, 1842	select() .....	<b>1742</b>
1139	SA_NOCLDWAIT .....	764-765, 1018, 1840, 2088	SEM.....	<b>457</b>
1140	SA_NODEFER .....	1840	semctl().....	<b>1760</b>
1141	SA_ONSTACK.....	755, 1839	semget() .....	<b>1763</b>
1142	SA_RESETHAND .....	601, 1839-1840	semid .....	490
1143	SA_RESTART .....	601, 1466, 1839, 1854	semop() .....	<b>1766</b>
1144	SA_SIGINFO.....	1838-1839, 1842, 1864	SEM .....	466, 468
1145	scalb() .....	<b>1722</b>	sem_close().....	<b>1743</b>
1146	scalbln().....	<b>1724</b>	sem_destroy() .....	<b>1744</b>
1147	scalblnf.....	1724	SEM_FAILED .....	1749
1148	scalblnl.....	1724	sem_getvalue() .....	<b>1745</b>
1149	scalbn .....	1724	sem_init().....	<b>1746</b>
1150	scalbn().....	<b>1726</b>	SEM_NSEMS_MAX.....	1746, 1972
1151	scalbnf.....	1724, 1726	sem_open().....	<b>1748</b>
1152	scalbnl.....	1724, 1726	sem_perm.....	490
1153	scanf.....	892	sem_post().....	<b>1751</b>
1154	scanf() .....	<b>1727</b>	sem_timedwait() .....	<b>1753</b>
1155	schedule alarm .....	570	sem_trywait() .....	<b>1755</b>
1156	scheduling documentation.....	504	SEM_UNDO .....	1766
1157	scheduling policy		sem_unlink().....	<b>1757</b>
1158	round robin.....	496	SEM_VALUE_MAX.....	1748, 1972
1159	SCHED_.....	466	.....	1746
1160	SCHED_FIFO .....	492, 495, 503, 756, 851	sem_wait .....	1755
1161	.....	1002, 1310, 1479, 1481, 1552, 1587	sem_wait().....	<b>1759</b>
1162	.....	1605, 1733, 1751	send().....	<b>1771</b>
1163	sched_getparam() .....	<b>1729</b>	sendmsg() .....	<b>1773</b>
1164	sched_getscheduler() .....	<b>1730</b>	sendto() .....	<b>1776</b>
1165	sched_get_priority_max().....	<b>1728</b>	service name .....	883
1166	sched_get_priority_min.....	1728	session.....	767, 1145, 1799, 1811
1167	SCHED_OTHER .....	495, 498, 1481, 1552, 1733	set cancelability state .....	1624
1168	SCHED_RR .....	492, 495-496, 503, 756, 851	set file creation mask .....	2054
1169	.....	1002, 1310, 1479, 1481, 1552, 1605	set process group ID for job control .....	1799

1170	set-group-ID	645, 761, 766, 791	shell scripts	
1171	set-user-ID	761, 766, 951, 1145	exec	760
1172	SETALL	1760, 1763	shell, login	760
1173	setbuf()	1779	SHM	457, 468
1174	setcontext	948	shmat()	1829
1175	setcontext()	1780	shmctl()	1831
1176	setegid()	1781	shmdt()	1833
1177	setenv()	1782	shmget()	1834
1178	seteuid()	1784	shm_id	490
1179	setgid()	1785	SHMLBA	1829
1180	setgrent	730	SHM_	468
1181	setgrent()	1787	shm_open()	1823
1182	sethostent	732	shm_perm	490
1183	sethostent()	1788	SHM_RDONLY	1829
1184	setitimer	977	SHM_RND	1829
1185	setitimer()	1789	shm_unlink()	1827
1186	setjmp()	1790	should	452
1187	setkey()	1792	shutdown()	1836
1188	setlocale()	1793	SHUT_	468
1189	setlogmask	666	SIGABRT	543
1190	setlogmask()	1797	sigaction()	1838
1191	setnetent	734	sigaddset()	1844
1192	setnetent()	1798	SIGALRM	570, 977, 1885, 2050, 2067
1193	setpgid()	1799	sigaltstack()	1845
1194	setpgrp()	1801	SIGBUS	494, 1253, 1256, 1631
1195	setpriority	1002	SIGCANCEL	1511
1196	setpriority()	1802	SIGCHLD	667, 764-765, 1018, 1045, 1839
1197	setprotoent	736	.....	1842, 1851, 1977, 2088, 2094
1198	setprotoent()	1803	SIGCLD	1842
1199	setpwent	738	SIGCONT	484, 765, 767, 1144-1145
1200	setpwent()	1804	sigdelset()	1847
1201	setregid()	1805	sigemptyset()	1848
1202	setreuid()	1807	SIGEV_	466
1203	setrlimit	1015	SIGEV_NONE	479, 492
1204	setrlimit()	1809	SIGEV_SIGNAL	479, 2019
1205	setservent	740	SIGEV_THREAD	479-480
1206	setservent()	1810	sigfillset()	1850
1207	setsid()	1811	SIGFPE	1631, 1858
1208	setsockopt()	1813	sighold()	1851
1209	setstate	1075	SIGHUP	661, 765, 767
1210	setstate()	1816	sigignore	1851
1211	setuid()	1817	sigignore()	1853
1212	setutxent	742	SIGILL	1631, 1858
1213	setutxent()	1820	SIGINT	853, 1977
1214	SETVAL	1760, 1763	siginterrupt()	1854
1215	setvbuf()	1821	sigismember()	1856
1216	shall	452	SIGKILL	1145, 1838, 1842, 1851
1217	shell	760, 767, 980, 998, 1145, 1800, 2091	siglongjmp()	1857
1218	job	1145	signal generation and delivery	478
1219	login	980	realtime	479
1220			signal handler	1858

## Index

1221	signal()	1858	sinhf	1882
1222	signaling a condition	1519	sinhl	1882
1223	signals	478	sinl	1879
1224	signbit()	1860	sinl()	1884
1225	sigpause	1851	SIO	457
1226	sigpause()	1861	SI_	468
1227	sigpending()	1862	SI_ASYNCIO	482
1228	SIGPIPE	784, 817, 873, 877, 900, 903, 1649, 2160	SI_MESGQ	482
1229	SIGPOLL	661, 1081-1082	SI_QUEUE	482
1230	sigprocmask	1631	SI_TIMER	482
1231	sigprocmask()	1863	SI_USER	482
1232	SIGPROF	977	sleep()	1885
1233	sigqueue()	1864	SND	468
1234	SIGQUEUE_MAX	1864, 1972	SNDZERO	1085
1235	SIGQUIT	1977	snprintf	861
1236	sigrelse	1851	snprintf()	1887
1237	sigrelse()	1866	SO	468
1238	SIGRTMAX	479-480, 1864, 1872, 1876	socket I/O mode	510
1239	SIGRTMIN	479-480, 1864, 1872, 1876	socket out-of-band data	511
1240	SIGSEGV	494, 1016, 1297, 1631, 1858	socket owner	510
1241	sigset	1851	socket queue limits	510
1242	sigset()	1867	socket receive queue	511
1243	sigsetjmp()	1868	socket types	509
1244	SIGSTKSZ	1845	socket()	1888
1245	SIGSTOP	479, 1838, 1842, 1851	socketpair()	1890
1246	sigsuspend()	1870	sockets	508
1247	sigtimedwait()	1872	addressing	509
1248	SIGTSTP	479	asynchronous errors	512
1249	SIGTTIN	479, 821, 827, 1668	connection indication queue	512
1250	SIGTTOU	479, 784, 817, 873, 877, 900	Internet Protocols	516
1251		902, 1988, 1990, 1992, 1999, 2002, 2160	IPv4	517
1252	SIGURG	1082	IPv6	517
1253	SIGVTALRM	977	local UNIX connections	516
1254	sigwait()	1876	options	513
1255	sigwaitinfo	1872	pending error	510
1256	sigwaitinfo()	1878	signals	512
1257	SIGXCPU	1015	SOCK_	468
1258	SIGXFSZ	1015, 2040	SOCK_DGRAM	516, 1888, 1890
1259	SIG_	466, 468	SOCK_RAW	516
1260	SIG_BLOCK	1631	SOCK_SEQPACKET	516, 1888, 1890
1261	SIG_DFL	480, 755, 1016, 1838, 1840, 1858	SOCK_STREAM	516, 1888, 1890
1262	SIG_ERR	601, 1858	SPI	458
1263	SIG_HOLD	1851	SPN	458
1264	SIG_IGN	480-481, 755, 761, 764-765	sporadic server policy	
1265		1018, 1838, 1858, 2088	execution capacity	497
1266	SIG_SETMASK	1631	replenishment period	497
1267	SIG_UNBLOCK	1631	sprintf	861
1268	sin()	1879	sprintf()	1892
1269	sinh	1879	spurious wakeup	1519
1270	sinh()	1881	sqrt()	1893
1271	sinh()	1882	sqrtf	1893

1272	sqrtl.....	1893	streams	
1273	srand.....	1662	stream orientation.....	486
1274	srand().....	<b>1895</b>	STREAM_MAX.....	799, 848, 1341, 1972
1275	srand48.....	721	strerror().....	<b>1917</b>
1276	srand48().....	<b>1896</b>	strerror_r.....	1917
1277	srandom.....	1075	strfmon().....	<b>1919</b>
1278	srandom().....	<b>1897</b>	strftime().....	<b>1924</b>
1279	SS.....	<b>458</b>	strlen().....	<b>1929</b>
1280	sscanf.....	892	strncasecmp.....	1906
1281	sscanf().....	<b>1898</b>	strncasecmp().....	<b>1931</b>
1282	SSIZE_MAX.....	1270, 1286, 1666, 1677, 1920, 2157	strncat().....	<b>1932</b>
1283	SS.....	468	strncmp().....	<b>1933</b>
1284	SS_DISABLE.....	1845-1846	strncpy().....	<b>1934</b>
1285	SS_ONSTACK.....	1845	strpbrk().....	<b>1935</b>
1286	stack size.....	1470	strptime().....	<b>1936</b>
1287	stat().....	<b>1899</b>	strchr().....	<b>1940</b>
1288	statvfs.....	906	strspn().....	<b>1941</b>
1289	statvfs().....	<b>1903</b>	strstr().....	<b>1942</b>
1290	stderr.....	1904	strtod().....	<b>1943</b>
1291	STDERR_FILENO.....	1904	strtof.....	1943
1292	stdin.....	<b>1904</b>	strtof().....	<b>1947</b>
1293	STDIN_FILENO.....	1341, 1904	strtoimax().....	<b>1948</b>
1294	stdio locking functions.....	832	strtok().....	<b>1949</b>
1295	stdio with explicit client locking.....	944	strtok_r.....	1949
1296	stdout.....	1904	strtol().....	<b>1952</b>
1297	STDOUT_FILENO.....	1341, 1904	strtold.....	1943
1298	STR.....	468	strtold().....	<b>1954</b>
1299	strcasecmp().....	<b>1906</b>	strtoll.....	1952
1300	strcat().....	<b>1907</b>	strtoll().....	<b>1955</b>
1301	strchr().....	<b>1908</b>	strtoul().....	<b>1956</b>
1302	strcmp().....	<b>1909</b>	strtoull.....	1956
1303	strcoll().....	<b>1911</b>	strtoumax.....	1948
1304	strcpy().....	<b>1913</b>	strtoumax().....	<b>1959</b>
1305	strcspn().....	<b>1915</b>	strxfrm().....	<b>1960</b>
1306	strdup().....	<b>1916</b>	ST_NOSUID.....	755, 906
1307	STREAM.....	1084, 1086, 1648, 1652, 1667, 2159	ST_RDONLY.....	906
1308	stream		superuser.....	548, 647, 1161, 2063
1309	byte-oriented.....	486	supplementary groups.....	647, 969
1310	wide-oriented.....	486	SVID.....	1868
1311	STREAM head/tail.....	<b>488</b>	SVR4.....	1255, 1300
1312	stream-full-policy attribute.....	522-523, 526, 1400	SV.....	468
1313	stream-min-size attribute.....	526, 1404	swab().....	<b>1962</b>
1314	streams.....	484	swapcontext.....	1209
1315	STREAMS.....	472, 661, 776, 795, 982, 1080	swapcontext().....	<b>1963</b>
1316	.....	1095, 1319-1320, 1337, 1463	swprintf.....	925
1317	access.....	489	swprintf().....	<b>1964</b>
1318	streams		swscanf.....	934
1319	interaction with file descriptors.....	485	swscanf().....	<b>1965</b>
1320	STREAMS		symbols	
1321	multiplexed.....	1088	POSIX.1.....	463
1322	overview.....	488	symlink().....	<b>1966</b>

## Index

- 1323 SYMLINK\_MAX .....856, 1966
- 1324 SYMLOOP\_MAX .....600, 676, 777, 795, 848
- 1325 .....888, 907, 915, 921, 1150, 1239, 1308
- 1326 .....1684, 1775, 1778, 1972, 2041, 2071
- 1327 sync() .....**1968**
- 1328 synchronously accept a signal .....1873
- 1329 sysconf() .....**1969**
- 1330 syslog .....666
- 1331 syslog() .....**1976**
- 1332 system crash .....909
- 1333 System III .....647, 2056
- 1334 system interfaces .....533
- 1335 system name .....2056
- 1336 system trace event type definitions .....526
- 1337 System V .....570, 647, 762, 767, 789-790
- 1338 .....858, 998, 1145, 1233, 1718, 1811, 1842
- 1339 .....1868, 1994, 2056
- 1340 system() .....**1977**
- 1341 S\_ .....468
- 1342 S\_BANDURG .....1082
- 1343 S\_ERROR .....1081
- 1344 S\_HANGUP .....1082
- 1345 S\_HIPRI .....1081
- 1346 S\_IFBLK .....1238
- 1347 S\_IFCHR .....1238
- 1348 S\_IFDIR .....1238
- 1349 S\_IFIFO .....1238
- 1350 S\_IFREG .....1238
- 1351 S\_INPUT .....1081
- 1352 S\_IRGRP .....780, 904, 1238
- 1353 S\_IROTH .....780, 904, 1238
- 1354 S\_IRUSR .....780, 904, 1238
- 1355 S\_IRWXG .....1238
- 1356 S\_IRWXO .....1238
- 1357 S\_IRWXU .....1238
- 1358 S\_ISGID .....643-645, 1238, 2040, 2158
- 1359 S\_ISUID .....643-644, 1238, 2040, 2158
- 1360 S\_ISVTX .....643, 1238, 1709, 1718, 2061
- 1361 S\_IWGRP .....780, 904, 1238
- 1362 S\_IWOTH .....780, 904, 1238
- 1363 S\_IWUSR .....780, 904, 1238
- 1364 S\_IXGRP .....1238
- 1365 S\_IXOTH .....1238
- 1366 S\_IXUSR .....1238
- 1367 S\_MSG .....1081
- 1368 S\_OUTPUT .....1081
- 1369 S\_RDBAND .....1081-1082
- 1370 S\_RDNORM .....1081
- 1371 S\_WRBAND .....1081
- 1372 S\_WRNORM .....1081
- 1373 TABSIZE .....603, 1203
- tan() .....**1982**
- tanf .....1982
- tanf() .....**1984**
- tanh() .....**1985**
- tanhf .....1985
- tanhL .....1985
- tanl .....1982
- tanl() .....**1987**
- tcdrain() .....**1988**
- tcflow() .....**1990**
- tcflush() .....**1992**
- tcgetattr() .....**1994**
- tcgetpgrp() .....**1996**
- tcgetsid() .....**1998**
- TCIFLUSH .....1992
- TCIOFF .....1990
- TCIOFLUSH .....1992
- TCION .....1990
- TCOFLUSH .....1992
- TCOOFF .....1990
- TCOON .....1990
- TCP\_ .....468
- TCSADRAIN .....2001
- TCSAFLUSH .....2001
- TCSANOW .....2001
- tcsendbreak() .....**1999**
- tcsetattr() .....**2001**
- tcsetpgrp() .....**2004**
- TCT .....**458**
- tdelete() .....**2006**
- TEF .....**458**
- telldir() .....**2010**
- tempnam() .....**2011**
- terminal access control .....1994, 2002
- terminal device name .....2044
- terminate a process .....766
- terminology .....**451**
- termios structure .....1994
- tfind .....2006
- tfind() .....**2013**
- tgamma() .....**2014**
- tgammaf .....2014
- tgammaL .....2014
- THR .....**458**
- thread cancelation
  - cleanup handlers .....507
  - creation .....1542
  - creation attributes .....1470
  - thread ID .....1546
  - thread IDs .....501
  - thread mutexes .....501

1374	thread scheduling.....	502	TRC.....	459
1375	thread termination.....	1547	TRI.....	459
1376	thread-safety.....	500, 832	TRL.....	459
1377	thread-specific data key creation.....	1562	trunc().....	2039
1378	thread-specific data key deletion.....	1564	truncate().....	2040
1379	thread-specific data management.....	1556	truncation-status attribute.....	1426
1380	threads.....	500	truncf.....	2039
1381	regular file operations.....	508	truncf().....	2042
1382	time().....	2016	truncl.....	2039, 2042
1383	timer ID.....	2021	TSA.....	459
1384	TIMER_.....	467-468	tsearch.....	2006
1385	TIMER_ABSTIME.....	499, 656, 2023	tsearch().....	2043
1386	timer_create().....	2019	TSF.....	460
1387	timer_delete().....	2022	TSH.....	460
1388	timer_getoverrun().....	2023	TSP.....	460
1389	timer_gettime.....	2023	TSS.....	460
1390	TIMER_MAX.....	1972	ttynam().....	2044
1391	timer_settime.....	2023	ttynam_r.....	2044
1392	times().....	2026	TTY_NAME_MAX.....	1972, 2044
1393	timezone().....	2028	twalk.....	2006
1394	TMO.....	458	twalk().....	2046
1395	tmpfile().....	2029	TYM.....	460
1396	tmpnam().....	2031	tzname.....	2047-2048
1397	TMP_MAX.....	2011, 2030-2031	TZNAME_MAX.....	1972
1398	TMR.....	459	tzset.....	2048
1399	toascii().....	2033	tzset().....	2048
1400	tolower().....	2034	t_uscalar_t.....	1083
1401	TOSTOP.....	784, 817, 873, 877, 900, 902, 2159	ualarm().....	2050
1402	toupper().....	2035	UINT.....	468
1403	towctrans().....	2036	UINT_MAX.....	570, 1886
1404	towlower().....	2037	ulimit().....	2052
1405	towupper().....	2038	ULLONG_MAX.....	1957
1406	TPI.....	459	ULONG_MAX.....	1957, 2132
1407	TPP.....	459	UL_GETFSIZE.....	2052
1408	TPS.....	459	UL_SETFSIZE.....	2052
1409	trace event, POSIX_TRACE_ERROR.....	527	umask().....	2054
1410	trace event, POSIX_TRACE_FILTER.....	527, 1438	uname().....	2056
1411	trace event, POSIX_TRACE_OVERFLOW.....	527	undefined.....	452
1412	trace event, POSIX_TRACE_RESUME.....	527	underlying function.....	486
1413	trace event, POSIX_TRACE_START.....	527, 1448	ungetc().....	2058
1414	trace event, POSIX_TRACE_STOP.....	527, 1448	ungetwc().....	2059
1415	trace functions.....	529	unicast.....	517
1416	trace-name attribute.....	526, 1395	unlink().....	2061
1417	TRACE_EVENT_NAME_MAX.....	1426, 1428	unlockpt().....	2065
1418	TRACE_SYS_MAX.....	1423	unsetenv().....	2066
1419	TRACE_USER_EVENT_MAX.....	1426, 1428	unspecified.....	452
1420	TRACING.....	1393, 1395, 1397-1399	UP.....	460
1421	.....	1402-1403, 1406-1412, 1414-1418, 1420	US-ASCII.....	1094
1422	.....	1422, 1426, 1428, 1430-1432, 1434-1436	user ID	
1423	.....	1438, 1440-1441, 1444-1448, 1450-1452	real and effective.....	1807
1424	TRAP_.....	468	setting real and effective.....	1807

## Index

1425	user trace event type definitions.....	529	wscpy().....	2103
1426	USER_PROCESS.....	742-743	wscspn().....	2104
1427	usleep().....	2067	wcsftime().....	2105
1428	UTC.....	2048	wcslen().....	2107
1429	utime().....	2069	wcsncat().....	2108
1430	utimes().....	2071	wcsncmp().....	2109
1431	va_arg().....	2073	wcsncpy().....	2110
1432	va_copy.....	2073	wcspbrk().....	2111
1433	va_end.....	2073	wcsrchr().....	2112
1434	va_start.....	2073	wcsrtombs().....	2113
1435	Version 7.....	570, 647, 1145, 2056	wcsspn().....	2115
1436	vfork().....	2074	wcsstr().....	2116
1437	vfprintf().....	2076	wcstod().....	2117
1438	vfscanf().....	2077	wcstof.....	2117
1439	vwprintf().....	2078	wcstof().....	2120
1440	vwscanf().....	2079	wcstoimax().....	2121
1441	VISIT.....	2006, 2046	wcstok().....	2122
1442	vprintf.....	2076	wcstol().....	2124
1443	vprintf().....	2080	wcstold.....	2117
1444	vscanf.....	2077	wcstold().....	2127
1445	vscanf().....	2081	wcstoll.....	2124
1446	vsnprintf.....	2076	wcstoll().....	2128
1447	vsnprintf().....	2082	wcstombs().....	2129
1448	vsprintf.....	2076, 2082	wcstoul().....	2131
1449	vsscanf.....	2077	wcstoull().....	2134
1450	vsscanf().....	2083	wcstoumax.....	2121
1451	vswprintf.....	2078	wcstoumax().....	2135
1452	vswprintf().....	2084	wcswcs().....	2136
1453	vswscanf.....	2079	wcswidth().....	2137
1454	vswscanf().....	2085	wcsxfrm().....	2138
1455	vwprintf.....	2078	wctob().....	2140
1456	vwprintf().....	2086	wctomb().....	2141
1457	vwscanf.....	2079	wctrans().....	2143
1458	vwscanf().....	2087	wctype().....	2144
1459	wait for process termination.....	2091	wcwidth().....	2146
1460	wait for thread termination.....	1558	WEOF.....	532, 1117, 1119, 1122, 1124 .....1126-1127, 1129, 1131, 1133, 1135 .....1137, 1139, 2037-2038, 2059
1461	wait().....	2088	WEXITED.....	2094
1462	waitid().....	2094	WEXITSTATUS.....	2089
1463	waiting on a condition.....	1527	wide-oriented stream.....	486
1464	waitpid.....	2088	WIFCONTINUED.....	2089
1465	waitpid().....	2096	WIFEXITED.....	2089
1466	WARNING.....	843	WIFSIGNALED.....	2089
1467	warning		WIFSTOPPED.....	2089, 2092
1468	OB.....	456	wmemchr().....	2147
1469	OF.....	456	wmemcmp().....	2148
1470	WCONTINUED.....	2088, 2094	wmemcpy().....	2149
1471	wcrtomb().....	2097	wmemmove().....	2150
1472	wscat().....	2099	wmemset().....	2151
1473	wcschr().....	2100	WNOHANG.....	1842, 2088, 2094
1474	wscmp().....	2101		
1475	wscoll().....	2102		

1476	WNOWAIT.....	2094
1477	wordexp().....	<b>2152</b>
1478	wordfree.....	2152
1479	wprintf.....	925
1480	wprintf().....	<b>2156</b>
1481	WRDE_.....	468
1482	WRDE_APPEND.....	2153
1483	WRDE_BADCHAR.....	2154
1484	WRDE_BADVAL.....	2154
1485	WRDE_CMDSUB.....	2154
1486	WRDE_DOOFFS.....	2153
1487	WRDE_NOCMD.....	2153
1488	WRDE_NOSPACE.....	2154
1489	WRDE_REUSE.....	2153
1490	WRDE_SHOWERR.....	2153
1491	WRDE_SYNTAX.....	2154
1492	WRDE_UNDEF.....	2153
1493	write to a file.....	2161
1494	write().....	<b>2157</b>
1495	writew().....	<b>2165</b>
1496	wscanf.....	934
1497	wscanf().....	<b>2167</b>
1498	WSTOPPED.....	2094
1499	WSTOPSIG.....	2089
1500	WTERMSIG.....	2089
1501	WUNTRACED.....	2088, 2091
1502	XSI.....	<b>460</b>
1503	XSI interprocess communication.....	<b>489</b>
1504	XSR.....	<b>461</b>
1505	X_OK.....	548
1506	y0().....	<b>2168</b>
1507	y1.....	2168
1508	yn.....	2168
1509	zombie process.....	764

# Introduction

## 1.1 Scope

The scope of IEEE Std 1003.1-200x is described in the Base Definitions volume of IEEE Std 1003.1-200x.

## 1.2 Conformance

Conformance requirements for IEEE Std 1003.1-200x are defined in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 2, Conformance.

## 1.3 Normative References

Normative references for IEEE Std 1003.1-200x are defined in the Base Definitions volume of IEEE Std 1003.1-200x.

## 1.4 Change History

Change history is described in the Rationale (Informative) volume of IEEE Std 1003.1-200x, and in the CHANGE HISTORY section of reference pages.

## 1.5 Terminology

This section appears in the Base Definitions volume of IEEE Std 1003.1-200x, but is repeated here for convenience:

For the purposes of IEEE Std 1003.1-200x, the following terminology definitions apply:

### **can**

Describes a permissible optional feature or behavior available to the user or application. The feature or behavior is mandatory for an implementation that conforms to IEEE Std 1003.1-200x. An application can rely on the existence of the feature or behavior.

### **implementation-defined**

Describes a value or behavior that is not defined by IEEE Std 1003.1-200x but is selected by an implementor. The value or behavior may vary among implementations that conform to IEEE Std 1003.1-200x. An application should not rely on the existence of the value or behavior. An application that relies on such a value or behavior cannot be assured to be portable across conforming implementations.

The implementor shall document such a value or behavior so that it can be used correctly by an application.

### **legacy**

Describes a feature or behavior that is being retained for compatibility with older applications, but which has limitations which make it inappropriate for developing portable

33 applications. New applications should use alternative means of obtaining equivalent  
34 functionality.

35 **may**

36 Describes a feature or behavior that is optional for an implementation that conforms to  
37 IEEE Std 1003.1-200x. An application should not rely on the existence of the feature or  
38 behavior. An application that relies on such a feature or behavior cannot be assured to be  
39 portable across conforming implementations.

40 To avoid ambiguity, the opposite of *may* is expressed as *need not*, instead of *may not*.

41 **shall**

42 For an implementation that conforms to IEEE Std 1003.1-200x, describes a feature or  
43 behavior that is mandatory. An application can rely on the existence of the feature or  
44 behavior.

45 For an application or user, describes a behavior that is mandatory.

46 **should**

47 For an implementation that conforms to IEEE Std 1003.1-200x, describes a feature or  
48 behavior that is recommended but not mandatory. An application should not rely on the  
49 existence of the feature or behavior. An application that relies on such a feature or behavior  
50 cannot be assured to be portable across conforming implementations.

51 For an application, describes a feature or behavior that is recommended programming  
52 practice for optimum portability.

53 **undefined**

54 Describes the nature of a value or behavior not defined by IEEE Std 1003.1-200x which  
55 results from use of an invalid program construct or invalid data input.

56 The value or behavior may vary among implementations that conform to  
57 IEEE Std 1003.1-200x. An application should not rely on the existence or validity of the  
58 value or behavior. An application that relies on any particular value or behavior cannot be  
59 assured to be portable across conforming implementations.

60 **unspecified**

61 Describes the nature of a value or behavior not specified by IEEE Std 1003.1-200x which  
62 results from use of a valid program construct or valid data input.

63 The value or behavior may vary among implementations that conform to  
64 IEEE Std 1003.1-200x. An application should not rely on the existence or validity of the  
65 value or behavior. An application that relies on any particular value or behavior cannot be  
66 assured to be portable across conforming implementations.

## 67 1.6 Definitions

68 Concepts and definitions are defined in the Base Definitions volume of IEEE Std 1003.1-200x.

## 69 1.7 Relationship to Other Documents

### 70 1.7.1 System Interfaces

71 This subsection describes some of the features provided by the System Interfaces volume of  
 72 IEEE Std 1003.1-200x that are assumed to be globally available by all systems conforming to this  
 73 volume of IEEE Std 1003.1-200x. This subsection does not attempt to detail all of the features  
 74 defined in the System Interfaces volume of IEEE Std 1003.1-200x that are required by all of the  
 75 utilities defined in this volume of IEEE Std 1003.1-200x; the utility and function descriptions  
 76 point out additional functionality required to provide the corresponding specific features  
 77 needed by each.

78 The following subsections describe frequently used concepts. Many of these concepts are  
 79 described in the Base Definitions volume of IEEE Std 1003.1-200x. Utility and function  
 80 description statements override these defaults when appropriate.

#### 81 1.7.1.1 Process Attributes

82 The following process attributes, as described in the System Interfaces volume of  
 83 IEEE Std 1003.1-200x, are assumed to be supported for all processes in this volume of  
 84 IEEE Std 1003.1-200x:

85	Controlling Terminal	Real Group ID
86	Current Working Directory	Real User ID
87	Effective Group ID	Root Directory
88	Effective User ID	Saved Set-Group-ID
89	File Descriptors	Saved Set-User-ID
90	File Mode Creation Mask	Session Membership
91	Process Group ID	Supplementary Group IDs
92	Process ID	

93 A conforming implementation may include additional process attributes.

#### 94 1.7.1.2 Concurrent Execution of Processes

95 The following functionality of the *fork()* function defined in the System Interfaces volume of  
 96 IEEE Std 1003.1-200x shall be available on all systems conforming to this volume of  
 97 IEEE Std 1003.1-200x:

- 98 1. Independent processes shall be capable of executing independently without either process  
 99 terminating.
- 100 2. A process shall be able to create a new process with all of the attributes referenced in  
 101 Section 1.7.1.1, determined according to the semantics of a call to the *fork()* function  
 102 defined in the System Interfaces volume of IEEE Std 1003.1-200x followed by a call in the  
 103 child process to one of the *exec* functions defined in the System Interfaces volume of  
 104 IEEE Std 1003.1-200x.

## 105 1.7.1.3 File Access Permissions

106 The file access control mechanism described by the Base Definitions volume of |  
107 IEEE Std 1003.1-200x, Section 4.4, File Access Permissions shall apply to all files on an |  
108 implementation conforming to this volume of IEEE Std 1003.1-200x. |

## 109 1.7.1.4 File Read, Write, and Creation

110 If a file that does not exist is to be written, it shall be created as described below, unless the |  
111 utility description states otherwise.

112 When a file that does not exist is created, the following features defined in the System Interfaces |  
113 volume of IEEE Std 1003.1-200x shall apply unless the utility or function description states |  
114 otherwise:

- 115 1. The user ID of the file shall be set to the effective user ID of the calling process. |
- 116 2. The group ID of the file shall be set to the effective group ID of the calling process or the |  
117 group ID of the directory in which the file is being created.
- 118 3. If the file is a regular file, the permission bits of the file shall be set to: |

119 S\_IROTH | S\_IWOTH | S\_IRGRP | S\_IWGRP | S\_IRUSR | S\_IWUSR

120 (see the description of *File Modes* in the Base Definitions volume of IEEE Std 1003.1-200x, |  
121 Chapter 13, Headers, <sys/stat.h>) except that the bits specified by the file mode creation |  
122 mask of the process shall be cleared. If the file is a directory, the permission bits shall be set |  
123 to:

124 S\_IRWXU | S\_IRWXG | S\_IRWXO

125 except that the bits specified by the file mode creation mask of the process shall be cleared. |

- 126 4. The *st\_atime*, *st\_ctime*, and *st\_mtime* fields of the file shall be updated as specified in the |  
127 System Interfaces volume of IEEE Std 1003.1-200x, Section 2.5, Standard I/O Streams.
- 128 5. If the file is a directory, it shall be an empty directory; otherwise, the file shall have length |  
129 zero.
- 130 6. If the file is a symbolic link, the effect shall be undefined unless the {POSIX2\_SYMLINKS} |  
131 variable is in effect for the directory in which the symbolic link would be created.
- 132 7. Unless otherwise specified, the file created shall be a regular file.

133 When an attempt is made to create a file that already exists, the action shall depend on the type |  
134 of the file the utility is trying to create and on the type of the existing file as shown in Table 1-1 |  
135 (on page 2205).

136

**Table 1-1** Actions when Creating a File that Already Exists

137

138

139

140

141

142

143

144

145

146

147

148

149

150

151

Existing Type	New Type												Function Creating New
	A	B	C	D	F	L	M	P	Q	R	S	T	
A <i>fattach()</i> -ed STREAM	—	—	—	—	—	—	—	—	—	OF	—	—	<i>fattach()</i>
B Block Special	—	—	—	—	—	—	—	—	—	OF	—	—	<i>mknod()</i> **
C Character Special	—	—	—	—	—	—	—	—	—	OF	—	—	<i>mknod()</i> **
D Directory	—	—	—	F	F	—	—	—	—	OD	—	—	<i>mkdir()</i>
F FIFO Special File	—	—	—	F	F	—	—	—	—	O	—	—	<i>mkfifo()</i>
L Symbolic Link	FL	FL	FL	FL	FL	FL	FL	FL	FL	FL	FL	FL	<i>symlink()</i>
M Shared Memory	—	—	—	—	—	—	—	—	—	—	—	—	<i>shm_open()</i>
P Semaphore	—	—	—	—	—	—	—	—	—	—	—	—	<i>sem_open()</i>
Q Message Queue	—	—	—	—	—	—	—	—	—	—	—	—	<i>mq_open()</i>
R Regular File	—	—	—	F	F	—	—	—	—	RF	—	—	<i>open()</i>
S Socket	—	—	—	—	—	—	—	—	—	—	—	—	<i>bind()</i>
T Typed Memory	—	—	—	—	—	—	—	—	—	—	—	—	*
None.	—	—	—	NF	NF	NF	—	—	—	CF	NF	—	

152

The following codes are used in Table 1-1:

153

CF Create a new file as defined in Section 1.7.1.4 (on page 2204), items 1 through 7.

154

155

156

157

F Fail. When attempting to create a directory or FIFO special file, and the existing file is a directory, FIFO special file, or regular file, the attempt shall fail and the utility shall either continue with its operation or exit immediately with a non-zero exit status, depending on the description of the utility.

158

159

160

161

FL Follow link. Unless otherwise specified, the symbolic links shall be followed as specified for pathname resolution, and the operation performed shall be as if the target of the symbolic link (after all resolution) had been named. If the target of the symbolic link does not exist, it shall be as if that nonexistent target had been named directly.

162

NF Create a new file as described by the appropriate function.

163

164

O Open FIFO. When attempting to create a regular file, and the existing file is a FIFO special file:

165

166

1. If the FIFO is not already open for reading, the attempt shall block until the FIFO is opened for reading.

167

168

2. Once the FIFO is open for reading, the utility shall open the FIFO for writing and continue with its operation.

169

OD The directory shall be opened.

170

OF The named file shall be opened with the consequences defined for that file type.

171

RF Regular file. When attempting to create a regular file, and the existing file is a regular file:

172

173

1. The user ID, group ID, and permission bits of the file shall not be changed.

2. The file shall be truncated to zero length.

174

3. The *st\_ctime* and *st\_mtime* fields shall be marked for update.

175

— The effect is implementation-defined unless specified by the utility description.

176

\* There is no portable way to create a file of this type.

177

\*\* Not portable.

178 When a file is to be appended, the file shall be opened in a manner equivalent to using the  
179 O\_APPEND flag, without the O\_TRUNC flag, in the *open()* function defined in the System  
180 Interfaces volume of IEEE Std 1003.1-200x.

181 When a file is to be read or written, the file shall be opened with an access mode corresponding  
182 to the operation to be performed. If file access permissions deny access, the requested operation  
183 shall fail.

#### 184 1.7.1.5 File Removal

185 When a directory that is the root directory or current working directory of any process is  
186 removed, the effect is implementation-defined. If file access permissions deny access, the  
187 requested operation shall fail. Otherwise, when a file is removed:

- 188 1. Its directory entry shall be removed from the file system.
- 189 2. The link count of the file shall be decremented.
- 190 3. If the file is an empty directory (see the Base Definitions volume of IEEE Std 1003.1-200x,  
191 Section 3.143, Empty Directory):
  - 192 a. If no process has the directory open, the space occupied by the directory shall be  
193 freed and the directory shall no longer be accessible.
  - 194 b. If one or more processes have the directory open, the directory contents shall be  
195 preserved until all references to the file have been closed.
- 196 4. If the file is a directory that is not empty, the *st\_ctime* field shall be marked for update.
- 197 5. If the file is not a directory:
  - 198 a. If the link count becomes zero:
    - 199 i. If no process has the file open, the space occupied by the file shall be freed and  
200 the file shall no longer be accessible.
    - 201 ii. If one or more processes have the file open, the file contents shall be preserved  
202 until all references to the file have been closed.
  - 203 b. If the link count is not reduced to zero, the *st\_ctime* field shall be marked for update.
- 204 6. The *st\_ctime* and *st\_mtime* fields of the containing directory shall be marked for update.

#### 205 1.7.1.6 File Time Values

206 All files shall have the three time values described by the Base Definitions volume of  
207 IEEE Std 1003.1-200x, Section 4.7, File Times Update.

#### 208 1.7.1.7 File Contents

209 When a reference is made to the contents of a file, *pathname*, this means the equivalent of all of  
210 the data placed in the space pointed to by *buf* when performing the *read()* function calls in the  
211 following operations defined in the System Interfaces volume of IEEE Std 1003.1-200x:

```
212 while (read (fildes, buf, nbytes) > 0) |  
213 ; |
```

214 If the file is indicated by a pathname *pathname*, the file descriptor shall be determined by the  
215 equivalent of the following operation defined in the System Interfaces volume of  
216 IEEE Std 1003.1-200x:

217            `fildes = open (pathname, O_RDONLY);` |

218            The value of *nbytes* in the above sequence is unspecified; if the file is of a type where the data  
219            returned by *read()* would vary with different values, the value shall be one that results in the  
220            most data being returned. |

221            If the *read()* function calls would return an error, it is unspecified whether the contents of the file  
222            are considered to include any data from offsets in the file beyond where the error would be  
223            returned. |

#### 224    1.7.1.8    *Pathname Resolution*

225            The pathname resolution algorithm, described by the Base Definitions volume of |  
226            IEEE Std 1003.1-200x, Section 4.11, Pathname Resolution, shall be used by implementations |  
227            conforming to this volume of IEEE Std 1003.1-200x; see also the Base Definitions volume of |  
228            IEEE Std 1003.1-200x, Section 4.5, File Hierarchy. |

#### 229    1.7.1.9    *Changing the Current Working Directory*

230            When the current working directory (see the Base Definitions volume of IEEE Std 1003.1-200x,  
231            Section 3.436, Working Directory) is to be changed, unless the utility or function description  
232            states otherwise, the operation shall succeed unless a call to the *chdir()* function defined in the  
233            System Interfaces volume of IEEE Std 1003.1-200x would fail when invoked with the new  
234            working directory pathname as its argument. |

#### 235    1.7.1.10   *Establish the Locale*

236            The functionality of the *setlocale()* function defined in the System Interfaces volume of |  
237            IEEE Std 1003.1-200x shall be available on all systems conforming to this volume of |  
238            IEEE Std 1003.1-200x; that is, utilities that require the capability of establishing an international  
239            operating environment shall be permitted to set the specified category of the international  
240            environment. |

#### 241    1.7.1.11   *Actions Equivalent to Functions*

242            Some utility descriptions specify that a utility performs actions equivalent to a function defined |  
243            in the System Interfaces volume of IEEE Std 1003.1-200x. Such specifications require only that  
244            the external effects be equivalent, not that any effect within the utility and visible only to the  
245            utility be equivalent. |

### 246    1.7.2       **Concepts Derived from the ISO C Standard**

247            Some of the standard utilities perform complex data manipulation using their own procedure |  
248            and arithmetic languages, as defined in their EXTENDED DESCRIPTION or OPERANDS |  
249            sections. Unless otherwise noted, the arithmetic and semantic concepts (precision, type |  
250            conversion, control flow, and so on) shall be equivalent to those defined in the ISO C standard, |  
251            as described in the following sections. Note that there is no requirement that the standard |  
252            utilities be implemented in any particular programming language. |

#### 253    1.7.2.1     *Arithmetic Precision and Operations*

254            Integer variables and constants, including the values of operands and option-arguments, used |  
255            by the standard utilities listed in this volume of IEEE Std 1003.1-200x shall be implemented as |  
256            equivalent to the ISO C standard **signed long** data type; floating point shall be implemented as |  
257            equivalent to the ISO C standard **double** type. Conversions between types shall be as described |  
258            in the ISO C standard. All variables shall be initialized to zero if they are not otherwise assigned |

259           by the input to the application. |  
260           Arithmetic operators and functions shall be implemented as equivalent to those in the cited |  
261           ISO C standard section, as listed in Table 1-2 (on page 2209). |

Table 1-2 ISO C Standard Operators and Functions

Operation	ISO C Standard Equivalent Reference
( )	Section 6.5.1, Primary Expressions
postfix ++ postfix --	Section 6.5.2, Postfix Operators
unary + unary - prefix ++ prefix -- ~ ! sizeof()	Section 6.5.3, Unary Operators
* / %	Section 6.5.5, Multiplicative Operators
+ -	Section 6.5.6, Additive Operators
<< >>	Section 6.5.7, Bitwise Shift Operators
<, <= >, >=	Section 6.5.8, Relational Operators
== !=	Section 6.5.9, Equality Operators
&	Section 6.5.10, Bitwise AND Operator
^	Section 6.5.11, Bitwise Exclusive OR Operator
	Section 6.5.12, Bitwise Inclusive OR Operator
&&	Section 6.5.13, Logical AND Operator
	Section 6.5.14, Logical OR Operator
expr?expr:expr	Section 6.5.15, Conditional Operator
=, *=, /=, %= <<=, >>=, &=, ^=,  =	Section 6.5.16, Assignment Operators
if () if () ... else switch ()	Section 6.8.4, Selection Statements
while () do ... while () for ()	Section 6.8.5, Iteration Statements
goto continue break return	Section 6.8.6, Jump Statements

The evaluation of arithmetic expressions shall be equivalent to that described in Section 6.5, Expressions, of the ISO C standard.

306 1.7.2.2 *Mathematical Functions*

307 Any mathematical functions with the same names as those in the following sections of the ISO C  
308 standard:

- 309 • Section 7.12, Mathematics, `<math.h>`
- 310 • Section 7.20.2, Pseudo-Random Sequence Generation Functions

311 shall be implemented to return the results equivalent to those returned from a call to the  
312 corresponding function described in the ISO C standard.

313 **1.8 Portability**

314 Some of the utilities in the Shell and Utilities volume of IEEE Std 1003.1-200x and functions in  
315 the System Interfaces volume of IEEE Std 1003.1-200x describe functionality that might not be  
316 fully portable to systems meeting the requirements for POSIX conformance (see the Base  
317 Definitions volume of IEEE Std 1003.1-200x, Chapter 2, Conformance).

318 Where optional, enhanced, or reduced functionality is specified, the text is shaded and a code in  
319 the margin identifies the nature of the option, extension, or warning (see Section 1.8.1). For  
320 maximum portability, an application should avoid such functionality.

321 Unless the primary task of a utility is to produce textual material on its standard output,  
322 application developers should not rely on the format or content of any such material that may be  
323 produced. Where the primary task *is* to provide such material, but the output format is  
324 incompletely specified, the description is marked with the OF margin code and shading.  
325 Application developers are warned not to expect that the output of such an interface on one  
326 system is any guide to its behavior on another system.

327 **1.8.1 Codes**

328 Codes and their meanings are listed in the Base Definitions volume of IEEE Std 1003.1-200x, but  
329 are repeated here for convenience:

330 ADV **Advisory Information**

331 The functionality described is optional. The functionality described is also an extension to the  
332 ISO C standard.

333 Where applicable, functions are marked with the ADV margin legend in the SYNOPSIS section.  
334 Where additional semantics apply to a function, the material is identified by use of the ADV  
335 margin legend.

336 AIO **Asynchronous Input and Output**

337 The functionality described is optional. The functionality described is also an extension to the  
338 ISO C standard.

339 Where applicable, functions are marked with the AIO margin legend in the SYNOPSIS section.  
340 Where additional semantics apply to a function, the material is identified by use of the AIO  
341 margin legend.

342 BAR **Barriers**

343 The functionality described is optional. The functionality described is also an extension to the  
344 ISO C standard.

345 Where applicable, functions are marked with the BAR margin legend in the SYNOPSIS section.  
346 Where additional semantics apply to a function, the material is identified by use of the BAR  
347 margin legend.

348	BE	<b>Batch Environment Services and Utilities</b>
349		The functionality described is optional.
350		Where applicable, utilities are marked with the BE margin legend in the SYNOPSIS section.
351		Where additional semantics apply to a utility, the material is identified by use of the BE margin
352		legend.
353	CD	<b>C-Language Development Utilities</b>
354		The functionality described is optional.
355		Where applicable, utilities are marked with the CD margin legend in the SYNOPSIS section.
356		Where additional semantics apply to a utility, the material is identified by use of the CD margin
357		legend.
358	CPT	<b>Process CPU-Time Clocks</b>
359		The functionality described is optional. The functionality described is also an extension to the
360		ISO C standard.
361		Where applicable, functions are marked with the CPT margin legend in the SYNOPSIS section.
362		Where additional semantics apply to a function, the material is identified by use of the CPT
363		margin legend.
364	CS	<b>Clock Selection</b>
365		The functionality described is optional. The functionality described is also an extension to the
366		ISO C standard.
367		Where applicable, functions are marked with the CS margin legend in the SYNOPSIS section.
368		Where additional semantics apply to a function, the material is identified by use of the CS
369		margin legend.
370	CX	<b>Extension to the ISO C standard</b>
371		The functionality described is an extension to the ISO C standard. Application writers may make
372		use of an extension as it is supported on all IEEE Std 1003.1-200x-conforming systems.
373		With each function or header from the ISO C standard, a statement to the effect that “any
374		conflict is unintentional” is included. That is intended to refer to a direct conflict.
375		IEEE Std 1003.1-200x acts in part as a profile of the ISO C standard, and it may choose to further
376		constrain behaviors allowed to vary by the ISO C standard. Such limitations are not considered
377		conflicts.
378	FD	<b>FORTTRAN Development Utilities</b>
379		The functionality described is optional.
380		Where applicable, utilities are marked with the FD margin legend in the SYNOPSIS section.
381		Where additional semantics apply to a utility, the material is identified by use of the FD margin
382		legend.
383	FR	<b>FORTTRAN Runtime Utilities</b>
384		The functionality described is optional.
385		Where applicable, utilities are marked with the FR margin legend in the SYNOPSIS section.
386		Where additional semantics apply to a utility, the material is identified by use of the FR margin
387		legend.
388	FSC	<b>File Synchronization</b>
389		The functionality described is optional. The functionality described is also an extension to the
390		ISO C standard.
391		Where applicable, functions are marked with the FSC margin legend in the SYNOPSIS section.
392		Where additional semantics apply to a function, the material is identified by use of the FSC

- 393 margin legend.
- 394 IP6 **IPV6**  
395 The functionality described is optional. The functionality described is also an extension to the  
396 ISO C standard.
- 397 Where applicable, functions are marked with the IP6 margin legend in the SYNOPSIS section.  
398 Where additional semantics apply to a function, the material is identified by use of the IP6  
399 margin legend.
- 400 MC1 **Advisory Information and either Memory Mapped Files or Shared Memory Objects**  
401 The functionality described is optional. The functionality described is also an extension to the  
402 ISO C standard.
- 403 This is a shorthand notation for combinations of multiple option codes.
- 404 Where applicable, functions are marked with the MC1 margin legend in the SYNOPSIS section.  
405 Where additional semantics apply to a function, the material is identified by use of the MC1  
406 margin legend.
- 407 Refer to the Base Definitions volume of IEEE Std 1003.1-200x, Section 1.5.2, Margin Code  
408 Notation.
- 409 MC2 **Memory Mapped Files, Shared Memory Objects, or Memory Protection**  
410 The functionality described is optional. The functionality described is also an extension to the  
411 ISO C standard.
- 412 This is a shorthand notation for combinations of multiple option codes.
- 413 Where applicable, functions are marked with the MC2 margin legend in the SYNOPSIS section.  
414 Where additional semantics apply to a function, the material is identified by use of the MC2  
415 margin legend.
- 416 Refer to the Base Definitions volume of IEEE Std 1003.1-200x, Section 1.5.2, Margin Code  
417 Notation.
- 418 MF **Memory Mapped Files**  
419 The functionality described is optional. The functionality described is also an extension to the  
420 ISO C standard.
- 421 Where applicable, functions are marked with the MF margin legend in the SYNOPSIS section.  
422 Where additional semantics apply to a function, the material is identified by use of the MF  
423 margin legend.
- 424 ML **Process Memory Locking**  
425 The functionality described is optional. The functionality described is also an extension to the  
426 ISO C standard.
- 427 Where applicable, functions are marked with the ML margin legend in the SYNOPSIS section.  
428 Where additional semantics apply to a function, the material is identified by use of the ML  
429 margin legend.
- 430 MLR **Range Memory Locking**  
431 The functionality described is optional. The functionality described is also an extension to the  
432 ISO C standard.
- 433 Where applicable, functions are marked with the MLR margin legend in the SYNOPSIS section.  
434 Where additional semantics apply to a function, the material is identified by use of the MLR  
435 margin legend.

- 436 MON **Monotonic Clock**  
 437 The functionality described is optional. The functionality described is also an extension to the  
 438 ISO C standard.
- 439 Where applicable, functions are marked with the MON margin legend in the SYNOPSIS section.  
 440 Where additional semantics apply to a function, the material is identified by use of the MON  
 441 margin legend.
- 442 MPR **Memory Protection**  
 443 The functionality described is optional. The functionality described is also an extension to the  
 444 ISO C standard.
- 445 Where applicable, functions are marked with the MPR margin legend in the SYNOPSIS section.  
 446 Where additional semantics apply to a function, the material is identified by use of the MPR  
 447 margin legend.
- 448 MSG **Message Passing**  
 449 The functionality described is optional. The functionality described is also an extension to the  
 450 ISO C standard.
- 451 Where applicable, functions are marked with the MSG margin legend in the SYNOPSIS section.  
 452 Where additional semantics apply to a function, the material is identified by use of the MSG  
 453 margin legend.
- 454 MX **IEC 60559 Floating-Point Option**  
 455 The functionality described is optional. The functionality described is also an extension to the  
 456 ISO C standard.
- 457 Where applicable, functions are marked with the MX margin legend in the SYNOPSIS section.  
 458 Where additional semantics apply to a function, the material is identified by use of the MX  
 459 margin legend.
- 460 OB **Obsolescent**  
 461 The functionality described may be withdrawn in a future version of this volume of  
 462 IEEE Std 1003.1-200x. Strictly Conforming POSIX Applications and Strictly Conforming XSI  
 463 Applications shall not use obsolescent features.
- 464 OF **Output Format Incompletely Specified**  
 465 The functionality described is an XSI extension. The format of the output produced by the utility  
 466 is not fully specified. It is therefore not possible to post-process this output in a consistent  
 467 fashion. Typical problems include unknown length of strings and unspecified field delimiters.
- 468 OH **Optional Header**  
 469 In the SYNOPSIS section of some interfaces in the System Interfaces volume of  
 470 IEEE Std 1003.1-200x an included header is marked as in the following example:
- 471 OH `#include <sys/types.h>`  
 472 `#include <grp.h>`  
 473 `struct group *getgrnam(const char *name);`
- 474 This indicates that the marked header is not required on XSI-conformant systems.
- 475 PIO **Prioritized Input and Output**  
 476 The functionality described is optional. The functionality described is also an extension to the  
 477 ISO C standard.
- 478 Where applicable, functions are marked with the PIO margin legend in the SYNOPSIS section.  
 479 Where additional semantics apply to a function, the material is identified by use of the PIO  
 480 margin legend.

481	PS	<b>Process Scheduling</b>
482		The functionality described is optional. The functionality described is also an extension to the
483		ISO C standard.
484		Where applicable, functions are marked with the PS margin legend in the SYNOPSIS section.
485		Where additional semantics apply to a function, the material is identified by use of the PS
486		margin legend.
487	RS	<b>Raw Sockets</b>
488		The functionality described is optional. The functionality described is also an extension to the
489		ISO C standard.
490		Where applicable, functions are marked with the RS margin legend in the SYNOPSIS section.
491		Where additional semantics apply to a function, the material is identified by use of the RS
492		margin legend.
493	RTS	<b>Realtime Signals Extension</b>
494		The functionality described is optional. The functionality described is also an extension to the
495		ISO C standard.
496		Where applicable, functions are marked with the RTS margin legend in the SYNOPSIS section.
497		Where additional semantics apply to a function, the material is identified by use of the RTS
498		margin legend.
499	SD	<b>Software Development Utilities</b>
500		The functionality described is optional.
501		Where applicable, utilities are marked with the SD margin legend in the SYNOPSIS section.
502		Where additional semantics apply to a utility, the material is identified by use of the SD
503		margin legend.
504	SEM	<b>Semaphores</b>
505		The functionality described is optional. The functionality described is also an extension to the
506		ISO C standard.
507		Where applicable, functions are marked with the SEM margin legend in the SYNOPSIS section.
508		Where additional semantics apply to a function, the material is identified by use of the SEM
509		margin legend.
510	SHM	<b>Shared Memory Objects</b>
511		The functionality described is optional. The functionality described is also an extension to the
512		ISO C standard.
513		Where applicable, functions are marked with the SHM margin legend in the SYNOPSIS section.
514		Where additional semantics apply to a function, the material is identified by use of the SHM
515		margin legend.
516	SIO	<b>Synchronized Input and Output</b>
517		The functionality described is optional. The functionality described is also an extension to the
518		ISO C standard.
519		Where applicable, functions are marked with the SIO margin legend in the SYNOPSIS section.
520		Where additional semantics apply to a function, the material is identified by use of the SIO
521		margin legend.
522	SPI	<b>Spin Locks</b>
523		The functionality described is optional. The functionality described is also an extension to the
524		ISO C standard.

525           Where applicable, functions are marked with the SPI margin legend in the SYNOPSIS section.  
526           Where additional semantics apply to a function, the material is identified by use of the SPI  
527           margin legend.

528    SPN       **Spawn**  
529           The functionality described is optional. The functionality described is also an extension to the  
530           ISO C standard.

531           Where applicable, functions are marked with the SPN margin legend in the SYNOPSIS section.  
532           Where additional semantics apply to a function, the material is identified by use of the SPN  
533           margin legend.

534    SS         **Process Sporadic Server**  
535           The functionality described is optional. The functionality described is also an extension to the  
536           ISO C standard.

537           Where applicable, functions are marked with the SS margin legend in the SYNOPSIS section.  
538           Where additional semantics apply to a function, the material is identified by use of the SS  
539           margin legend.

540    TCT       **Thread CPU-Time Clocks**  
541           The functionality described is optional. The functionality described is also an extension to the  
542           ISO C standard.

543           Where applicable, functions are marked with the TCT margin legend in the SYNOPSIS section.  
544           Where additional semantics apply to a function, the material is identified by use of the TCT  
545           margin legend.

546    TEF       **Trace Event Filter**  
547           The functionality described is optional. The functionality described is also an extension to the  
548           ISO C standard.

549           Where applicable, functions are marked with the TEF margin legend in the SYNOPSIS section.  
550           Where additional semantics apply to a function, the material is identified by use of the TEF  
551           margin legend.

552    THR       **Threads**  
553           The functionality described is optional. The functionality described is also an extension to the  
554           ISO C standard.

555           Where applicable, functions are marked with the THR margin legend in the SYNOPSIS section.  
556           Where additional semantics apply to a function, the material is identified by use of the THR  
557           margin legend.

558    TMO       **Timeouts**  
559           The functionality described is optional. The functionality described is also an extension to the  
560           ISO C standard.

561           Where applicable, functions are marked with the TMO margin legend in the SYNOPSIS section.  
562           Where additional semantics apply to a function, the material is identified by use of the TMO  
563           margin legend.

564    TMR       **Timers**  
565           The functionality described is optional. The functionality described is also an extension to the  
566           ISO C standard.

567           Where applicable, functions are marked with the TMR margin legend in the SYNOPSIS section.  
568           Where additional semantics apply to a function, the material is identified by use of the TMR  
569           margin legend.

570	TPI	<b>Thread Priority Inheritance</b>
571		The functionality described is optional. The functionality described is also an extension to the
572		ISO C standard.
573		Where applicable, functions are marked with the TPI margin legend in the SYNOPSIS section.
574		Where additional semantics apply to a function, the material is identified by use of the TPI
575		margin legend.
576	TPP	<b>Thread Priority Protection</b>
577		The functionality described is optional. The functionality described is also an extension to the
578		ISO C standard.
579		Where applicable, functions are marked with the TPP margin legend in the SYNOPSIS section.
580		Where additional semantics apply to a function, the material is identified by use of the TPP
581		margin legend.
582	TPS	<b>Thread Execution Scheduling</b>
583		The functionality described is optional. The functionality described is also an extension to the
584		ISO C standard.
585		Where applicable, functions are marked with the TPS margin legend for the SYNOPSIS section.
586		Where additional semantics apply to a function, the material is identified by use of the TPS
587		margin legend.
588	TRC	<b>Trace</b>
589		The functionality described is optional. The functionality described is also an extension to the
590		ISO C standard.
591		Where applicable, functions are marked with the TRC margin legend in the SYNOPSIS section.
592		Where additional semantics apply to a function, the material is identified by use of the TRC
593		margin legend.
594	TRI	<b>Trace Inherit</b>
595		The functionality described is optional. The functionality described is also an extension to the
596		ISO C standard.
597		Where applicable, functions are marked with the TRI margin legend in the SYNOPSIS section.
598		Where additional semantics apply to a function, the material is identified by use of the TRI
599		margin legend.
600	TRL	<b>Trace Log</b>
601		The functionality described is optional. The functionality described is also an extension to the
602		ISO C standard.
603		Where applicable, functions are marked with the TRL margin legend in the SYNOPSIS section.
604		Where additional semantics apply to a function, the material is identified by use of the TRL
605		margin legend.
606	TSA	<b>Thread Stack Address Attribute</b>
607		The functionality described is optional. The functionality described is also an extension to the
608		ISO C standard.
609		Where applicable, functions are marked with the TSA margin legend for the SYNOPSIS section.
610		Where additional semantics apply to a function, the material is identified by use of the TSA
611		margin legend.
612	TSF	<b>Thread-Safe Functions</b>
613		The functionality described is optional. The functionality described is also an extension to the
614		ISO C standard.

615 Where applicable, functions are marked with the TSF margin legend in the SYNOPSIS section.  
616 Where additional semantics apply to a function, the material is identified by use of the TSF  
617 margin legend.

618 TSH **Thread Process-Shared Synchronization**  
619 The functionality described is optional. The functionality described is also an extension to the  
620 ISO C standard.

621 Where applicable, functions are marked with the TSH margin legend in the SYNOPSIS section.  
622 Where additional semantics apply to a function, the material is identified by use of the TSH  
623 margin legend.

624 TSP **Thread Sporadic Server**  
625 The functionality described is optional. The functionality described is also an extension to the  
626 ISO C standard.

627 Where applicable, functions are marked with the TSP margin legend in the SYNOPSIS section.  
628 Where additional semantics apply to a function, the material is identified by use of the TSP  
629 margin legend.

630 TSS **Thread Stack Address Size**  
631 The functionality described is optional. The functionality described is also an extension to the  
632 ISO C standard.

633 Where applicable, functions are marked with the TSS margin legend in the SYNOPSIS section.  
634 Where additional semantics apply to a function, the material is identified by use of the TSS  
635 margin legend.

636 TYM **Typed Memory Objects**  
637 The functionality described is optional. The functionality described is also an extension to the  
638 ISO C standard.

639 Where applicable, functions are marked with the TYM margin legend in the SYNOPSIS section.  
640 Where additional semantics apply to a function, the material is identified by use of the TYM  
641 margin legend.

642 UP **User Portability Utilities**  
643 The functionality described is optional.

644 Where applicable, utilities are marked with the UP margin legend in the SYNOPSIS section.  
645 Where additional semantics apply to a utility, the material is identified by use of the UP margin  
646 legend.

647 XSI **Extension**  
648 The functionality described is an XSI extension. Functionality marked XSI is also an extension to  
649 the ISO C standard. Application writers may confidently make use of an extension on all  
650 systems supporting the X/Open System Interfaces Extension.

651 If an entire SYNOPSIS section is shaded and marked XSI, all the functionality described in that  
652 reference page is an extension. See the Base Definitions volume of IEEE Std 1003.1-200x, Section  
653 3.439, XSI.

654 XSR **XSI STREAMS**  
655 The functionality described is optional. The functionality described is also an extension to the  
656 ISO C standard.

657 Where applicable, functions are marked with the XSR margin legend in the SYNOPSIS section.  
658 Where additional semantics apply to a function, the material is identified by use of the XSR  
659 margin legend.

660 **1.9 Utility Limits**

661 This section lists magnitude limitations imposed by a specific implementation. The braces  
 662 notation, {LIMIT}, is used in this volume of IEEE Std 1003.1-200x to indicate these values, but the  
 663 braces are not part of the name.

664 **Table 1-3 Utility Limit Minimum Values**

Name	Description	Value
{POSIX2_BC_BASE_MAX}	The maximum <i>obase</i> value allowed by the <i>bc</i> utility.	99
{POSIX2_BC_DIM_MAX}	The maximum number of elements permitted in an array by the <i>bc</i> utility.	2048
{POSIX2_BC_SCALE_MAX}	The maximum <i>scale</i> value allowed by the <i>bc</i> utility.	99
{POSIX2_BC_STRING_MAX}	The maximum length of a string constant accepted by the <i>bc</i> utility.	1000
{POSIX2_COLL_WEIGHTS_MAX}	The maximum number of weights that can be assigned to an entry of the <i>LC_COLLATE order</i> keyword in the locale definition file; see the <b>border_start</b> keyword in the Base Definitions volume of IEEE Std 1003.1-200x, Section 7.3.2, <i>LC_COLLATE</i> .	2
{POSIX2_EXPR_NEST_MAX}	The maximum number of expressions that can be nested within parentheses by the <i>expr</i> utility.	32
{POSIX2_LINE_MAX}	Unless otherwise noted, the maximum length, in bytes, of the input line of a utility (either standard input or another file), when the utility is described as processing text files. The length includes room for the trailing newline.	2048
{POSIX2_RE_DUP_MAX}	The maximum number of repeated occurrences of a BRE permitted when using the interval notation $\{m,n\}$ ; see the Base Definitions volume of IEEE Std 1003.1-200x, Section 9.3.6, BREs Matching Multiple Characters.	255
{POSIX2_VERSION}	This value indicates the version of the utilities in this volume of IEEE Std 1003.1-200x that are provided by the implementation. It changes with each published version.	200xxxL

697 The values specified in Table 1-3 represent the lowest values conforming implementations shall  
 698 provide and, consequently, the largest values on which an application can rely without further  
 699 enquiries, as described below. These values shall be accessible to applications via the *getconf*  
 700 utility (see *getconf* (on page 2683)) and through the *sysconf*() function defined in the System  
 701 Interfaces volume of IEEE Std 1003.1-200x. The literal names shown in Table 1-3 apply only to  
 702 the *getconf* utility; the high-level language binding describes the exact form of each name to be  
 703 used by the interfaces in that binding.

704 Implementations may provide more liberal, or less restrictive, values than shown in Table 1-3.  
 705 These possibly more liberal values are accessible using the symbols in Table 1-4 (on page 2219).

706 The `sysconf()` function defined in the System Interfaces volume of IEEE Std 1003.1-200x or the  
 707 `getconf` utility return the value of each symbol on each specific implementation. The value so  
 708 retrieved is the largest, or most liberal, value that is available throughout the session lifetime, as  
 709 determined at session creation. The literal names shown in the table apply only to the `getconf`  
 710 utility; the high-level language binding describes the exact form of each name to be used by the  
 711 interfaces in that binding.

712 All numeric limits defined by the System Interfaces volume of IEEE Std 1003.1-200x, such as  
 713 {PATH\_MAX}, shall also apply to this volume of IEEE Std 1003.1-200x. All the utilities defined  
 714 by this volume of IEEE Std 1003.1-200x are implicitly limited by these values, unless otherwise  
 715 noted in the utility descriptions.

716 It is not guaranteed that the application can actually reach the specified limit of an  
 717 implementation in any given case, or at all, as a lack of virtual memory or other resources may  
 718 prevent this. The limit value indicates only that the implementation does not specifically impose  
 719 any arbitrary, more restrictive limit.

720 **Table 1-4** Symbolic Utility Limits

Name	Description	Minimum Value
{BC_BASE_MAX}	The maximum <i>obase</i> value allowed by the <i>bc</i> utility.	{POSIX2_BC_BASE_MAX}
{BC_DIM_MAX}	The maximum number of elements permitted in an array by the <i>bc</i> utility.	{POSIX2_BC_DIM_MAX}
{BC_SCALE_MAX}	The maximum <i>scale</i> value allowed by the <i>bc</i> utility.	{POSIX2_BC_SCALE_MAX}
{BC_STRING_MAX}	The maximum length of a string constant accepted by the <i>bc</i> utility.	{POSIX2_BC_STRING_MAX}
{COLL_WEIGHTS_MAX}	The maximum number of weights that can be assigned to an entry of the <i>LC_COLLATE</i> <b>order</b> keyword in the locale definition file; see the <b>order_start</b> keyword in the Base Definitions volume of IEEE Std 1003.1-200x, Section 7.3.2, <i>LC_COLLATE</i> .	{POSIX2_COLL_WEIGHTS_MAX}
{EXPR_NEST_MAX}	The maximum number of expressions that can be nested within parentheses by the <i>expr</i> utility.	{POSIX2_EXPR_NEST_MAX}
{LINE_MAX}	Unless otherwise noted, the maximum length, in bytes, of the input line of a utility (either standard input or another file), when the utility is described as	{POSIX2_LINE_MAX}

754  
755  
756  
757  
758  
759  
760  
761  
762  
763  
764  
765  
766  
767  
768

Name	Description	Minimum Value
{RE_DUP_MAX}	processing text files. The length includes room for the trailing newline. The maximum number of repeated occurrences of a BRE permitted when using the interval notation $\{m,n\}$ ; see the Base Definitions volume of IEEE Std 1003.1-200x, Section 9.3.6, BREs Matching Multiple Characters.	{POSIX2_RE_DUP_MAX}

769  
770  
771  
772  
773

The following value may be a constant within an implementation or may vary from one pathname to another.

{POSIX2\_SYMLINKS}

When referring to a directory, the system supports the creation of symbolic links within that directory; for non-directory files, the meaning of {POSIX2\_SYMLINKS} is undefined.

## 774 1.10 Grammar Conventions

775  
776  
777  
778  
779  
780  
781  
782  
783  
784

Portions of this volume of IEEE Std 1003.1-200x are expressed in terms of a special grammar notation. It is used to portray the complex syntax of certain program input. The grammar is based on the syntax used by the *yacc* utility. However, it does not represent fully functional *yacc* input, suitable for program use; the lexical processing and all semantic requirements are described only in textual form. The grammar is not based on source used in any traditional implementation and has not been tested with the semantic code that would normally be required to accompany it. Furthermore, there is no implication that the partial *yacc* code presented represents the most efficient, or only, means of supporting the complex syntax within the utility. Implementations may use other programming languages or algorithms, as long as the syntax supported is the same as that represented by the grammar.

785  
786

The following typographical conventions are used in the grammar; they have no significance except to aid in reading.

787  
788  
789  
790  
791

- The identifiers for the reserved words of the language are shown with a leading capital letter. (These are terminals in the grammar; for example, **While**, **Case**.)
- The identifiers for terminals in the grammar are all named with uppercase letters and underscores; for example, **NEWLINE**, **ASSIGN\_OP**, **NAME**.
- The identifiers for non-terminals are all lowercase.

## 792 1.11 Utility Description Defaults

793 This section describes all of the subsections used within the utility descriptions, including:

- 794 • Intended usage of the section
- 795 • Global defaults that affect all the standard utilities
- 796 • The meanings of notations used in this volume of IEEE Std 1003.1-200x that are specific to
- 797 individual utility sections

### 798 NAME

799 This section gives the name or names of the utility and briefly states its purpose.

### 800 SYNOPSIS

801 The SYNOPSIS section summarizes the syntax of the calling sequence for the utility,  
 802 including options, option-arguments, and operands. Standards for utility naming are  
 803 described in the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2, Utility  
 804 Syntax Guidelines; for describing the utility's arguments in the Base Definitions volume  
 805 of IEEE Std 1003.1-200x, Section 12.1, Utility Argument Syntax.

### 806 DESCRIPTION

807 The DESCRIPTION section describes the actions of the utility. If the utility has a very  
 808 complex set of subcommands or its own procedural language, an EXTENDED  
 809 DESCRIPTION section is also provided. Most explanations of optional functionality are  
 810 omitted here, as they are usually explained in the OPTIONS section.

811 As stated in Section 1.7.1.11 (on page 2207), some functions are described in terms of  
 812 equivalent functionality. When specific functions are cited, the implementation shall  
 813 provide equivalent functionality including side effects associated with successful  
 814 execution of the function. The treatment of errors and intermediate results from the  
 815 individual functions cited is generally not specified by this volume of  
 816 IEEE Std 1003.1-200x. See the utility's EXIT STATUS and CONSEQUENCES OF  
 817 ERRORS sections for all actions associated with errors encountered by the utility.

### 818 OPTIONS

819 The OPTIONS section describes the utility options and option-arguments, and how  
 820 they modify the actions of the utility. Standard utilities that have options either fully  
 821 comply with the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2, Utility  
 822 Syntax Guidelines or describe all deviations. Apparent disagreements between  
 823 functionality descriptions in the OPTIONS and DESCRIPTION (or EXTENDED  
 824 DESCRIPTION) sections are always resolved in favor of the OPTIONS section.

825 Each OPTIONS section that uses the phrase "The ... utility shall conform to the Utility  
 826 Syntax Guidelines ..." refers only to the use of the utility as specified by this volume of  
 827 IEEE Std 1003.1-200x; implementation extensions should also conform to the  
 828 guidelines, but may allow exceptions for historical practice.

829 Unless otherwise stated in the utility description, when given an option unrecognized  
 830 by the implementation, or when a required option-argument is not provided, standard  
 831 utilities shall issue a diagnostic message to standard error and exit with a non-zero exit  
 832 status.

833 All utilities in this volume of IEEE Std 1003.1-200x shall be capable of processing  
 834 arguments using eight-bit transparency.

835 **Default Behavior:** When this section is listed as "None.", it means that the  
 836 implementation need not support any options. Standard utilities that do not accept  
 837 options, but that do accept operands, shall recognize "--" as a first argument to be

838 discarded.

839 The requirement for recognizing "--" is because conforming applications need a way |  
840 to shield their operands from any arbitrary options that the implementation may |  
841 provide as an extension. For example, if the standard utility *foo* is listed as taking no |  
842 options, and the application needed to give it a pathname with a leading hyphen, it |  
843 could safely do it as:

```
844     foo -- -myfile
```

845 and avoid any problems with **-m** used as an extension.

#### 846 OPERANDS

847 The OPERANDS section describes the utility operands, and how they affect the actions |  
848 of the utility. Apparent disagreements between functionality descriptions in the |  
849 OPERANDS and DESCRIPTION (or EXTENDED DESCRIPTION) sections shall be |  
850 resolved in favor of the OPERANDS section.

851 If an operand naming a file can be specified as '-', which means to use the standard |  
852 input instead of a named file, this is explicitly stated in this section. Unless otherwise |  
853 stated, the use of multiple instances of '-' to mean standard input in a single |  
854 command produces unspecified results.

855 Unless otherwise stated, the standard utilities that accept operands shall process those |  
856 operands in the order specified in the command line.

857 **Default Behavior:** When this section is listed as "None.", it means that the |  
858 implementation need not support any operands.

#### 859 STDIN

860 The STDIN section describes the standard input of the utility. This section is frequently |  
861 merely a reference to the following section, as many utilities treat standard input and |  
862 input files in the same manner. Unless otherwise stated, all restrictions described in the |  
863 INPUT FILES section shall apply to this section as well.

864 Use of a terminal for standard input can cause any of the standard utilities that read |  
865 standard input to stop when used in the background. For this reason, applications |  
866 should not use interactive features in scripts to be placed in the background.

867 The specified standard input format of the standard utilities shall not depend on the |  
868 existence or value of the environment variables defined in this volume of |  
869 IEEE Std 1003.1-200x, except as provided by this volume of IEEE Std 1003.1-200x.

870 **Default Behavior:** When this section is listed as "Not used.", it means that the |  
871 standard input shall not be read when the utility is used as described by this volume of |  
872 IEEE Std 1003.1-200x.

#### 873 INPUT FILES

874 The INPUT FILES section describes the files, other than the standard input, used as |  
875 input by the utility. It includes files named as operands and option-arguments as well |  
876 as other files that are referred to, such as start-up and initialization files, databases, and |  
877 so on. Commonly-used files are generally described in one place and cross-referenced |  
878 by other utilities.

879 All utilities in this volume of IEEE Std 1003.1-200x shall be capable of processing input |  
880 files using eight-bit transparency.

881 When a standard utility reads a seekable input file and terminates without an error |  
882 before it reaches end-of-file, the utility shall ensure that the file offset in the open file

883 description is properly positioned just past the last byte processed by the utility. For  
 884 files that are not seekable, the state of the file offset in the open file description for that  
 885 file is unspecified. A conforming application shall not assume that the following three  
 886 commands are equivalent:

```
887     tail -n +2 file
888     (sed -n 1q; cat) < file
889     cat file | (sed -n 1q; cat)
```

890 The second command is equivalent to the first only when the file is seekable. The third  
 891 command leaves the file offset in the open file description in an unspecified state. Other  
 892 utilities, such as *head*, *read*, and *sh*, have similar properties.

893 Some of the standard utilities, such as filters, process input files a line or a block at a  
 894 time and have no restrictions on the maximum input file size. Some utilities may have  
 895 size limitations that are not as obvious as file space or memory limitations. Such  
 896 limitations should reflect resource limitations of some sort, not arbitrary limits set by  
 897 implementors. Implementations shall document those utilities that are limited by  
 898 constraints other than file system space, available memory, and other limits specifically  
 899 cited by this volume of IEEE Std 1003.1-200x, and identify what the constraint is and  
 900 indicate a way of estimating when the constraint would be reached. Similarly, some  
 901 utilities descend the directory tree (recursively). Implementations shall also document  
 902 any limits that they may have in descending the directory tree that are beyond limits  
 903 cited by this volume of IEEE Std 1003.1-200x.

904 When an input file is described as a *text file*, the utility produces undefined results if  
 905 given input that is not from a text file, unless otherwise stated. Some utilities (for  
 906 example, *make*, *read*, *sh*) allow for continued input lines using an escaped <newline>  
 907 convention; unless otherwise stated, the utility need not be able to accumulate more  
 908 than {LINE\_MAX} bytes from a set of multiple, continued input lines. Thus, for a  
 909 conforming application the total of all the continued lines in a set cannot exceed  
 910 {LINE\_MAX}. If a utility using the escaped <newline> convention detects an end-of-  
 911 file condition immediately after an escaped <newline>, the results are unspecified.

912 Record formats are described in a notation similar to that used by the C-language  
 913 function, *printf()*. See the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 5,  
 914 File Format Notation for a description of this notation. The format description is  
 915 intended to be sufficiently rigorous to allow other applications to generate these input  
 916 files. However, since <blank>s can legitimately be included in some of the fields  
 917 described by the standard utilities, particularly in locales other than the POSIX locale,  
 918 this intent is not always realized.

919 **Default Behavior:** When this section is listed as “None.”, it means that no input files  
 920 are required to be supplied when the utility is used as described by this volume of  
 921 IEEE Std 1003.1-200x.

## 922 ENVIRONMENT VARIABLES

923 The ENVIRONMENT VARIABLES section lists what variables affect the utility’s  
 924 execution.

925 The entire manner in which environment variables described in this volume of  
 926 IEEE Std 1003.1-200x affect the behavior of each utility is described in the  
 927 ENVIRONMENT VARIABLES section for that utility, in conjunction with the global  
 928 XSI effects of the *LANG*, *LC\_ALL*, and *NLSPATH* environment variables described in the  
 929 Base Definitions volume of IEEE Std 1003.1-200x, Chapter 8, Environment Variables.  
 930 The existence or value of environment variables described in this volume of

931 IEEE Std 1003.1-200x shall not otherwise affect the specified behavior of the standard  
932 utilities. Any effects of the existence or value of environment variables not described by  
933 this volume of IEEE Std 1003.1-200x upon the standard utilities are unspecified.

934 For those standard utilities that use environment variables as a means for selecting a  
935 utility to execute (such as *CC* in *make*), the string provided to the utility is subjected to  
936 the path search described for *PATH* in the Base Definitions volume of  
937 IEEE Std 1003.1-200x, Chapter 8, Environment Variables.

938 All utilities in this volume of IEEE Std 1003.1-200x shall be capable of processing |  
939 environment variable names and values using eight-bit transparency. |

940 **Default Behavior:** When this section is listed as “None.”, it means that the behavior of  
941 the utility is not directly affected by environment variables described by this volume of  
942 IEEE Std 1003.1-200x when the utility is used as described by this volume of  
943 IEEE Std 1003.1-200x.

#### 944 ASYNCHRONOUS EVENTS

945 The ASYNCHRONOUS EVENTS section lists how the utility reacts to such events as  
946 signals and what signals are caught.

947 **Default Behavior:** When this section is listed as “Default.”, or it refers to “the standard  
948 action for all other signals; see Section 1.11 (on page 2221)” it means that the action  
949 taken as a result of the signal shall be one of the following:

- 950 1. The action shall be that inherited from the parent according to the rules of |  
951 inheritance of signal actions defined in the System Interfaces volume of |  
952 IEEE Std 1003.1-200x. |
- 953 2. When no action has been taken to change the default, the default action shall be |  
954 that specified by the System Interfaces volume of IEEE Std 1003.1-200x. |
- 955 3. The result of the utility’s execution is as if default actions had been taken.

956 A utility is permitted to catch a signal, perform some additional processing (such as  
957 deleting temporary files), restore the default signal action (or action inherited from the  
958 parent process), and resignal itself.

#### 959 STDOUT

960 The STDOUT section completely describes the standard output of the utility. This  
961 section is frequently merely a reference to the following section, OUTPUT FILES,  
962 because many utilities treat standard output and output files in the same manner. |

963 Use of a terminal for standard output may cause any of the standard utilities that write  
964 standard output to stop when used in the background. For this reason, applications  
965 should not use interactive features in scripts to be placed in the background.

966 Record formats are described in a notation similar to that used by the C-language  
967 function, *printf()*. See the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 5,  
968 File Format Notation for a description of this notation.

969 The specified standard output of the standard utilities shall not depend on the  
970 existence or value of the environment variables defined in this volume of  
971 IEEE Std 1003.1-200x, except as provided by this volume of IEEE Std 1003.1-200x.

972 Some of the standard utilities describe their output using the verb *display*, defined in  
973 the Base Definitions volume of IEEE Std 1003.1-200x, Section 3.132, Display. Output  
974 described in the STDOUT sections of such utilities may be produced using means other  
975 than standard output. When standard output is directed to a terminal, the output

976 described shall be written directly to the terminal. Otherwise, the results are undefined.

977 **Default Behavior:** When this section is listed as “Not used.”, it means that the  
 978 standard output shall not be written when the utility is used as described by this  
 979 volume of IEEE Std 1003.1-200x.

## 980 STDERR

981 The STDERR section describes the standard error output of the utility. Only those  
 982 messages that are purposely sent by the utility are described.

983 Use of a terminal for standard error may cause any of the standard utilities that write  
 984 standard error output to stop when used in the background. For this reason,  
 985 applications should not use interactive features in scripts to be placed in the  
 986 background.

987 The format of diagnostic messages for most utilities is unspecified, but the language  
 988 and cultural conventions of diagnostic and informative messages whose format is  
 989 unspecified by this volume of IEEE Std 1003.1-200x should be affected by the setting of  
 990 XSI *LC\_MESSAGES* and *NLSPATH*.

991 The specified standard error output of standard utilities shall not depend on the  
 992 existence or value of the environment variables defined in this volume of  
 993 IEEE Std 1003.1-200x, except as provided by this volume of IEEE Std 1003.1-200x.

994 **Default Behavior:** When this section is listed as “Used only for diagnostic messages.”,  
 995 it means that, unless otherwise stated, the diagnostic messages shall be sent to the  
 996 standard error only when the exit status is non-zero and the utility is used as described  
 997 by this volume of IEEE Std 1003.1-200x.

998 When this section is listed as “Not used.”, it means that the standard error shall not be  
 999 used when the utility is used as described in this volume of IEEE Std 1003.1-200x.

## 1000 OUTPUT FILES

1001 The OUTPUT FILES section completely describes the files created or modified by the  
 1002 utility. Temporary or system files that are created for internal usage by this utility or  
 1003 other parts of the implementation (for example, spool, log, and audit files) are not  
 1004 described in this, or any, section. The utilities creating such files and the names of such  
 1005 files are unspecified. If applications are written to use temporary or intermediate files,  
 1006 they should use the *TMPDIR* environment variable, if it is set and represents an  
 1007 accessible directory, to select the location of temporary files.

1008 Implementations shall ensure that temporary files, when used by the standard utilities,  
 1009 are named so that different utilities or multiple instances of the same utility can operate  
 1010 simultaneously without regard to their working directories, or any other process  
 1011 characteristic other than process ID. There are two exceptions to this rule:

- 1012 1. Resources for temporary files other than the name space (for example, disk space,  
 1013 available directory entries, or number of processes allowed) are not guaranteed.
- 1014 2. Certain standard utilities generate output files that are intended as input for other  
 1015 utilities (for example, *lex* generates *lex.yy.c*), and these cannot have unique  
 1016 names. These cases are explicitly identified in the descriptions of the respective  
 1017 utilities.

1018 Any temporary file created by the implementation shall be removed by the  
 1019 implementation upon a utility’s successful exit, exit because of errors, or before  
 1020 termination by any of the SIGHUP, SIGINT, or SIGTERM signals, unless specified  
 1021 otherwise by the utility description.

1022 Receipt of the SIGQUIT signal should generally cause termination (unless in some  
1023 debugging mode) that would bypass any attempted recovery actions.

1024 Record formats are described in a notation similar to that used by the C-language  
1025 function, *printf()*; see the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 5,  
1026 File Format Notation for a description of this notation.

1027 **Default Behavior:** When this section is listed as “None.”, it means that no files are  
1028 created or modified as a consequence of direct action on the part of the utility when the  
1029 utility is used as described by this volume of IEEE Std 1003.1-200x. However, the  
1030 utility may create or modify system files, such as log files, that are outside the utility’s  
1031 normal execution environment.

#### 1032 EXTENDED DESCRIPTION

1033 The EXTENDED DESCRIPTION section provides a place for describing the actions of  
1034 very complicated utilities, such as text editors or language processors, which typically  
1035 have elaborate command languages.

1036 **Default Behavior:** When this section is listed as “None.”, no further description is  
1037 necessary.

#### 1038 EXIT STATUS

1039 The EXIT STATUS section describes the values the utility shall return to the calling  
1040 program, or shell, and the conditions that cause these values to be returned. Usually,  
1041 utilities return zero for successful completion and values greater than zero for various  
1042 error conditions. If specific numeric values are listed in this section, the system shall  
1043 use those values for the errors described. In some cases, status values are listed more  
1044 loosely, such as >0. A strictly conforming application shall not rely on any specific  
1045 value in the range shown and shall be prepared to receive any value in the range.

1046 For example, a utility may list zero as a successful return, 1 as a failure for a specific  
1047 reason, and >1 as “an error occurred”. In this case, unspecified conditions may cause a  
1048 2 or 3, or other value, to be returned. A conforming application should be written so  
1049 that it tests for successful exit status values (zero in this case), rather than relying upon  
1050 the single specific error value listed in this volume of IEEE Std 1003.1-200x. In that  
1051 way, it has maximum portability, even on implementations with extensions.

1052 Unspecified error conditions may be represented by specific values not listed in this  
1053 volume of IEEE Std 1003.1-200x.

#### 1054 CONSEQUENCES OF ERRORS

1055 The CONSEQUENCES OF ERRORS section describes the effects on the environment,  
1056 file systems, process state, and so on, when error conditions occur. It does not describe  
1057 error messages produced or exit status values used.

1058 The many reasons for failure of a utility are generally not specified by the utility  
1059 descriptions. Utilities may terminate prematurely if they encounter: invalid usage of  
1060 options, arguments, or environment variables; invalid usage of the complex syntaxes  
1061 expressed in EXTENDED DESCRIPTION sections; difficulties accessing, creating,  
1062 reading, or writing files; or difficulties associated with the privileges of the process.

1063 The following shall apply to each utility, unless otherwise stated:

- 1064 • If the requested action cannot be performed on an operand representing a file,  
1065 directory, user, process, and so on, the utility shall issue a diagnostic message to  
1066 standard error and continue processing the next operand in sequence, but the final  
1067 exit status shall be returned as non-zero.

1068 For a utility that recursively traverses a file hierarchy (such as *find* or *chown -R*), if  
 1069 the requested action cannot be performed on a file or directory encountered in the  
 1070 hierarchy, the utility shall issue a diagnostic message to standard error and continue  
 1071 processing the remaining files in the hierarchy, but the final exit status shall be  
 1072 returned as non-zero.

1073 • If the requested action characterized by an option or option-argument cannot be  
 1074 performed, the utility shall issue a diagnostic message to standard error and the exit  
 1075 status returned shall be non-zero.

1076 • When an unrecoverable error condition is encountered, the utility shall exit with a  
 1077 non-zero exit status.

1078 • A diagnostic message shall be written to standard error whenever an error  
 1079 condition occurs.

1080 When a utility encounters an error condition several actions are possible, depending on  
 1081 the severity of the error and the state of the utility. Included in the possible actions of  
 1082 various utilities are: deletion of temporary or intermediate work files; deletion of  
 1083 incomplete files; validity checking of the file system or directory.

1084 **Default Behavior:** When this section is listed as “Default.”, it means that any changes  
 1085 to the environment are unspecified.

#### 1086 APPLICATION USAGE

1087 This section is non-normative.

1088 The APPLICATION USAGE section gives advice to the application programmer or user  
 1089 about the way the utility should be used.

#### 1090 EXAMPLES

1091 This section is non-normative.

1092 The EXAMPLES section gives one or more examples of usage, where appropriate. In  
 1093 the event of conflict between an example and a normative part of the specification, the  
 1094 normative material is to be taken as correct.

1095 In all examples, quoting has been used, showing how sample commands (utility names  
 1096 combined with arguments) could be passed correctly to a shell (see *sh*) or as a string to  
 1097 the *system()* function defined in the System Interfaces volume of IEEE Std 1003.1-200x.  
 1098 Such quoting would not be used if the utility is invoked using one of the *exec* functions  
 1099 defined in the System Interfaces volume of IEEE Std 1003.1-200x.

#### 1100 RATIONALE

1101 This section is non-normative.

1102 This section contains historical information concerning the contents of this volume of  
 1103 IEEE Std 1003.1-200x and why features were included or discarded by the standard  
 1104 developers.

#### 1105 FUTURE DIRECTIONS

1106 This section is non-normative.

1107 The FUTURE DIRECTIONS section should be used as a guide to current thinking; there  
 1108 is not necessarily a commitment to implement all of these future directions in their  
 1109 entirety.

#### 1110 SEE ALSO

1111 This section is non-normative.

1112                   The SEE ALSO section lists related entries.

1113                   **CHANGE HISTORY**

1114                   This section is non-normative.

1115                   The CHANGE HISTORY section shows the derivation of the description used by this  
1116                   volume of IEEE Std 1003.1-200x and lists the functional differences between Issues 4  
1117                   and 6.

1118                   Certain of the standard utilities describe how they can invoke other utilities or applications, such  
1119                   as by passing a command string to the command interpreter. The external influences (STDIN,  
1120                   ENVIRONMENT VARIABLES, and so on) and external effects (STDOUT, CONSEQUENCES OF  
1121                   ERRORS, and so on) of such invoked utilities are not described in the section concerning the  
1122                   standard utility that invokes them.

1123   **1.12 Considerations for Utilities in Support of Files of Arbitrary Size**

1124                   The following utilities support files of any size up to the maximum that can be created by the  
1125                   implementation. This support includes correct writing of file size-related values (such as file  
1126                   sizes and offsets, line numbers, and block counts) and correct interpretation of command line  
1127                   arguments that contain such values.

1128                   *basename*     Return non-directory portion of pathname.

1129                   *cat*             Concatenate and print files.

1130                   *cd*             Change working directory.

1131                   *chgrp*         Change file group ownership.

1132                   *chmod*         Change file modes.

1133                   *chown*         Change file ownership.

1134                   *cksum*         Write file checksums and sizes.

1135                   *cmp*            Compare two files.

1136                   *cp*             Copy files.

1137                   *dd*             Convert and copy a file.

1138                   *df*             Report free disk space.

1139                   *dirname*       Return directory portion of pathname.

1140                   *du*             Estimate file space usage.

1141                   *find*           Find files.

1142                   *ln*             Link files.

1143                   *ls*             List directory contents.

1144                   *mkdir*         Make directories.

1145                   *mv*             Move files.

1146                   *pathchk*       Check pathnames.

1147                   *pwd*            Return working directory name.

1148	<i>rm</i>	Remove directory entries.
1149	<i>rmdir</i>	Remove directories.
1150	<i>sh</i>	Shell, the standard command language interpreter.
1151	<i>sum</i>	Print checksum and block or byte count of a file.
1152	<i>test</i>	Evaluate expression.
1153	<i>touch</i>	Change file access and modification times.
1154	<i>ulimit</i>	Set or report file size limit.
1155	Exceptions to the requirement that utilities support files of any size up to the maximum are as follows:	
1156		
1157	1.	Uses of files as command scripts, or for configuration or control, are exempt. For example, it is not required that <i>sh</i> be able to read an arbitrarily large <b>.profile</b> .
1158		
1159	2.	Shell input and output redirection are exempt. For example, it is not required that the redirections <i>sum</i> < <i>file</i> or <i>echo foo</i> > <i>file</i> succeed for an arbitrarily large existing file.
1160		

### 1161 1.13 Built-In Utilities

1162 Any of the standard utilities may be implemented as *regular built-in* utilities within the  
 1163 command language interpreter. This is usually done to increase the performance of frequently  
 1164 used utilities or to achieve functionality that would be more difficult in a separate environment.  
 1165 The utilities named in Table 1-5 are frequently provided in built-in form. All of the utilities  
 1166 named in the table have special properties in terms of command search order within the shell, as  
 1167 described in Section 2.9.1.1 (on page 2249).

1168 **Table 1-5** Regular Built-in Utilities

1169	<i>alias</i>	<i>false</i>	<i>jobs</i>	<i>true</i>
1170	<i>bg</i>	<i>fc</i>	<i>kill</i>	<i>umask</i>
1171	<i>cd</i>	<i>fg</i>	<i>newgrp</i>	<i>unalias</i>
1172	<i>command</i>	<i>getopts</i>	<i>read</i>	<i>wait</i>

1173 However, all of the standard utilities, including the regular built-ins in the table, but not the  
 1174 special built-ins described in Section 2.14 (on page 2266), shall be implemented in a manner so  
 1175 that they can be accessed via the *exec* family of functions as defined in the System Interfaces  
 1176 volume of IEEE Std 1003.1-200x and can be invoked directly by those standard utilities that  
 1177 require it (*env*, *find*, *nice*, *nohup*, *time*, *xargs*).

1178 Since *exec*-able versions shall be provided for all utilities except for those listed in Section 2.14  
 1179 (on page 2266), an application running on a system that conforms to IEEE Std 1003.1-200x can  
 1180 use the *exec* family of functions, in addition to the shell command interface provided by the  
 1181 *system()* and *popen()* functions, to execute any of these utilities.



1183

1184 This chapter contains the definition of the Shell Command Language.

## 1185 **2.1 Shell Introduction**

1186 The shell is a command language interpreter. This chapter describes the syntax of that command  
1187 language as it is used by the *sh* utility and the *system()* and *popen()* functions defined in the  
1188 System Interfaces volume of IEEE Std 1003.1-200x.

1189 The shell operates according to the following general overview of operations. The specific  
1190 details are included in the cited sections of this chapter.

- 1191 1. The shell reads its input from a file (see *sh*), from the `-c` option or from the *system()* and  
1192 *popen()* functions defined in the System Interfaces volume of IEEE Std 1003.1-200x. If the  
1193 first line of a file of shell commands starts with the characters "#!", the results are  
1194 unspecified.
- 1195 2. The shell breaks the input into tokens: words and operators; see Section 2.3 (on page 2233).
- 1196 3. The shell parses the input into simple commands (see Section 2.9.1 (on page 2248)) and  
1197 compound commands (see Section 2.9.4 (on page 2253)).
- 1198 4. The shell performs various expansions (separately) on different parts of each command,  
1199 resulting in a list of pathnames and fields to be treated as a command and arguments; see  
1200 Section 2.6 (on page 2238).
- 1201 5. The shell performs redirection (see Section 2.7 (on page 2244)) and removes redirection  
1202 operators and their operands from the parameter list.
- 1203 6. The shell executes a function (see Section 2.9.5 (on page 2256)), built-in (see Section 2.14  
1204 (on page 2266)), executable file, or script, giving the names of the arguments as positional  
1205 parameters numbered 1 to *n*, and the name of the command (or in the case of a function  
1206 within a script, the name of the script) as the positional parameter numbered 0 (see Section  
1207 2.9.1.1 (on page 2249)).
- 1208 7. The shell optionally waits for the command to complete and collects the exit status (see  
1209 Section 2.8.2 (on page 2248)).

## 1210 2.2 Quoting

1211 Quoting is used to remove the special meaning of certain characters or words to the shell.  
 1212 Quoting can be used to preserve the literal meaning of the special characters in the next  
 1213 paragraph, prevent reserved words from being recognized as such, and prevent parameter  
 1214 expansion and command substitution within here-document processing (see Section 2.7.4 (on  
 1215 page 2246)).

1216 The application shall quote the following characters if they are to represent themselves:

1217 | & ; < > ( ) \$ ' \ " ' <space> <tab> <newline> |

1218 and the following may need to be quoted under certain circumstances. That is, these characters  
 1219 may be special depending on conditions described elsewhere in this volume of  
 1220 IEEE Std 1003.1-200x:

1221 \* ? [ # ~ = % |

1222 The various quoting mechanisms are the escape character, single-quotes, and double-quotes.  
 1223 The here-document represents another form of quoting; see Section 2.7.4 (on page 2246).

### 1224 2.2.1 Escape Character (Backslash)

1225 A backslash that is not quoted shall preserve the literal value of the following character, with the  
 1226 exception of a <newline>. If a <newline> follows the backslash, the shell shall interpret this as  
 1227 line continuation. The backslash and <newline>s shall be removed before splitting the input into  
 1228 tokens. Since the escaped <newline> is removed entirely from the input and is not replaced by  
 1229 any white space, it cannot serve as a token separator.

### 1230 2.2.2 Single-Quotes

1231 Enclosing characters in single-quotes ( ' ' ) shall preserve the literal value of each character  
 1232 within the single-quotes. A single-quote cannot occur within single-quotes.

### 1233 2.2.3 Double-Quotes

1234 Enclosing characters in double-quotes ( " " ) shall preserve the literal value of all characters  
 1235 within the double-quotes, with the exception of the characters dollar sign, backquote, and  
 1236 backslash, as follows:

1237 \$ The dollar sign shall retain its special meaning introducing parameter expansion (see  
 1238 Section 2.6.2 (on page 2239)), a form of command substitution (see Section 2.6.3 (on page  
 1239 2242)), and arithmetic expansion (see Section 2.6.4 (on page 2243)).

1240 The input characters within the quoted string that are also enclosed between "\$(" and the  
 1241 matching ')' is not affected by the double-quotes, but rather shall define that command  
 1242 whose output replaces the "\$(...)" when the word is expanded. The tokenizing rules in  
 1243 Section 2.3 (on page 2233), not including the alias substitutions in Section 2.3.1 (on page  
 1244 2234), shall be applied recursively to find the matching ')'. |

1245 Within the string of characters from an enclosed "\${" to the matching '}', an even number  
 1246 of unescaped double-quotes or single-quotes, if any, shall occur. A preceding backslash  
 1247 character shall be used to escape a literal '{' or '}'. The rule in Section 2.6.2 (on page  
 1248 2239) shall be used to determine the matching '}'.

1249 ` The backquote shall retain its special meaning introducing the other form of command  
 1250 substitution (see Section 2.6.3 (on page 2242)). The portion of the quoted string from the  
 1251 initial backquote and the characters up to the next backquote that is not preceded by a

1252           backslash, having escape characters removed, defines that command whose output replaces  
 1253           "`\ . . \`" when the word is expanded. Either of the following cases produces undefined  
 1254           results:

- 1255           • A single-quoted or double-quoted string that begins, but does not end, within the  
 1256           "`\ . . \`" sequence
- 1257           • A "`\ . . \`" sequence that begins, but does not end, within the same double-quoted  
 1258           string

1259           \  
 1260           The backslash shall retain its special meaning as an escape character (see Section 2.2.1 (on  
 page 2232)) only when followed by one of the following characters when considered special:

1261           \$   \   "   \  
                   <newline>

1262           The application shall ensure that a double-quote is preceded by a backslash to be included  
 1263           within double-quotes. The parameter '@' has special meaning inside double-quotes and is  
 1264           described in Section 2.5.2 (on page 2235).

## 1265 2.3 Token Recognition

1266           The shell shall read its input in terms of lines from a file, from a terminal in the case of an  
 1267           interactive shell, or from a string in the case of `sh -c` or `system()`. The input lines can be of  
 1268           unlimited length. These lines shall be parsed using two major modes: ordinary token recognition  
 1269           and processing of here-documents.

1270           When an **io\_here** token has been recognized by the grammar (see Section 2.10 (on page 2257)),  
 1271           one or more of the subsequent lines immediately following the next **NEWLINE** token form the  
 1272           body of one or more here-documents and shall be parsed according to the rules of Section 2.7.4  
 1273           (on page 2246).

1274           When it is not processing an **io\_here**, the shell shall break its input into tokens by applying the  
 1275           first applicable rule below to the next character in its input. The token shall be from the current  
 1276           position in the input until a token is delimited according to one of the rules below; the characters  
 1277           forming the token are exactly those in the input, including any quoting characters. If it is  
 1278           indicated that a token is delimited, and no characters have been included in a token, processing  
 1279           shall continue until an actual token is delimited.

- 1280           1. If the end of input is recognized, the current token shall be delimited. If there is no current  
 1281           token, the end-of-input indicator shall be returned as the token.
- 1282           2. If the previous character was used as part of an operator and the current character is not  
 1283           quoted and can be used with the current characters to form an operator, it shall be used as  
 1284           part of that (operator) token.
- 1285           3. If the previous character was used as part of an operator and the current character cannot  
 1286           be used with the current characters to form an operator, the operator containing the  
 1287           previous character shall be delimited.
- 1288           4. If the current character is backslash, single-quote, or double-quote (`'\'`, `'\''`, or `'\"'`)  
 1289           and it is not quoted, it shall affect quoting for subsequent characters up to the end of the  
 1290           quoted text. The rules for quoting are as described in Section 2.2 (on page 2232). During  
 1291           token recognition no substitutions shall be actually performed, and the result token shall  
 1292           contain exactly the characters that appear in the input (except for <newline> joining),  
 1293           unmodified, including any embedded or enclosing quotes or substitution operators,  
 1294           between the quote mark and the end of the quoted text. The token shall not be delimited by  
 1295           the end of the quoted field.

- 1296 5. If the current character is an unquoted '\$' or '`', the shell shall identify the start of any  
 1297 candidates for parameter expansion (Section 2.6.2 (on page 2239)), command substitution  
 1298 (Section 2.6.3 (on page 2242)), or arithmetic expansion (Section 2.6.4 (on page 2243)) from  
 1299 their introductory unquoted character sequences: '\$' or "\${", "\$(" or ``, and "\$(",  
 1300 respectively. The shell shall read sufficient input to determine the end of the unit to be  
 1301 expanded (as explained in the cited sections). While processing the characters, if instances  
 1302 of expansions or quoting are found nested within the substitution, the shell shall  
 1303 recursively process them in the manner specified for the construct that is found. The  
 1304 characters found from the beginning of the substitution to its end, allowing for any  
 1305 recursion necessary to recognize embedded constructs, shall be included unmodified in the  
 1306 result token, including any embedded or enclosing substitution operators or quotes. The  
 1307 token shall not be delimited by the end of the substitution.
- 1308 6. If the current character is not quoted and can be used as the first character of a new  
 1309 operator, the current token (if any) shall be delimited. The current character shall be used  
 1310 as the beginning of the next (operator) token.
- 1311 7. If the current character is an unquoted <newline>, the current token shall be delimited.
- 1312 8. If the current character is an unquoted <blank>, any token containing the previous  
 1313 character is delimited and the current character shall be discarded.
- 1314 9. If the previous character was part of a word, the current character shall be appended to  
 1315 that word.
- 1316 10. If the current character is a '#', it and all subsequent characters up to, but excluding, the  
 1317 next <newline> shall be discarded as a comment. The <newline> that ends the line is not  
 1318 considered part of the comment.
- 1319 11. The current character is used as the start of a new word.

1320 Once a token is delimited, it is categorized as required by the grammar in Section 2.10 (on page  
 1321 2257).

### 1322 2.3.1 Alias Substitution

1323 UP XSI The processing of aliases shall be supported on all XSI-conformant systems or if the system  
 1324 supports the User Portability Utilities option (and the rest of this section is not further shaded for  
 1325 these options).

1326 After a token has been delimited, but before applying the grammatical rules in Section 2.10 (on  
 1327 page 2257), a resulting word that is identified to be the command name word of a simple  
 1328 command shall be examined to determine whether it is an unquoted, valid alias name. However,  
 1329 reserved words in correct grammatical context shall not be candidates for alias substitution. A  
 1330 valid alias name (see the Base Definitions volume of IEEE Std 1003.1-200x, Section 3.10, Alias  
 1331 Name) shall be one that has been defined by the *alias* utility and not subsequently undefined  
 1332 using *unalias*. Implementations also may provide predefined valid aliases that are in effect when  
 1333 the shell is invoked. To prevent infinite loops in recursive aliasing, if the shell is not currently  
 1334 processing an alias of the same name, the word shall be replaced by the value of the alias;  
 1335 otherwise, it shall not be replaced.

1336 If the value of the alias replacing the word ends in a <blank>, the shell shall check the next  
 1337 command word for alias substitution; this process shall continue until a word is found that is not  
 1338 a valid alias or an alias value does not end in a <blank>.

1339 When used as specified by this volume of IEEE Std 1003.1-200x, alias definitions shall not be  
 1340 inherited by separate invocations of the shell or by the utility execution environments invoked  
 1341 by the shell; see Section 2.12 (on page 2263).

## 1342 2.4 Reserved Words

1343 Reserved words are words that have special meaning to the shell; see Section 2.9 (on page 2248).  
 1344 The following words shall be recognized as reserved words:

1345	!	do	esac	in	
1346	{	done	fi	then	
1347	}	elif	for	until	
1348	case	else	if	while	

1349 This recognition shall only occur when none of the characters is quoted and when the word is  
 1350 used as:

- 1351 • The first word of a command
- 1352 • The first word following one of the reserved words other than **case**, **for**, or **in**
- 1353 • The third word in a **case** or **for** command (only **in** is valid in this case)

1354 See the grammar in Section 2.10 (on page 2257).

1355 The following words may be recognized as reserved words on some implementations (when  
 1356 none of the characters are quoted), causing unspecified results:

1357	[[	]]	function	select	
------	----	----	----------	--------	--

1358 Words that are the concatenation of a name and a colon (':') are reserved; their use produces  
 1359 unspecified results.

## 1360 2.5 Parameters and Variables

1361 A parameter can be denoted by a name, a number, or one of the special characters listed in  
 1362 Section 2.5.2. A variable is a parameter denoted by a name.

1363 A parameter is set if it has an assigned value (null is a valid value). Once a variable is set, it can  
 1364 only be unset by using the *unset* special built-in command.

### 1365 2.5.1 Positional Parameters

1366 A positional parameter is a parameter denoted by the decimal value represented by one or more  
 1367 digits, other than the single digit 0. The digits denoting the positional parameters shall always be  
 1368 interpreted as a decimal value, even if there is a leading zero. When a positional parameter with  
 1369 more than one digit is specified, the application shall enclose the digits in braces (see Section  
 1370 2.6.2 (on page 2239)). Positional parameters are initially assigned when the shell is invoked (see  
 1371 *sh*), temporarily replaced when a shell function is invoked (see Section 2.9.5 (on page 2256)), and  
 1372 can be reassigned with the *set* special built-in command.

### 1373 2.5.2 Special Parameters

1374 Listed below are the special parameters and the values to which they shall expand. Only the  
 1375 values of the special parameters are listed; see Section 2.6 (on page 2238) for a detailed summary  
 1376 of all the stages involved in expanding words.

1377 @ Expands to the positional parameters, starting from one. When the expansion occurs within  
 1378 double-quotes, and where field splitting (see Section 2.6.5 (on page 2243)) is performed,  
 1379 each positional parameter shall expand as a separate field, with the provision that the  
 1380 expansion of the first parameter shall still be joined with the beginning part of the original  
 1381 word (assuming that the expanded parameter was embedded within a word), and the

- 1382 expansion of the last parameter shall still be joined with the last part of the original word. If |  
 1383 there are no positional parameters, the expansion of '@' shall generate zero fields, even |  
 1384 when '@' is double-quoted. |
- 1385 \* Expands to the positional parameters, starting from one. When the expansion occurs within |  
 1386 a double-quoted string (see Section 2.2.3 (on page 2232)), it shall expand to a single field |  
 1387 with the value of each parameter separated by the first character of the *IFS* variable, or by a |  
 1388 <space> if *IFS* is unset. If *IFS* is set to a null string, this is not equivalent to unsetting it; its |  
 1389 first character does not exist, so the parameter values are concatenated.
- 1390 # Expands to the decimal number of positional parameters. The command name (parameter |  
 1391 0) shall not be counted in the number given by '#' because it is a special parameter, not a |  
 1392 positional parameter.
- 1393 ? Expands to the decimal exit status of the most recent pipeline (see Section 2.9.2 (on page |  
 1394 2250)).
- 1395 – (Hyphen.) Expands to the current option flags (the single-letter option names concatenated |  
 1396 into a string) as specified on invocation by the *set* special built-in command or implicitly by |  
 1397 the shell.
- 1398 \$ Expands to the decimal process ID of the invoked shell. In a subshell (see Section 2.12 (on |  
 1399 page 2263)), '\$' shall expand to the same value as that of the current shell.
- 1400 ! Expands to the decimal process ID of the most recent background command (see Section |  
 1401 2.9.3 (on page 2251)) executed from the current shell. (For example, background commands |  
 1402 executed from subshells do not affect the value of "\$!" in the current shell environment.) |  
 1403 For a pipeline, the process ID is that of the last command in the pipeline.
- 1404 0 (Zero.) Expands to the name of the shell or shell script. See *sh* (on page 3048) for a detailed |  
 1405 description of how this name is derived.
- 1406 See the description of the *IFS* variable in Section 2.5.3.

### 1407 2.5.3 Shell Variables

1408 Variables shall be initialized from the environment (as defined by the Base Definitions volume of |  
 1409 IEEE Std 1003.1-200x, Chapter 8, Environment Variables and the *exec* function in the System |  
 1410 Interfaces volume of IEEE Std 1003.1-200x) and can be given new values with variable |  
 1411 assignment commands. If a variable is initialized from the environment, it shall be marked for |  
 1412 export immediately; see the *export* special built-in. New variables can be defined and initialized |  
 1413 with variable assignments, with the *read* or *getopts* utilities, with the *name* parameter in a *for* |  
 1414 loop, with the  $\${name=word}$  expansion, or with other mechanisms provided as implementation |  
 1415 extensions.

1416 The following variables shall affect the execution of the shell.

1417 UP XSI **ENV** The processing of the *ENV* shell variable shall be supported on all XSI- |  
 1418 conformant systems or if the system supports the User Portability Utilities |  
 1419 option. |

1420 This variable, when and only when an interactive shell is invoked, shall be |  
 1421 subjected to parameter expansion (see Section 2.6.2 (on page 2239)) by the |  
 1422 shell and the resulting value shall be used as a pathname of a file containing |  
 1423 shell commands to execute in the current environment. The file need not be |  
 1424 executable. If the expanded value of *ENV* is not an absolute pathname, the |  
 1425 results are unspecified. *ENV* shall be ignored if the user's real and effective |  
 1426 user IDs or real and effective group IDs are different. |

1427	<i>HOME</i>	The pathname of the user's home directory. The contents of <i>HOME</i> are used in tilde expansion (see Section 2.6.1 (on page 2239)).
1428		
1429	<i>IFS</i>	(Input Field Separators.) A string treated as a list of characters that is used for field splitting and to split lines into fields with the <i>read</i> command. If <i>IFS</i> is not set, the shell shall behave as if the value of <i>IFS</i> is <space>, <tab>, and <newline>; see Section 2.6.5 (on page 2243).
1430		
1431		
1432		
1433	<i>LANG</i>	The default value for the internationalization variables that are unset or null. If <i>LANG</i> is unset or null, the corresponding value from the implementation-defined default locale is used. If any of the internationalization variables contains an invalid setting, the utility behaves as if none of the variables had been defined.
1434		
1435		
1436		
1437		
1438	<i>LC_ALL</i>	The default value for the <i>LC_*</i> variables, as described in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 8, Environment Variables.
1439		
1440	<i>LC_COLLATE</i>	Determine the behavior of range expressions, equivalence classes, and multi-character collating elements within pattern matching.
1441		
1442	<i>LC_CTYPE</i>	Determine the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters), which characters are defined as letters (character class <b>alpha</b> ) and <blank>s (character class <b>blank</b> ), and the behavior of character classes within pattern matching. Changing the value of <i>LC_CTYPE</i> after the shell has started shall not affect the lexical processing of shell commands in the current shell execution environment or its subshells. Invoking a shell script or performing <i>exec sh</i> subjects the new shell to the changes in <i>LC_CTYPE</i> .
1443		
1444		
1445		
1446		
1447		
1448		
1449		
1450	<i>LC_MESSAGES</i>	Determine the language in which messages should be written.
1451	<i>LINENO</i>	Set by the shell to a decimal number representing the current sequential line number (numbered starting with 1) within a script or function before it executes each command. If the user unsets or resets <i>LINENO</i> , the variable may lose its special meaning for the life of the shell. If the shell is not currently executing a script or function, the value of <i>LINENO</i> is unspecified. This volume of IEEE Std 1003.1-200x specifies the effects of the variable only for systems supporting the User Portability Utilities option.
1452		
1453		
1454		
1455		
1456		
1457		
1458	XSI <i>NLSPATH</i>	Determine the location of message catalogs for the processing of <i>LC_MESSAGES</i> .
1459		
1460	<i>PATH</i>	A string formatted as described in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 8, Environment Variables, used to effect command interpretation; see Section 2.9.1.1 (on page 2249).
1461		
1462		
1463	<i>PPID</i>	Set by the shell to the decimal process ID of the process that invoked this shell. In a subshell (see Section 2.12 (on page 2263)), <i>PPID</i> shall be set to the same value as that of the parent of the current shell. For example, <i>echo\$PPID</i> and ( <i>echo\$PPID</i> ) would produce the same value. This volume of IEEE Std 1003.1-200x specifies the effects of the variable only for systems supporting the User Portability Utilities option.
1464		
1465		
1466		
1467		
1468		
1469	<i>PS1</i>	Each time an interactive shell is ready to read a command, the value of this variable shall be subjected to parameter expansion and written to standard error. The default value shall be "\$ ". For users who have specific additional implementation-defined privileges, the default may be another, implementation-defined value. The shell shall replace each instance of the
1470		
1471		
1472		
1473		

1474		character '!' in <i>PS1</i> with the history file number of the next command to be
1475		typed. Escaping the '!' with another '!' (that is, "!!") shall place the literal
1476		character '!' in the prompt. This volume of IEEE Std 1003.1-200x specifies
1477		the effects of the variable only for systems supporting the User Portability
1478		Utilities option.
1479	<i>PS2</i>	Each time the user enters a <newline> prior to completing a command line in
1480		an interactive shell, the value of this variable shall be subjected to parameter
1481		expansion and written to standard error. The default value is "> ". This
1482		volume of IEEE Std 1003.1-200x specifies the effects of the variable only for
1483		systems supporting the User Portability Utilities option.
1484	<i>PS4</i>	When an execution trace ( <i>set -x</i> ) is being performed in an interactive shell,
1485		before each line in the execution trace, the value of this variable shall be
1486		subjected to parameter expansion and written to standard error. The default
1487		value is "+ ". This volume of IEEE Std 1003.1-200x specifies the effects of the
1488		variable only for systems supporting the User Portability Utilities option.
1489	<i>PWD</i>	Set by the shell to be an absolute pathname of the current working directory,
1490		containing no components of type symbolic link, no components that are dot,
1491		and no components that are dot-dot when the shell is initialized. If an
1492		application sets or unsets the value of <i>PWD</i> , the behaviors of the <i>cd</i> and <i>pwd</i>
1493		utilities are unspecified.

## 1494 2.6 Word Expansions

1495 This section describes the various expansions that are performed on words. Not all expansions  
1496 are performed on every word, as explained in the following sections.

1497 Tilde expansions, parameter expansions, command substitutions, arithmetic expansions, and  
1498 quote removals that occur within a single word expand to a single field. It is only field splitting  
1499 or pathname expansion that can create multiple fields from a single word. The single exception  
1500 to this rule is the expansion of the special parameter '@' within double-quotes, as described in  
1501 Section 2.5.2 (on page 2235).

1502 The order of word expansion shall be as follows:

- 1503 1. Tilde expansion (see Section 2.6.1 (on page 2239)), parameter expansion (see Section 2.6.2  
1504 (on page 2239)), command substitution (see Section 2.6.3 (on page 2242)), and arithmetic  
1505 expansion (see Section 2.6.4 (on page 2243)) shall be performed, beginning to end. See item  
1506 5 in Section 2.3 (on page 2233).
- 1507 2. Field splitting (see Section 2.6.5 (on page 2243)) shall be performed on the portions of the  
1508 fields generated by step 1, unless *IFS* is null.
- 1509 3. Pathname expansion (see Section 2.6.6 (on page 2244)) shall be performed, unless *set -f* is  
1510 in effect.
- 1511 4. Quote removal (see Section 2.6.7 (on page 2244)) shall always be performed last.

1512 The expansions described in this section shall occur in the same shell environment as that in  
1513 which the command is executed.

1514 If the complete expansion appropriate for a word results in an empty field, that empty field shall  
1515 be deleted from the list of fields that form the completely expanded command, unless the  
1516 original word contained single-quote or double-quote characters.

1517 The '\$' character is used to introduce parameter expansion, command substitution, or  
 1518 arithmetic evaluation. If an unquoted '\$' is followed by a character that is either not numeric,  
 1519 the name of one of the special parameters (see Section 2.5.2 (on page 2235)), a valid first  
 1520 character of a variable name, a left curly brace ('{') or a left parenthesis, the result is  
 1521 unspecified.

### 1522 2.6.1 Tilde Expansion

1523 A *tilde-prefix* consists of an unquoted tilde character at the beginning of a word, followed by all  
 1524 of the characters preceding the first unquoted slash in the word, or all the characters in the word  
 1525 if there is no slash. In an assignment (see the Base Definitions volume of IEEE Std 1003.1-200x, |  
 1526 Section 4.21, Variable Assignment), multiple tilde-prefixes can be used: at the beginning of the |  
 1527 word (that is, following the equal sign of the assignment), following any unquoted colon, or  
 1528 both. A tilde-prefix in an assignment is terminated by the first unquoted colon or slash. If none  
 1529 of the characters in the tilde-prefix are quoted, the characters in the tilde-prefix following the  
 1530 tilde are treated as a possible login name from the user database. A portable login name cannot  
 1531 contain characters outside the set given in the description of the *LOGNAME* environment  
 1532 variable in the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.3, Other Environment  
 1533 Variables. If the login name is null (that is, the tilde-prefix contains only the tilde), the tilde-  
 1534 prefix is replaced by the value of the variable *HOME*. If *HOME* is unset, the results are |  
 1535 unspecified. Otherwise, the tilde-prefix shall be replaced by a pathname of the initial working |  
 1536 directory associated with the login name obtained using the *getpwnam()* function as defined in |  
 1537 the System Interfaces volume of IEEE Std 1003.1-200x. If the system does not recognize the login  
 1538 name, the results are undefined.

### 1539 2.6.2 Parameter Expansion

1540 The format for parameter expansion is as follows:

1541  $\${expression}$  |

1542 where *expression* consists of all characters until the matching '}'. Any '}' escaped by a  
 1543 backslash or within a quoted string, and characters in embedded arithmetic expansions,  
 1544 command substitutions, and variable expansions, shall not be examined in determining the  
 1545 matching '}'.

1546 The simplest form for parameter expansion is:

1547  $\${parameter}$  |

1548 The value, if any, of *parameter* shall be substituted.

1549 The parameter name or symbol can be enclosed in braces, which are optional except for  
 1550 positional parameters with more than one digit or when *parameter* is followed by a character that  
 1551 could be interpreted as part of the name. The matching closing brace shall be determined by  
 1552 counting brace levels, skipping over enclosed quoted strings, and command substitutions.

1553 If the parameter name or symbol is not enclosed in braces, the expansion shall use the longest  
 1554 valid name (see the Base Definitions volume of IEEE Std 1003.1-200x, Section 3.230, Name),  
 1555 whether or not the symbol represented by that name exists.

1556 If a parameter expansion occurs inside double-quotes:

- 1557 • Pathname expansion shall not be performed on the results of the expansion.
- 1558 • Field splitting shall not be performed on the results of the expansion, with the exception of  
 1559 '@'; see Section 2.5.2 (on page 2235).

1560 In addition, a parameter expansion can be modified by using one of the following formats. In  
 1561 each case that a value of *word* is needed (based on the state of *parameter*, as described below),  
 1562 *word* shall be subjected to tilde expansion, parameter expansion, command substitution, and  
 1563 arithmetic expansion. If *word* is not needed, it shall not be expanded. The '}' character that  
 1564 delimits the following parameter expansion modifications shall be determined as described  
 1565 previously in this section and in Section 2.2.3 (on page 2232). (For example, `${foo-bar}xyz`  
 1566 would result in the expansion of `foo` followed by the string `xyz` if `foo` is set, else the string  
 1567 `"barxyz"`).

1568 `${parameter:-word}` **Use Default Values.** If *parameter* is unset or null, the expansion of *word*  
 1569 shall be substituted; otherwise, the value of *parameter* shall be substituted.

1570 `${parameter:=word}` **Assign Default Values.** If *parameter* is unset or null, the expansion of  
 1571 *word* shall be assigned to *parameter*. In all cases, the final value of  
 1572 *parameter* shall be substituted. Only variables, not positional parameters  
 1573 or special parameters, can be assigned in this way.

1574 `${parameter:?[word]}` **Indicate Error if Null or Unset.** If *parameter* is unset or null, the  
 1575 expansion of *word* (or a message indicating it is unset if *word* is omitted)  
 1576 shall be written to standard error and the shell exits with a non-zero exit  
 1577 status. Otherwise, the value of *parameter* shall be substituted. An  
 1578 interactive shell need not exit.

1579 `${parameter:+word}` **Use Alternative Value.** If *parameter* is unset or null, null shall be  
 1580 substituted; otherwise, the expansion of *word* shall be substituted.

1581 In the parameter expansions shown previously, use of the colon in the format shall result in a  
 1582 test for a parameter that is unset or null; omission of the colon shall result in a test for a  
 1583 parameter that is only unset. The following table summarizes the effect of the colon:

	<i>parameter</i> Set and Not Null	<i>parameter</i> Set But Null	<i>parameter</i> Unset
<code>\${parameter:-word}</code>	substitute <i>parameter</i>	substitute <i>word</i>	substitute <i>word</i>
<code>\${parameter-word}</code>	substitute <i>parameter</i>	substitute null	substitute <i>word</i>
<code>\${parameter:=word}</code>	substitute <i>parameter</i>	assign <i>word</i>	assign <i>word</i>
<code>\${parameter=word}</code>	substitute <i>parameter</i>	substitute <i>parameter</i>	assign null
<code>\${parameter:?word}</code>	substitute <i>parameter</i>	error, exit	error, exit
<code>\${parameter?word}</code>	substitute <i>parameter</i>	substitute null	error, exit
<code>\${parameter:+word}</code>	substitute <i>word</i>	substitute null	substitute null
<code>\${parameter+word}</code>	substitute <i>word</i>	substitute <i>word</i>	substitute null

1594 In all cases shown with “substitute”, the expression is replaced with the value shown. In all  
 1595 cases shown with “assign”, *parameter* is assigned that value, which also replaces the expression.

1596 `${#parameter}` **String Length.** The length in characters of the value of *parameter* shall be  
 1597 substituted. If *parameter* is '\*' or '@', the result of the expansion is  
 1598 unspecified.

1599 The following four varieties of parameter expansion provide for substring processing. In each  
 1600 case, pattern matching notation (see Section 2.13 (on page 2264)), rather than regular expression  
 1601 notation, shall be used to evaluate the patterns. If *parameter* is '\*' or '@', the result of the  
 1602 expansion is unspecified. Enclosing the full parameter expansion string in double-quotes shall  
 1603 not cause the following four varieties of pattern characters to be quoted, whereas quoting  
 1604 characters within the braces shall have this effect.

1605 `${parameter%word}` **Remove Smallest Suffix Pattern.** The *word* shall be expanded to produce  
 1606 a pattern. The parameter expansion shall then result in *parameter*, with the

1607 smallest portion of the suffix matched by the *pattern* deleted.

1608 `${parameter%%word}` **Remove Largest Suffix Pattern.** The *word* shall be expanded to produce a |  
 1609 pattern. The parameter expansion shall then result in *parameter*, with the |  
 1610 largest portion of the suffix matched by the *pattern* deleted.

1611 `${parameter#word}` **Remove Smallest Prefix Pattern.** The *word* shall be expanded to produce |  
 1612 a pattern. The parameter expansion shall then result in *parameter*, with the |  
 1613 smallest portion of the prefix matched by the *pattern* deleted.

1614 `${parameter##word}` **Remove Largest Prefix Pattern.** The *word* shall be expanded to produce a |  
 1615 pattern. The parameter expansion shall then result in *parameter*, with the |  
 1616 largest portion of the prefix matched by the *pattern* deleted.

1617 **Examples**

1618 `${parameter:-word}`  
 1619 In this example, *ls* is executed only if *x* is null or unset. (The `$(ls)` command substitution  
 1620 notation is explained in Section 2.6.3 (on page 2242).)

1621  `${x:-$(ls)}`  |

1622 `${parameter:=word}`  
 1623 unset X  
 1624 echo \${X:=abc}  
 1625 **abc**

1626 `${parameter:?word}`  
 1627 unset posix  
 1628 echo \${posix:?}  
 1629 **sh: posix: parameter null or not set**

1630 `${parameter:+word}`  
 1631 set a b c  
 1632 echo \${3:+posix}  
 1633 **posix**

1634 `${#parameter}`  
 1635 HOME=/usr/posix  
 1636 echo \${#HOME}  
 1637 **10**

1638 `${parameter%word}`  
 1639 x=file.c  
 1640 echo \${x%.c}.o  
 1641 **file.o**

1642 `${parameter%%word}`  
 1643 x=posix/src/std  
 1644 echo \${x%%/\*}  
 1645 **posix**

1646 `${parameter#word}`  
 1647 x=\$HOME/src/cmd  
 1648 echo \${x#\$HOME}  
 1649 **/src/cmd**

1650 `${parameter##word}`  
 1651 x=/one/two/three

1652           echo \${x##\*/}  
1653           three

1654           The double-quoting of patterns is different depending on where the double-quotes are placed:

1655           "\${x#\*}"    The asterisk is a pattern character.

1656           "\${x#" \* }"   The literal asterisk is quoted and not special.

### 1657 2.6.3 Command Substitution

1658           Command substitution allows the output of a command to be substituted in place of the  
1659           command name itself. Command substitution shall occur when the command is enclosed as  
1660           follows:

1661           \$(*command*)

1662           or (backquoted version):

1663           `*command*`

1664           The shell shall expand the command substitution by executing *command* in a subshell  
1665           environment (see Section 2.12 (on page 2263)) and replacing the command substitution (the text  
1666           of *command* plus the enclosing "\$()" or backquotes) with the standard output of the command,  
1667           removing sequences of one or more <newline>s at the end of the substitution. Embedded  
1668           <newline>s before the end of the output shall not be removed; however, they may be treated as  
1669           field delimiters and eliminated during field splitting, depending on the value of *IFS* and quoting  
1670           that is in effect.

1671           Within the backquoted style of command substitution, backslash shall retain its literal meaning,  
1672           except when followed by: '\$', '`', or '\\\' (dollar sign, backquote, backslash). The search for  
1673           the matching backquote shall be satisfied by the first backquote found without a preceding  
1674           backslash; during this search, if a non-escaped backquote is encountered within a shell  
1675           comment, a here-document, an embedded command substitution of the \$(*command*) form, or a  
1676           quoted string, undefined results occur. A single-quoted or double-quoted string that begins, but  
1677           does not end, within the "`...`" sequence produces undefined results.

1678           With the \$(*command*) form, all characters following the open parenthesis to the matching closing  
1679           parenthesis constitute the *command*. Any valid shell script can be used for *command*, except:

- 1680           • A script consisting solely of redirections produces unspecified results
- 1681           • See the restriction on single subshells described below

1682           The results of command substitution shall not be processed for further tilde expansion,  
1683           parameter expansion, command substitution, or arithmetic expansion. If a command  
1684           substitution occurs inside double-quotes, it shall not be performed on the results of the  
1685           substitution.

1686           Command substitution can be nested. To specify nesting within the backquoted version, the  
1687           application shall precede the inner backquotes with backslashes, for example:

1688           `\*command*`

1689           If the command substitution consists of a single subshell, such as:

1690           \$( (*command*) )

1691           a conforming application shall separate the "\$(" and '((' into two tokens (that is, separate  
1692           them with white space). This is required to avoid any ambiguities with arithmetic expansion.

1693 **2.6.4 Arithmetic Expansion**

1694 Arithmetic expansion provides a mechanism for evaluating an arithmetic expression and  
 1695 substituting its value. The format for arithmetic expansion shall be as follows:

1696 `$( (expression) )` |

1697 The expression shall be treated as if it were in double-quotes, except that a double-quote inside  
 1698 the expression is not treated specially. The shell shall expand all tokens in the expression for  
 1699 parameter expansion, command substitution, and quote removal. |

1700 Next, the shell shall treat this as an arithmetic expression and substitute the value of the  
 1701 expression. The arithmetic expression shall be processed according to the rules given in Section  
 1702 1.7.2.1 (on page 2207), with the following exceptions: |

- 1703 • Only signed long integer arithmetic is required. |
- 1704 • Only the decimal-constant, octal-constant, and hexadecimal-constant constants specified in  
 1705 the ISO C standard, Subclause 6.4.4.1 are required to be recognized as constants. |
- 1706 • The *sizeof()* operator and the prefix and postfix "++" and "--" operators are not required. |
- 1707 • Selection, iteration, and jump statements are not supported. |

1708 As an extension, the shell may recognize arithmetic expressions beyond those listed. The shell  
 1709 may use a signed integer type with a rank larger than the rank of **signed long**. The shell may use  
 1710 a real-floating type instead of **signed long** as long as it does not affect the results in cases where  
 1711 there is no overflow. If the expression is invalid, the expansion fails and the shell shall write a  
 1712 message to standard error indicating the failure. |

1713 **Examples**

1714 A simple example using arithmetic expansion:

```
1715     # repeat a command 100 times |
1716     x=100 |
1717     while [ $x -gt 0 ] |
1718     do |
1719         command |
1720         x=$(( $x-1 )) |
1721     done |
```

1722 **2.6.5 Field Splitting**

1723 After parameter expansion (Section 2.6.2 (on page 2239)), command substitution (Section 2.6.3  
 1724 (on page 2242)), and arithmetic expansion (Section 2.6.4), the shell shall scan the results of  
 1725 expansions and substitutions that did not occur in double-quotes for field splitting and multiple  
 1726 fields can result.

1727 The shell shall treat each character of the *IFS* as a delimiter and use the delimiters to split the  
 1728 results of parameter expansion and command substitution into fields. |

- 1729 1. If the value of *IFS* is a <space>, <tab>, and <newline>, or if it is unset, any sequence of  
 1730 <space>s, <tab>s, or <newline>s at the beginning or end of the input shall be ignored and  
 1731 any sequence of those characters within the input shall delimit a field. For example, the  
 1732 input:

1733 `<newline><space><tab>foo<tab><tab>bar<space>` |

- 1734 yields two fields, **foo** and **bar**.
- 1735 2. If the value of *IFS* is null, no field splitting shall be performed.
- 1736 3. Otherwise, the following rules shall be applied in sequence. The term “*IFS* white space” is  
 1737 used to mean any sequence (zero or more instances) of white space characters that are in  
 1738 the *IFS* value (for example, if *IFS* contains <space>/<comma>/<tab>, any sequence of  
 1739 <space>s and <tab>s is considered *IFS* white space).
- 1740 a. *IFS* white space shall be ignored at the beginning and end of the input.
- 1741 b. Each occurrence in the input of an *IFS* character that is not *IFS* white space, along  
 1742 with any adjacent *IFS* white space, shall delimit a field, as described previously.
- 1743 c. Non-zero-length *IFS* white space shall delimit a field.

### 1744 2.6.6 Pathname Expansion

1745 After field splitting, if *set -f* is not in effect, each field in the resulting command line shall be  
 1746 expanded using the algorithm described in Section 2.13 (on page 2264), qualified by the rules in  
 1747 Section 2.13.3 (on page 2265).

### 1748 2.6.7 Quote Removal

1749 The quote characters: ‘\’, ‘\’’, and ‘’’ (backslash, single-quote, double-quote) that were  
 1750 present in the original word shall be removed unless they have themselves been quoted.

## 1751 2.7 Redirection

1752 Redirection is used to open and close files for the current shell execution environment (see  
 1753 Section 2.12 (on page 2263)) or for any command. *Redirection operators* can be used with numbers  
 1754 representing file descriptors (see the Base Definitions volume of IEEE Std 1003.1-200x, Section  
 1755 3.165, File Descriptor) as described below.

1756 The overall format used for redirection is:

1757 `[n]redir-op word` |

1758 The number *n* is an optional decimal number designating the file descriptor number; the  
 1759 application shall ensure it is delimited from any preceding text and immediately precede the  
 1760 redirection operator *redir-op*. If *n* is quoted, the number shall not be recognized as part of the  
 1761 redirection expression. For example:

1762 `echo \2>a` |

1763 writes the character 2 into file **a**. If any part of *redir-op* is quoted, no redirection expression is  
 1764 recognized. For example:

1765 `echo 2\>a` |

1766 writes the characters 2>a to standard output. The optional number, redirection operator, and  
 1767 *word* shall not appear in the arguments provided to the command to be executed (if any).

1768 Open files are represented by decimal numbers starting with zero. The largest possible value is  
 1769 implementation-defined; however, all implementations shall support at least 0 to 9, inclusive, for  
 1770 use by the application. These numbers are called *file descriptors*. The values 0, 1, and 2 have  
 1771 special meaning and conventional uses and are implied by certain redirection operations; they  
 1772 are referred to as *standard input*, *standard output*, and *standard error*, respectively. Programs  
 1773 usually take their input from standard input, and write output on standard output. Error

1774 messages are usually written on standard error. The redirection operators can be preceded by  
 1775 one or more digits (with no intervening <blank>s allowed) to designate the file descriptor  
 1776 number.

1777 If the redirection operator is "<<" or "<<-", the word that follows the redirection operator shall  
 1778 be subjected to quote removal; it is unspecified whether any of the other expansions occur. For  
 1779 the other redirection operators, the word that follows the redirection operator shall be subjected  
 1780 to tilde expansion, parameter expansion, command substitution, arithmetic expansion, and  
 1781 quote removal. Pathname expansion shall not be performed on the word by a non-interactive  
 1782 shell; an interactive shell may perform it, but shall do so only when the expansion would result  
 1783 in one word.

1784 If more than one redirection operator is specified with a command, the order of evaluation is  
 1785 from beginning to end.

1786 A failure to open or create a file shall cause a redirection to fail.

### 1787 2.7.1 Redirecting Input

1788 Input redirection shall cause the file whose name results from the expansion of *word* to be  
 1789 opened for reading on the designated file descriptor, or standard input if the file descriptor is not  
 1790 specified.

1791 The general format for redirecting input is:

```
1792     [n]<word
```

1793 where the optional *n* represents the file descriptor number. If the number is omitted, the  
 1794 redirection shall refer to standard input (file descriptor 0).

### 1795 2.7.2 Redirecting Output

1796 The two general formats for redirecting output are:

```
1797     [n]>word
1798     [n]>|word
```

1799 where the optional *n* represents the file descriptor number. If the number is omitted, the  
 1800 redirection shall refer to standard output (file descriptor 1).

1801 Output redirection using the '>' format shall fail if the *noclobber* option is set (see the  
 1802 description of *set -C*) and the file named by the expansion of *word* exists and is a regular file.  
 1803 Otherwise, redirection using the '>' or ">|" formats shall cause the file whose name results  
 1804 from the expansion of *word* to be created and opened for output on the designated file  
 1805 descriptor, or standard output if none is specified. If the file does not exist, it shall be created;  
 1806 otherwise, it shall be truncated to be an empty file after being opened.

### 1807 2.7.3 Appending Redirected Output

1808 Appended output redirection shall cause the file whose name results from the expansion of  
 1809 *word* to be opened for output on the designated file descriptor. The file is opened as if the *open()*  
 1810 function as defined in the System Interfaces volume of IEEE Std 1003.1-200x was called with the  
 1811 *O\_APPEND* flag. If the file does not exist, it shall be created.

1812 The general format for appending redirected output is as follows:

```
1813     [n]>>word
```

1814 where the optional *n* represents the file descriptor number. If the number is omitted, the  
 1815 redirection refers to standard output (file descriptor 1).

#### 1816 2.7.4 Here-Document

1817 The redirection operators "<<" and "<<-" both allow redirection of lines contained in a shell  
 1818 input file, known as a *here-document*, to the input of a command.

1819 The here-document shall be treated as a single word that begins after the next <newline> and  
 1820 continues until there is a line containing only the delimiter and a <newline>, with no <blank>s in  
 1821 between. Then the next here-document starts, if there is one. The format is as follows:

```
1822     [n]<<word
1823         here-document
1824     delimiter
```

1825 where the optional *n* represents the file descriptor number. If the number is omitted, the here-  
 1826 document refers to standard input (file descriptor 0).

1827 If any character in *word* is quoted, the delimiter shall be formed by performing quote removal on  
 1828 *word*, and the here-document lines shall not be expanded. Otherwise, the delimiter shall be the  
 1829 *word* itself.

1830 If no characters in *word* are quoted, all lines of the here-document shall be expanded for  
 1831 parameter expansion, command substitution, and arithmetic expansion. In this case, the  
 1832 backslash in the input behaves as the backslash inside double-quotes (see Section 2.2.3 (on page  
 1833 2232)). However, the double-quote character (' ') shall not be treated specially within a here-  
 1834 document, except when the double-quote appears within "\$()", "' '", or "\${ }".

1835 If the redirection symbol is "<<-", all leading tab characters shall be stripped from input lines  
 1836 and the line containing the trailing delimiter. If more than one "<<" or "<<-" operator is  
 1837 specified on a line, the here-document associated with the first operator shall be supplied first by  
 1838 the application and shall be read first by the shell.

#### 1839 Examples

1840 An example of a here-document follows:

```
1841     cat <<eof1; cat <<eof2
1842     Hi ,
1843     eof1
1844     Helene.
1845     eof2
```

#### 1846 2.7.5 Duplicating an Input File Descriptor

1847 The redirection operator:

```
1848     [n]<&word
```

1849 shall duplicate one input file descriptor from another, or shall close one. If *word* evaluates to one  
 1850 or more digits, the file descriptor denoted by *n*, or standard input if *n* is not specified, shall be  
 1851 made to be a copy of the file descriptor denoted by *word*; if the digits in *word* do not represent a  
 1852 file descriptor already open for input, a redirection error shall result; see Section 2.8.1 (on page  
 1853 2247). If *word* evaluates to '-', file descriptor *n*, or standard input if *n* is not specified, shall be  
 1854 closed. If *word* evaluates to something else, the behavior is unspecified.

## 1855 2.7.6 Duplicating an Output File Descriptor

1856 The redirection operator:

1857 `[n]>&word`

1858 shall duplicate one output file descriptor from another, or shall close one. If *word* evaluates to  
 1859 one or more digits, the file descriptor denoted by *n*, or standard output if *n* is not specified, shall  
 1860 be made to be a copy of the file descriptor denoted by *word*; if the digits in *word* do not represent  
 1861 a file descriptor already open for output, a redirection error shall result; see Section 2.8.1. If *word*  
 1862 evaluates to '-', file descriptor *n*, or standard output if *n* is not specified, is closed. If *word*  
 1863 evaluates to something else, the behavior is unspecified.

## 1864 2.7.7 Open File Descriptors for Reading and Writing

1865 The redirection operator:

1866 `[n]<>word`

1867 shall cause the file whose name is the expansion of *word* to be opened for both reading and  
 1868 writing on the file descriptor denoted by *n*, or standard input if *n* is not specified. If the file does  
 1869 not exist, it shall be created.

## 1870 2.8 Exit Status and Errors

### 1871 2.8.1 Consequences of Shell Errors

1872 For a non-interactive shell, an error condition encountered by a special built-in (see Section 2.14  
 1873 (on page 2266)) or other type of utility shall cause the shell to write a diagnostic message to  
 1874 standard error and exit as shown in the following table:

1875	Error	Special Built-In	Other Utilities
1876	Shell language syntax error	Shall exit	Shall exit
1877	Utility syntax error (option or operand error)	Shall exit	Shall not exit
1878	Redirection error	Shall exit	Shall not exit
1879	Variable assignment error	Shall exit	Shall not exit
1880	Expansion error	Shall exit	Shall exit
1881	Command not found	N/A	May exit
1882	Dot script not found	Shall exit	N/A

1883 An expansion error is one that occurs when the shell expansions defined in Section 2.6 (on page  
 1884 2238) are carried out (for example, "\$ {x!y}", because '!' is not a valid operator); an  
 1885 implementation may treat these as syntax errors if it is able to detect them during tokenization,  
 1886 rather than during expansion.

1887 If any of the errors shown as "shall exit" or "(may) exit" occur in a subshell, the subshell shall  
 1888 (respectively may) exit with a non-zero status, but the script containing the subshell shall not  
 1889 exit because of the error.

1890 In all of the cases shown in the table, an interactive shell shall write a diagnostic message to  
 1891 standard error without exiting.

## 1892 2.8.2 Exit Status for Commands

1893 Each command has an exit status that can influence the behavior of other shell commands. The  
 1894 exit status of commands that are not utilities is documented in this section. The exit status of the  
 1895 standard utilities is documented in their respective sections.

1896 If a command is not found, the exit status shall be 127. If the command name is found, but it is  
 1897 not an executable utility, the exit status shall be 126. Applications that invoke utilities without  
 1898 using the shell should use these exit status values to report similar errors.

1899 If a command fails during word expansion or redirection, its exit status shall be greater than  
 1900 zero.

1901 Internally, for purposes of deciding whether a command exits with a non-zero exit status, the  
 1902 shell shall recognize the entire status value retrieved for the command by the equivalent of the  
 1903 *wait()* function WEXITSTATUS macro (as defined in the System Interfaces volume of the  
 1904 IEEE Std 1003.1-200x). When reporting the exit status with the special parameter '?', the shell  
 1905 shall report the full eight bits of exit status available. The exit status of a command that  
 1906 terminated because it received a signal shall be reported as greater than 128.

## 1907 2.9 Shell Commands

1908 This section describes the basic structure of shell commands. The following command  
 1909 descriptions each describe a format of the command that is only used to aid the reader in  
 1910 recognizing the command type, and does not formally represent the syntax. Each description  
 1911 discusses the semantics of the command; for a formal definition of the command language,  
 1912 consult Section 2.10 (on page 2257).

1913 A *command* is one of the following:

- 1914 • *Simple command* (see Section 2.9.1)
- 1915 • *Pipeline* (see Section 2.9.2 (on page 2250))
- 1916 • *List* or *compound-list* (see Section 2.9.3 (on page 2251))
- 1917 • *Compound command* (see Section 2.9.4 (on page 2253))
- 1918 • *Function definition* (see Section 2.9.5 (on page 2256))

1919 Unless otherwise stated, the exit status of a command shall be that of the last simple command |  
 1920 executed by the command. There shall be no limit on the size of any shell command other than |  
 1921 that imposed by the underlying system (memory constraints, {ARG\_MAX}, and so on). |

### 1922 2.9.1 Simple Commands

1923 A *simple command* is a sequence of optional variable assignments and redirections, in any  
 1924 sequence, optionally followed by words and redirections, terminated by a control operator.

1925 When a given simple command is required to be executed (that is, when any conditional  
 1926 construct such as an AND-OR list or a **case** statement has not bypassed the simple command),  
 1927 the following expansions, assignments, and redirections shall all be performed from the |  
 1928 beginning of the command text to the end: |

- 1929 1. The words that are recognized as variable assignments or redirections according to Section  
 1930 2.10.2 (on page 2257) are saved for processing in steps 3 and 4.
- 1931 2. The words that are not variable assignments or redirections shall be expanded. If any fields  
 1932 remain following their expansion, the first field shall be considered the command name

- 1933 and remaining fields are the arguments for the command.
- 1934 3. Redirections shall be performed as described in Section 2.7 (on page 2244).
- 1935 4. Each variable assignment shall be expanded for tilde expansion, parameter expansion,  
1936 command substitution, arithmetic expansion, and quote removal prior to assigning the  
1937 value.

1938 In the preceding list, the order of steps 3 and 4 may be reversed for the processing of special  
1939 built-in utilities; see Section 2.14 (on page 2266).

1940 If no command name results, variable assignments shall affect the current execution  
1941 environment. Otherwise, the variable assignments shall be exported for the execution  
1942 environment of the command and shall not affect the current execution environment (except for  
1943 special built-ins). If any of the variable assignments attempt to assign a value to a read-only  
1944 variable, a variable assignment error shall occur. See Section 2.8.1 (on page 2247) for the  
1945 consequences of these errors.

1946 If there is no command name, any redirections shall be performed in a subshell environment; it  
1947 is unspecified whether this subshell environment is the same one as that used for a command  
1948 substitution within the command. (To affect the current execution environment, see the *exec* (on  
1949 page 2277) special built-in.) If any of the redirections performed in the current shell execution  
1950 environment fail, the command shall immediately fail with an exit status greater than zero, and  
1951 the shell shall write an error message indicating the failure. See Section 2.8.1 (on page 2247) for  
1952 the consequences of these failures on interactive and non-interactive shells.

1953 If there is a command name, execution shall continue as described in Section 2.9.1.1. If there is  
1954 no command name, but the command contained a command substitution, the command shall  
1955 complete with the exit status of the last command substitution performed. Otherwise, the  
1956 command shall complete with a zero exit status.

#### 1957 2.9.1.1 Command Search and Execution

1958 If a simple command results in a command name and an optional list of arguments, the  
1959 following actions shall be performed:

- 1960 1. If the command name does not contain any slashes, the first successful step in the  
1961 following sequence shall occur:
- 1962 a. If the command name matches the name of a special built-in utility, that special  
1963 built-in utility shall be invoked.
- 1964 b. If the command name matches the name of a function known to this shell, the  
1965 function shall be invoked as described in Section 2.9.5 (on page 2256). If the  
1966 implementation has provided a standard utility in the form of a function, it shall not  
1967 be recognized at this point. It shall be invoked in conjunction with the path search in  
1968 step 1d.
- 1969 c. If the command name matches the name of a utility listed in the following table, that  
1970 utility shall be invoked.
- |      |                |                |               |                |  |
|------|----------------|----------------|---------------|----------------|--|
| 1971 | <i>alias</i>   | <i>false</i>   | <i>jobs</i>   | <i>true</i>    |  |
| 1972 | <i>bg</i>      | <i>fc</i>      | <i>kill</i>   | <i>umask</i>   |  |
| 1973 | <i>cd</i>      | <i>fg</i>      | <i>newgrp</i> | <i>unalias</i> |  |
| 1974 | <i>command</i> | <i>getopts</i> | <i>read</i>   | <i>wait</i>    |  |
- 1975 d. Otherwise, the command shall be searched for using the *PATH* environment variable  
1976 as described in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 8,  
1977 Environment Variables:

- 1978 i. If the search is successful:
- 1979 a. If the system has implemented the utility as a regular built-in or as a shell
- 1980 function, it shall be invoked at this point in the path search.
- 1981 b. Otherwise, the shell executes the utility in a separate utility environment
- 1982 (see Section 2.12 (on page 2263)) with actions equivalent to calling the
- 1983 *execve()* function as defined in the System Interfaces volume of
- 1984 IEEE Std 1003.1-200x with the *path* argument set to the pathname
- 1985 resulting from the search, *arg0* set to the command name, and the
- 1986 remaining arguments set to the operands, if any.
- 1987 If the *execve()* function fails due to an error equivalent to the [ENOEXEC]
- 1988 error defined in the System Interfaces volume of IEEE Std 1003.1-200x, the
- 1989 shell shall execute a command equivalent to having a shell invoked with
- 1990 the command name as its first operand, with any remaining arguments |
- 1991 passed to the new shell. If the executable file is not a text file, the shell |
- 1992 may bypass this command execution. In this case, it shall write an error |
- 1993 message, and shall return an exit status of 126. |
- 1994 Once a utility has been searched for and found (either as a result of this specific
- 1995 search or as part of an unspecified shell start-up activity), an implementation
- 1996 may remember its location and need not search for the utility again unless the
- 1997 *PATH* variable has been the subject of an assignment. If the remembered
- 1998 location fails for a subsequent invocation, the shell shall repeat the search to
- 1999 find the new location for the utility, if any.
- 2000 ii. If the search is unsuccessful, the command shall fail with an exit status of 127
- 2001 and the shell shall write an error message.
- 2002 2. If the command name contains at least one slash, the shell shall execute the utility in a
- 2003 separate utility environment with actions equivalent to calling the *execve()* function
- 2004 defined in the System Interfaces volume of IEEE Std 1003.1-200x with the *path* and *arg0*
- 2005 arguments set to the command name, and the remaining arguments set to the operands, if
- 2006 any.
- 2007 If the *execve()* function fails due to an error equivalent to the [ENOEXEC] error, the shell
- 2008 shall execute a command equivalent to having a shell invoked with the command name as |
- 2009 its first operand, with any remaining arguments passed to the new shell. If the executable |
- 2010 file is not a text file, the shell may bypass this command execution. In this case, it shall |
- 2011 write an error message and shall return an exit status of 126. |

## 2012 2.9.2 Pipelines

2013 A *pipeline* is a sequence of one or more commands separated by the control operator ' | '. The

2014 standard output of all but the last command shall be connected to the standard input of the next

2015 command.

2016 The format for a pipeline is:

```
2017 [!] command1 [ | command2 ... ] |
```

2018 The standard output of *command1* shall be connected to the standard input of *command2*. The

2019 standard input, standard output, or both of a command shall be considered to be assigned by the

2020 pipeline before any redirection specified by redirection operators that are part of the command

2021 (see Section 2.7 (on page 2244)).

2022 If the pipeline is not in the background (see Section 2.9.3.1 (on page 2252)), the shell shall wait for  
 2023 the last command specified in the pipeline to complete, and may also wait for all commands to  
 2024 complete.

### 2025 **Exit Status**

2026 If the reserved word **!** does not precede the pipeline, the exit status shall be the exit status of the  
 2027 last command specified in the pipeline. Otherwise, the exit status shall be the logical NOT of the  
 2028 exit status of the last command. That is, if the last command returns zero, the exit status shall be  
 2029 1; if the last command returns greater than zero, the exit status shall be zero.

### 2030 **2.9.3 Lists**

2031 An *AND-OR list* is a sequence of one or more pipelines separated by the operators "&&" and  
 2032 "||".

2033 A *list* is a sequence of one or more AND-OR lists separated by the operators ';' and '&' and  
 2034 optionally terminated by ';;', '&', or <newline>.

2035 The operators "&&" and "||" shall have equal precedence and shall be evaluated with left  
 2036 associativity. For example, both of the following commands write solely **bar** to standard output:

```
2037     false && echo foo || echo bar
2038     true  || echo foo && echo bar
```

2039 A ';' or <newline> terminator shall cause the preceding AND-OR list to be executed  
 2040 sequentially; an '&' shall cause asynchronous execution of the preceding AND-OR list.

2041 The term *compound-list* is derived from the grammar in Section 2.10 (on page 2257); it is  
 2042 equivalent to a sequence of *lists*, separated by <newline>s, that can be preceded or followed by  
 2043 an arbitrary number of <newline>s.

### 2044 **Examples**

2045 The following is an example that illustrates <newline>s in compound-lists:

```
2046     while
2047         # a couple of <newline>s
2048
2049         # a list
2050         date && who || ls; cat file
2051         # a couple of <newline>s
2052
2053         # another list
2054         wc file > output & true
2055
2056     do
2057         # 2 lists
2058         ls
2059         cat file
2060     done
```

2058 2.9.3.1 *Asynchronous Lists*

2059 If a command is terminated by the control operator ampersand ('&'), the shell shall execute the  
2060 command asynchronously in a subshell. This means that the shell shall not wait for the  
2061 command to finish before executing the next command.

2062 The format for running a command in the background is:

2063 `command1 & [command2 & . . . ]` |

2064 The standard input for an asynchronous list, before any explicit redirections are performed, shall  
2065 be considered to be assigned to a file that has the same properties as `/dev/null`. If it is an  
2066 interactive shell, this need not happen. In all cases, explicit redirection of standard input shall  
2067 override this activity.

2068 When an element of an asynchronous list (the portion of the list ended by an ampersand, such as  
2069 *command1*, above) is started by the shell, the process ID of the last command in the asynchronous  
2070 list element shall become known in the current shell execution environment; see Section 2.12 (on  
2071 page 2263). This process ID shall remain known until:

- 2072 1. The command terminates and the application waits for the process ID.
- 2073 2. Another asynchronous list invoked before "\$!" (corresponding to the previous  
2074 asynchronous list) is expanded in the current execution environment.

2075 The implementation need not retain more than the {CHILD\_MAX} most recent entries in its list  
2076 of known process IDs in the current shell execution environment.

2077 **Exit Status**

2078 The exit status of an asynchronous list shall be zero.

2079 2.9.3.2 *Sequential Lists*

2080 Commands that are separated by a semicolon (;) shall be executed sequentially.

2081 The format for executing commands sequentially shall be:

2082 `command1 [ ; command2 ] . . .` |

2083 Each command shall be expanded and executed in the order specified.

2084 **Exit Status**

2085 The exit status of a sequential list shall be the exit status of the last command in the list.

2086 2.9.3.3 *AND Lists*

2087 The control operator "&&" denotes an AND list. The format shall be:

2088 `command1 [ && command2 ] . . .` |

2089 First *command1* shall be executed. If its exit status is zero, *command2* shall be executed, and so on,  
2090 until a command has a non-zero exit status or there are no more commands left to execute. The  
2091 commands are expanded only if they are executed.

2092 **Exit Status**

2093 The exit status of an AND list shall be the exit status of the last command that is executed in the  
 2094 list.

2095 **2.9.3.4 OR Lists**

2096 The control operator "`||`" denotes an OR List. The format shall be:

```
2097     command1 [ || command2 ] . . .
```

2098 First, *command1* shall be executed. If its exit status is non-zero, *command2* shall be executed, and  
 2099 so on, until a command has a zero exit status or there are no more commands left to execute.

2100 **Exit Status**

2101 The exit status of an OR list shall be the exit status of the last command that is executed in the  
 2102 list.

2103 **2.9.4 Compound Commands**

2104 The shell has several programming constructs that are *compound commands*, which provide  
 2105 control flow for commands. Each of these compound commands has a reserved word or control  
 2106 operator at the beginning, and a corresponding terminator reserved word or operator at the end.  
 2107 In addition, each can be followed by redirections on the same line as the terminator. Each  
 2108 redirection shall apply to all the commands within the compound command that do not  
 2109 explicitly override that redirection.

2110 **2.9.4.1 Grouping Commands**

2111 The format for grouping commands is as follows:

```
2112 (compound-list)      Execute compound-list in a subshell environment; see Section 2.12 (on page  

  2113 2263). Variable assignments and built-in commands that affect the  

  2114 environment shall not remain in effect after the list finishes.
```

```
2115 { compound-list; }   Execute compound-list in the current process environment. The semicolon  

  2116 shown here is an example of a control operator delimiting the } reserved  

  2117 word. Other delimiters are possible, as shown in Section 2.10 (on page  

  2118 2257); a <newline> is frequently used.
```

2119 **Exit Status**

2120 The exit status of a grouping command shall be the exit status of *list*.

2121 **2.9.4.2 For Loop**

2122 The **for** loop shall execute a sequence of commands for each member in a list of *items*. The **for**  
 2123 loop requires that the reserved words **do** and **done** be used to delimit the sequence of  
 2124 commands.

2125 The format for the **for** loop is as follows:

```
2126     for name [ in [word . . . ] ]  

  2127     do  

  2128         compound-list  

  2129     done
```

2130 First, the list of words following **in** shall be expanded to generate a list of items. Then, the  
 2131 variable *name* shall be set to each item, in turn, and the *compound-list* executed each time. If no  
 2132 items result from the expansion, the *compound-list* shall not be executed. Omitting:

2133 `in word ...` |

2134 shall be equivalent to: |

2135 `in "$@"` |

### 2136 **Exit Status**

2137 The exit status of a **for** command shall be the exit status of the last command that executes. If  
 2138 there are no items, the exit status shall be zero.

#### 2139 2.9.4.3 *Case Conditional Construct*

2140 The conditional construct **case** shall execute the *compound-list* corresponding to the first one of  
 2141 several *patterns* (see Section 2.13 (on page 2264)) that is matched by the string resulting from the  
 2142 tilde expansion, parameter expansion, command substitution, arithmetic expansion, and quote  
 2143 removal of the given word. The reserved word **in** shall denote the beginning of the patterns to be  
 2144 matched. Multiple patterns with the same *compound-list* shall be delimited by the '|' symbol.  
 2145 The control operator ')' terminates a list of patterns corresponding to a given action. The  
 2146 *compound-list* for each list of patterns, with the possible exception of the last, shall be terminated  
 2147 with ";;". The **case** construct terminates with the reserved word **esac** (**case** reversed).

2148 The format for the **case** construct is as follows:

```
2149     case word in
2150         [(]pattern1) compound-list;;
2151         [[(]pattern[ | pattern] ... ) compound-list;;] ...
2152         [[(]pattern[ | pattern] ... ) compound-list]
2153     esac
```

2154 The ";;" is optional for the last *compound-list*.

2155 In order from the beginning to the end of the **case** statement, each *pattern* that labels a  
 2156 *compound-list* shall be subjected to tilde expansion, parameter expansion, command substitution,  
 2157 and arithmetic expansion, and the result of these expansions shall be compared against the  
 2158 expansion of *word*, according to the rules described in Section 2.13 (on page 2264) (which also  
 2159 describes the effect of quoting parts of the pattern). After the first match, no more patterns shall  
 2160 be expanded, and the *compound-list* shall be executed. The order of expansion and comparison of  
 2161 multiple *patterns* that label a *compound-list* statement is unspecified.

### 2162 **Exit Status**

2163 The exit status of **case** shall be zero if no patterns are matched. Otherwise, the exit status shall be  
 2164 the exit status of the last command executed in the *compound-list*.

#### 2165 2.9.4.4 *If Conditional Construct*

2166 The **if** command shall execute a *compound-list* and use its exit status to determine whether to  
 2167 execute another *compound-list*.

2168 The format for the **if** construct is as follows:

```

2169     if compound-list
2170     then
2171         compound-list
2172     [elif compound-list
2173     then
2174         compound-list] ...
2175     [else
2176         compound-list]

```

2177 The **if** *compound-list* shall be executed; if its exit status is zero, the **then** *compound-list* shall be  
 2178 executed and the command shall complete. Otherwise, each **elif** *compound-list* shall be executed,  
 2179 in turn, and if its exit status is zero, the **then** *compound-list* shall be executed and the command  
 2180 shall complete. Otherwise, the **else** *compound-list* shall be executed.

#### 2181 **Exit Status**

2182 The exit status of the **if** command shall be the exit status of the **then** or **else** *compound-list* that  
 2183 was executed, or zero, if none was executed.

#### 2184 2.9.4.5 *While Loop*

2185 The **while** loop shall continuously execute one *compound-list* as long as another *compound-list* has  
 2186 a zero exit status.

2187 The format of the **while** loop is as follows:

```

2188     while compound-list-1
2189     do
2190         compound-list-2
2191     done

```

2192 The *compound-list-1* shall be executed, and if it has a non-zero exit status, the **while** command  
 2193 shall complete. Otherwise, the *compound-list-2* shall be executed, and the process shall repeat.

#### 2194 **Exit Status**

2195 The exit status of the **while** loop shall be the exit status of the last *compound-list-2* executed, or  
 2196 zero if none was executed.

#### 2197 2.9.4.6 *Until Loop*

2198 The **until** loop shall continuously execute one *compound-list* as long as another *compound-list* has  
 2199 a non-zero exit status.

2200 The format of the **until** loop is as follows:

```

2201     until compound-list-1
2202     do
2203         compound-list-2
2204     done

```

2205 The *compound-list-1* shall be executed, and if it has a zero exit status, the **until** command  
 2206 completes. Otherwise, the *compound-list-2* shall be executed, and the process repeats.

2207 **Exit Status**

2208 The exit status of the **until** loop shall be the exit status of the last *compound-list-2* executed, or  
2209 zero if none was executed.

2210 **2.9.5 Function Definition Command**

2211 A function is a user-defined name that is used as a simple command to call a compound  
2212 command with new positional parameters. A function is defined with a *function definition*  
2213 *command*.

2214 The format of a function definition command is as follows:

```
2215 fname( ) compound-command[io-redirect ...]
```

2216 The function is named *fname*; the application shall ensure that it is a name (see the Base  
2217 Definitions volume of IEEE Std 1003.1-200x, Section 3.230, Name). An implementation may  
2218 allow other characters in a function name as an extension. The implementation shall maintain  
2219 separate name spaces for functions and variables.

2220 The argument *compound-command* represents a compound command, as described in Section  
2221 2.9.4 (on page 2253).

2222 When the function is declared, none of the expansions in Section 2.6 (on page 2238) shall be  
2223 performed on the text in *compound-command* or *io-redirect*; all expansions shall be performed as  
2224 normal each time the function is called. Similarly, the optional *io-redirect* redirections and any  
2225 variable assignments within *compound-command* shall be performed during the execution of the  
2226 function itself, not the function definition. See Section 2.8.1 (on page 2247) for the consequences  
2227 of failures of these operations on interactive and non-interactive shells.

2228 When a function is executed, it shall have the syntax-error and variable-assignment properties  
2229 described for special built-in utilities in the enumerated list at the beginning of Section 2.14 (on  
2230 page 2266).

2231 The *compound-command* shall be executed whenever the function name is specified as the name  
2232 of a simple command (see Section 2.9.1.1 (on page 2249)). The operands to the command  
2233 temporarily shall become the positional parameters during the execution of the *compound-*  
2234 *command*; the special parameter '# ' also shall be changed to reflect the number of operands. The  
2235 special parameter 0 shall be unchanged. When the function completes, the values of the  
2236 positional parameters and the special parameter '# ' shall be restored to the values they had  
2237 before the function was executed. If the special built-in *return* is executed in the *compound-*  
2238 *command*, the function completes and execution shall resume with the next command after the  
2239 function call.

2240 **Exit Status**

2241 The exit status of a function definition shall be zero if the function was declared successfully;  
2242 otherwise, it shall be greater than zero. The exit status of a function invocation shall be the exit  
2243 status of the last command executed by the function.

2244 **2.10 Shell Grammar**

2245 The following grammar defines the Shell Command Language. This formal syntax shall take  
2246 precedence over the preceding text syntax description.

2247 **2.10.1 Shell Grammar Lexical Conventions**

2248 The input language to the shell must be first recognized at the character level. The resulting  
2249 tokens shall be classified by their immediate context according to the following rules (applied in  
2250 order). These rules shall be used to determine what a “token” is that is subject to parsing at the  
2251 token level. The rules for token recognition in Section 2.3 (on page 2233) shall apply.

- 2252 1. A <newline> shall be returned as the token identifier **NEWLINE**.
- 2253 2. If the token is an operator, the token identifier for that operator shall result.
- 2254 3. If the string consists solely of digits and the delimiter character is one of '<' or '>', the  
2255 token identifier **IO\_NUMBER** shall be returned.
- 2256 4. Otherwise, the token identifier **TOKEN** results.

2257 Further distinction on **TOKEN** is context-dependent. It may be that the same **TOKEN** yields  
2258 **WORD**, a **NAME**, an **ASSIGNMENT**, or one of the reserved words below, dependent upon the  
2259 context. Some of the productions in the grammar below are annotated with a rule number from  
2260 the following list. When a **TOKEN** is seen where one of those annotated productions could be  
2261 used to reduce the symbol, the applicable rule shall be applied to convert the token identifier  
2262 type of the **TOKEN** to a token identifier acceptable at that point in the grammar. The reduction  
2263 shall then proceed based upon the token identifier type yielded by the rule applied. When more  
2264 than one rule applies, the highest numbered rule shall apply (which in turn may refer to another  
2265 rule). (Note that except in rule 7, the presence of an '=' in the token has no effect.)

2266 The **WORD** tokens shall have the word expansion rules applied to them immediately before the  
2267 associated command is executed, not at the time the command is parsed.

2268 **2.10.2 Shell Grammar Rules**

- 2269 1. [Command Name]

2270 When the **TOKEN** is exactly a reserved word, the token identifier for that reserved word  
2271 shall result. Otherwise, the token **WORD** shall be returned. Also, if the parser is in any  
2272 state where only a reserved word could be the next correct token, proceed as above.

2273 **Note:** Because at this point quote marks are retained in the token, quoted strings cannot be  
2274 recognized as reserved words. This rule also implies that reserved words are not  
2275 recognized except in certain positions in the input, such as after a <newline> or  
2276 semicolon; the grammar presumes that if the reserved word is intended, it is properly  
2277 delimited by the user, and does not attempt to reflect that requirement directly. Also  
2278 note that line joining is done before tokenization, as described in Section 2.2.1 (on page  
2279 2232), so escaped <newline>s are already removed at this point.

2280 Rule 1 is not directly referenced in the grammar, but is referred to by other rules, or applies  
2281 globally.

- 2282 2. [Redirection to or from filename]

2283 The expansions specified in Section 2.7 (on page 2244) shall occur. As specified there,  
2284 exactly one field can result (or the result is unspecified), and there are additional  
2285 requirements on pathname expansion.

- 2286 3. [Redirection from here-document]  
2287 Quote removal shall be applied to the word to determine the delimiter that is used to find  
2288 the end of the here-document that begins after the next <newline>.
- 2289 4. [Case statement termination]  
2290 When the **TOKEN** is exactly the reserved word **esac**, the token identifier for **esac** shall  
2291 result. Otherwise, the token **WORD** shall be returned.
- 2292 5. [**NAME** in **for**]  
2293 When the **TOKEN** meets the requirements for a name (see the Base Definitions volume of  
2294 IEEE Std 1003.1-200x, Section 3.230, Name), the token identifier **NAME** shall result.  
2295 Otherwise, the token **WORD** shall be returned.
- 2296 6. [Third word of **for** and **case**]  
2297 When the **TOKEN** is exactly the reserved word **in**, the token identifier for **in** shall result.  
2298 Otherwise, the token **WORD** shall be returned. (As indicated in the grammar, a *linebreak*  
2299 precedes the token **in**. If <newline>s are present at the indicated location, it is the token  
2300 after them that is treated in this fashion.)
- 2301 7. [Assignment preceding command name]  
2302 a. [When the first word]  
2303 If the **TOKEN** does not contain the character '=' , rule 1 is applied. Otherwise, 7b  
2304 shall be applied.  
2305 b. [Not the first word]  
2306 If the **TOKEN** contains the equal sign character:  
2307 — If it begins with '=' , the token **WORD** shall be returned.  
2308 — If all the characters preceding '=' form a valid name (see the Base Definitions  
2309 volume of IEEE Std 1003.1-200x, Section 3.230, Name), the token  
2310 **ASSIGNMENT\_WORD** shall be returned. (Quoted characters cannot participate  
2311 in forming a valid name.)  
2312 — Otherwise, it is unspecified whether it is **ASSIGNMENT\_WORD** or **WORD** that  
2313 is returned.
- 2314 Assignment to the **NAME** shall occur as specified in Section 2.9.1 (on page 2248).
- 2315 8. [**NAME** in function]  
2316 When the **TOKEN** is exactly a reserved word, the token identifier for that reserved word  
2317 shall result. Otherwise, when the **TOKEN** meets the requirements for a name, the token  
2318 identifier **NAME** shall result. Otherwise, rule 7 applies.
- 2319 9. [Body of function]  
2320 Word expansion and assignment shall never occur, even when required by the rules above,  
2321 when this rule is being parsed. Each **TOKEN** that might either be expanded or have  
2322 assignment applied to it shall instead be returned as a single **WORD** consisting only of  
2323 characters that are exactly the token described in Section 2.3 (on page 2233).

```

2324      /* -----
2325      The grammar symbols
2326      ----- */

2327      %token  WORD
2328      %token  ASSIGNMENT_WORD
2329      %token  NAME
2330      %token  NEWLINE
2331      %token  IO_NUMBER

2332      /* The following are the operators mentioned above. */

2333      %token  AND_IF    OR_IF    DSEMI
2334      /*      '&&'      '| |'      ';'      */

2335      %token  DLESS    DGREAT    LESSAND    GREATAND    LESSGREAT    DLESSDASH
2336      /*      '<<'      '>>'      '<&'      '>&'      '<>'      '<<-'      */

2337      %token  CLOBBER
2338      /*      '>|'      */

2339      /* The following are the reserved words. */

2340      %token  If      Then      Else      Elif      Fi      Do      Done
2341      /*      'if'      'then'      'else'      'elif'      'fi'      'do'      'done'      */

2342      %token  Case      Esac      While      Until      For
2343      /*      'case'      'esac'      'while'      'until'      'for'      */

2344      /* These are reserved words, not operator tokens, and are
2345      recognized when reserved words are recognized. */

2346      %token  Lbrace      Rbrace      Bang
2347      /*      '{'      '}'      '!'      */

2348      %token  In
2349      /*      'in'      */

2350      /* -----
2351      The Grammar
2352      ----- */

2353      %start  complete_command
2354      %%
2355      complete_command : list separator
2356                      | list
2357                      ;
2358      list             : list separator_op and_or
2359                      |
2360                      and_or
2361                      ;
2362      and_or           :
2363                      | and_or AND_IF linebreak pipeline
2364                      | and_or OR_IF linebreak pipeline
2365                      ;
2366      pipeline         :
2367                      | pipe_sequence
2368                      | Bang pipe_sequence
2369                      ;
2370      pipe_sequence    :
2371                      | pipe_sequence '|' linebreak command

```

```

2370                                     ;
2371     command                          : simple_command
2372                                     | compound_command
2373                                     | compound_command redirect_list
2374                                     | function_definition
2375                                     ;
2376     compound_command                   : brace_group
2377                                     | subshell
2378                                     | for_clause
2379                                     | case_clause
2380                                     | if_clause
2381                                     | while_clause
2382                                     | until_clause
2383                                     ;
2384     subshell                           : '(' compound_list ')'
2385                                     ;
2386     compound_list                       :                term
2387                                     | newline_list term
2388                                     |                term separator
2389                                     | newline_list term separator
2390                                     ;
2391     term                                : term separator and_or
2392                                     |                and_or
2393                                     ;
2394     for_clause                           : For name linebreak                do_group
2395                                     | For name linebreak in                sequential_sep_do_group
2396                                     | For name linebreak in wordlist sequential_sep do_group
2397                                     ;
2398     name                                 : NAME                                /* Apply rule 5 */
2399                                     ;
2400     in                                   : In                                  /* Apply rule 6 */
2401                                     ;
2402     wordlist                             : wordlist WORD
2403                                     |                WORD
2404                                     ;
2405     case_clause                          : Case WORD linebreak in linebreak case_list Esac
2406                                     | Case WORD linebreak in linebreak case_list_ns Esac
2407                                     | Case WORD linebreak in linebreak Esac
2408                                     ;
2409     case_list_ns                         : case_list case_item_ns
2410                                     | case_item_ns
2411                                     ;
2412     case_list                            : case_list case_item
2413                                     | case_item
2414                                     ;
2415     case_item_ns                         : pattern ')' linebreak linebreak
2416                                     | pattern ')' compound_list linebreak
2417                                     | '(' pattern ')' linebreak linebreak
2418                                     | '(' pattern ')' compound_list linebreak
2419                                     ;
2420     case_item                            : pattern ')' linebreak DSEMI linebreak
2421                                     | pattern ')' compound_list linebreak

```

```

2422         | '(' pattern ')' linebreak linebreak
2423         | '(' pattern ')' compound_list linebreak
2424         ;
2425     pattern      :          WORD          /* Apply rule 4 */
2426         | pattern '|' WORD          /* Do not apply rule (4) */
2427         ;
2428     if_clause    : If compound_list Then compound_list else_part Fi
2429         | If compound_list Then compound_list          Fi
2430         ;
2431     else_part    : Elif compound_list Then else_part
2432         | Else compound_list
2433         ;
2434     while_clause : While compound_list do_group
2435         ;
2436     until_clause : Until compound_list do_group
2437         ;
2438     function_definition : fname '(' ')' linebreak function_body
2439         ;
2440     function_body  : compound_command          /* Apply rule 9 */
2441         | compound_command redirect_list /* Apply rule 9 */
2442         ;
2443     fname          : NAME                      /* Apply rule 8 */
2444         ;
2445     brace_group    : Lbrace compound_list Rbrace
2446         ;
2447     do_group       : Do compound_list Done
2448         ;
2449     simple_command : cmd_prefix cmd_word cmd_suffix
2450         | cmd_prefix cmd_word
2451         | cmd_prefix
2452         | cmd_name cmd_suffix
2453         | cmd_name
2454         ;
2455     cmd_name       : WORD                      /* Apply rule 7a */
2456         ;
2457     cmd_word       : WORD                      /* Apply rule 7b */
2458         ;
2459     cmd_prefix     :          io_redirect
2460         | cmd_prefix io_redirect
2461         |          ASSIGNMENT_WORD
2462         | cmd_prefix ASSIGNMENT_WORD
2463         ;
2464     cmd_suffix     :          io_redirect
2465         | cmd_suffix io_redirect
2466         |          WORD
2467         | cmd_suffix WORD
2468         ;
2469     redirect_list :          io_redirect
2470         | redirect_list io_redirect
2471         ;
2472     io_redirect    :          io_file
2473         | IO_NUMBER io_file

```

```

2474         |             io_here
2475         | IO_NUMBER io_here
2476         ;
2477     io_file      : '<'      filename
2478         | LESSAND  filename
2479         | '>'      filename
2480         | GREATAND filename
2481         | DGREAT  filename
2482         | LESSGREAT filename
2483         | CLOBBER  filename
2484         ;
2485     filename     : WORD                /* Apply rule 2 */
2486         ;
2487     io_here      : DLESS      here_end
2488         | DLESSDASH here_end
2489         ;
2490     here_end     : WORD                /* Apply rule 3 */
2491         ;
2492     newline_list :             NEWLINE
2493         | newline_list NEWLINE
2494         ;
2495     linebreak    : newline_list
2496         | /* empty */
2497         ;
2498     separator_op : '&'
2499         | ';'
2500         ;
2501     separator    : separator_op linebreak
2502         | newline_list
2503         ;
2504     sequential_sep : ';' linebreak
2505         | newline_list
2506         ;

```

## 2507 2.11 Signals and Error Handling

2508 When a command is in an asynchronous list, the shell shall prevent SIGQUIT and SIGINT  
 2509 signals from the keyboard from interrupting the command. Otherwise, signals shall have the  
 2510 values inherited by the shell from its parent (see also the *trap* (on page 2297) special built-in).

2511 When a signal for which a trap has been set is received while the shell is waiting for the  
 2512 completion of a utility executing a foreground command, the trap associated with that signal  
 2513 shall not be executed until after the foreground command has completed. When the shell is  
 2514 waiting, by means of the *wait* utility, for asynchronous commands to complete, the reception of a  
 2515 signal for which a trap has been set shall cause the *wait* utility to return immediately with an exit  
 2516 status >128, immediately after which the trap associated with that signal shall be taken.

2517 If multiple signals are pending for the shell for which there are associated trap actions, the order  
 2518 of execution of trap actions is unspecified.

## 2519 2.12 Shell Execution Environment

2520 A shell execution environment consists of the following:

- 2521 • Open files inherited upon invocation of the shell, plus open files controlled by *exec*
- 2522 • Working directory as set by *cd*
- 2523 • File creation mask set by *umask*
- 2524 • Current traps set by *trap*
- 2525 • Shell parameters that are set by variable assignment (see the *set* (on page 2287) special built-
- 2526 in) or from the System Interfaces volume of IEEE Std 1003.1-200x environment inherited by
- 2527 the shell when it begins (see the *export* (on page 2281) special built-in)
- 2528 • Shell functions; see Section 2.9.5 (on page 2256)
- 2529 • Options turned on at invocation or by *set*
- 2530 • Process IDs of the last commands in asynchronous lists known to this shell environment; see
- 2531 Section 2.9.3.1 (on page 2252)
- 2532 • Shell aliases; see Section 2.3.1 (on page 2234)

2533 Utilities other than the special built-ins (see Section 2.14 (on page 2266)) shall be invoked in a  
2534 separate environment that consists of the following. The initial value of these objects shall be the  
2535 same as that for the parent shell, except as noted below.

- 2536 • Open files inherited on invocation of the shell, open files controlled by the *exec* special built-
- 2537 in plus any modifications, and additions specified by any redirections to the utility
- 2538 • Current working directory
- 2539 • File creation mask
- 2540 • If the utility is a shell script, traps caught by the shell shall be set to the default values and
- 2541 traps ignored by the shell shall be set to be ignored by the utility; if the utility is not a shell
- 2542 script, the trap actions (default or ignore) shall be mapped into the appropriate signal
- 2543 handling actions for the utility
- 2544 • Variables with the *export* attribute, along with those explicitly exported for the duration of the
- 2545 command, shall be passed to the utility as System Interfaces volume of IEEE Std 1003.1-200x
- 2546 environment variables

2547 The environment of the shell process shall not be changed by the utility unless explicitly  
2548 specified by the utility description (for example, *cd* and *umask*).

2549 A subshell environment shall be created as a duplicate of the shell environment, except that  
2550 signal traps set by that shell environment shall be set to the default values. Changes made to the  
2551 subshell environment shall not affect the shell environment. Command substitution, commands  
2552 that are grouped with parentheses, and asynchronous lists shall be executed in a subshell  
2553 environment. Additionally, each command of a multi-command pipeline is in a subshell  
2554 environment; as an extension, however, any or all commands in a pipeline may be executed in  
2555 the current environment. All other commands shall be executed in the current shell  
2556 environment.

## 2557 2.13 Pattern Matching Notation

2558 The pattern matching notation described in this section is used to specify patterns for matching  
2559 strings in the shell. Historically, pattern matching notation is related to, but slightly different  
2560 from, the regular expression notation described in the Base Definitions volume of  
2561 IEEE Std 1003.1-200x, Chapter 9, Regular Expressions. For this reason, the description of the  
2562 rules for this pattern matching notation are based on the description of regular expression  
2563 notation, modified to include backslash escape processing.

### 2564 2.13.1 Patterns Matching a Single Character

2565 The following *patterns matching a single character* shall match a single character: *ordinary* |  
2566 *characters*, *special pattern characters*, and *pattern bracket expressions*. The pattern bracket expression  
2567 also shall match a single collating element. A backslash character shall escape the following  
2568 character. The escaping backslash shall be discarded.

2569 An ordinary character is a pattern that shall match itself. It can be any character in the supported  
2570 character set except for NUL, those special shell characters in Section 2.2 (on page 2232) that  
2571 require quoting, and the following three special pattern characters. Matching shall be based on  
2572 the bit pattern used for encoding the character, not on the graphic representation of the  
2573 character. If any character (ordinary, shell special, or pattern special) is quoted, that pattern shall  
2574 match the character itself. The shell special characters always require quoting.

2575 When unquoted and outside a bracket expression, the following three characters shall have  
2576 special meaning in the specification of patterns:

2577 ? A question-mark is a pattern that shall match any character.

2578 \* An asterisk is a pattern that shall match multiple characters, as described in Section 2.13.2.

2579 [ The open bracket shall introduce a pattern bracket expression.

2580 The description of basic regular expression bracket expressions in the Base Definitions volume  
2581 of IEEE Std 1003.1-200x, Section 9.3.5, RE Bracket Expression shall also apply to the pattern  
2582 bracket expression, except that the exclamation mark character ('!') shall replace the  
2583 circumflex character ('^') in its role in a *non-matching list* in the regular expression notation. A  
2584 bracket expression starting with an unquoted circumflex character produces unspecified results.

2585 When pattern matching is used where shell quote removal is not performed (such as in the  
2586 argument to the *find name* primary when *find* is being called using one of the *exec* functions as  
2587 defined in the System Interfaces volume of IEEE Std 1003.1-200x, or in the *pattern* argument to  
2588 the *fnmatch()* function), special characters can be escaped to remove their special meaning by  
2589 preceding them with a backslash character. This escaping backslash is discarded. The sequence  
2590 "\\\" represents one literal backslash. All of the requirements and effects of quoting on ordinary,  
2591 shell special, and special pattern characters shall apply to escaping in this context.

### 2592 2.13.2 Patterns Matching Multiple Characters

2593 The following rules are used to construct *patterns matching multiple characters* from *patterns*  
2594 *matching a single character*:

2595 1. The asterisk ('\*') is a pattern that shall match any string, including the null string.

2596 2. The concatenation of *patterns matching a single character* is a valid pattern that shall match  
2597 the concatenation of the single characters or collating elements matched by each of the  
2598 concatenated patterns.

- 2599           3. The concatenation of one or more *patterns matching a single character* with one or more  
 2600 asterisks is a valid pattern. In such patterns, each asterisk shall match a string of zero or  
 2601 more characters, matching the greatest possible number of characters that still allows the  
 2602 remainder of the pattern to match the string.

### 2603 2.13.3 Patterns Used for Filename Expansion

2604 The rules described so far in Section 2.13.1 (on page 2264) and Section 2.13.2 (on page 2264) are  
 2605 qualified by the following rules that apply when pattern matching notation is used for filename  
 2606 expansion:

- 2607 1. The slash character in a pathname shall be explicitly matched by using one or more slashes |  
 2608 in the pattern; it shall neither be matched by the asterisk or question-mark special |  
 2609 characters nor by a bracket expression. Slashes in the pattern shall be identified before |  
 2610 bracket expressions; thus, a slash cannot be included in a pattern bracket expression used |  
 2611 for filename expansion. If a slash character is found following an unescaped open square |  
 2612 bracket character before a corresponding closing square bracket is found, the open bracket |  
 2613 shall be treated as an ordinary character. For example, the pattern "a[b/c]d" does not |  
 2614 match such pathnames as **abd** or **a/d**. It only matches a pathname of literally **a[b/c]d**.
- 2615 2. If a filename begins with a period ( '.' ), the period shall be explicitly matched by using a |  
 2616 period as the first character of the pattern or immediately following a slash character. The |  
 2617 leading period shall not be matched by: |
  - 2618 • The asterisk or question-mark special characters
  - 2619 • A bracket expression containing a non-matching list, such as "[!a]", a range |  
 2620 expression, such as "[%-0]", or a character class expression, such as "[[:punct:]]"

2621 It is unspecified whether an explicit period in a bracket expression matching list, such as  
 2622 "[.abc]", can match a leading period in a filename.

- 2623 3. Specified patterns shall be matched against existing filenames and pathnames, as |  
 2624 appropriate. Each component that contains a pattern character shall require read |  
 2625 permission in the directory containing that component. Any component, except the last, |  
 2626 that does not contain a pattern character shall require search permission. For example, |  
 2627 given the pattern:

2628           /fo`o`/bar/x\*/bam

2629 search permission is needed for directories / and **foo**, search and read permissions are  
 2630 needed for directory **bar**, and search permission is needed for each **x\*** directory. If the  
 2631 pattern matches any existing filenames or pathnames, the pattern shall be replaced with  
 2632 those filenames and pathnames, sorted according to the collating sequence in effect in the  
 2633 current locale. If the pattern contains an invalid bracket expression or does not match any  
 2634 existing filenames or pathnames, the pattern string shall be left unchanged.

2635 **2.14 Special Built-In Utilities**

2636 The following *special built-in* utilities shall be supported in the shell command language. The  
2637 output of each command, if any, shall be written to standard output, subject to the normal  
2638 redirection and piping possible with all commands.

2639 The term *built-in* implies that the shell can execute the utility directly and does not need to |  
2640 search for it. An implementation may choose to make any utility a built-in; however, the special |  
2641 built-in utilities described here differ from regular built-in utilities in two respects:

- 2642 1. A syntax error in a special built-in utility may cause a shell executing that utility to abort,  
2643 while a syntax error in a regular built-in utility shall not cause a shell executing that utility  
2644 to abort. (See Section 2.8.1 (on page 2247) for the consequences of errors on interactive and  
2645 non-interactive shells.) If a special built-in utility encountering a syntax error does not  
2646 abort the shell, its exit value shall be non-zero.
- 2647 2. Variable assignments specified with special built-in utilities remain in effect after the  
2648 built-in completes; this shall not be the case with a regular built-in or other utility.

2649 The special built-in utilities in this section need not be provided in a manner accessible via the  
2650 *exec* family of functions defined in the System Interfaces volume of IEEE Std 1003.1-200x.

2651 Some of the special built-ins are described as conforming to the Base Definitions volume of  
2652 IEEE Std 1003.1-200x, Section 12.2, Utility Syntax Guidelines. For those that are not, the  
2653 requirement in Section 1.11 (on page 2221) that "--" be recognized as a first argument to be  
2654 discarded does not apply and a conforming application shall not use that argument. |

2655 **NAME**  
2656 break — exit from for, while, or until loop

2657 **SYNOPSIS**  
2658 break [*n*]

2659 **DESCRIPTION**  
2660 The *break* utility shall exit from the smallest enclosing **for**, **while**, or **until** loop, if any; or from the  
2661 *n*th enclosing loop if *n* is specified. The value of *n* is an unsigned decimal integer greater than or  
2662 equal to 1. The default shall be equivalent to *n*=1. If *n* is greater than the number of enclosing |  
2663 loops, the outermost enclosing loop shall be exited. Execution shall continue with the command |  
2664 immediately following the loop.

2665 **OPTIONS**  
2666 None.

2667 **OPERANDS**  
2668 None.

2669 **STDIN**  
2670 None.

2671 **INPUT FILES**  
2672 None.

2673 **ENVIRONMENT VARIABLES**  
2674 None.

2675 **ASYNCHRONOUS EVENTS**  
2676 None.

2677 **STDOUT**  
2678 None.

2679 **STDERR**  
2680 None.

2681 **OUTPUT FILES**  
2682 None.

2683 **EXTENDED DESCRIPTION**  
2684 None.

2685 **EXIT STATUS**  
2686 0 Successful completion.  
2687 >0 The *n* value was not an unsigned decimal integer greater than or equal to 1.

2688 **CONSEQUENCES OF ERRORS**  
2689 None.

**2690 APPLICATION USAGE**

2691 None.

**2692 EXAMPLES**

```
2693     for i in * do
2694         if test -d "$i" then break fi done
```

**2695 RATIONALE**

2696 In early proposals, consideration was given to expanding the syntax of *break* and *continue* to refer  
2697 to a label associated with the appropriate loop as a preferable alternative to the *n* method.  
2698 However, this volume of IEEE Std 1003.1-200x does reserve the namespace of command names  
2699 ending with a colon. It is anticipated that a future implementation could take advantage of this  
2700 and provide something like:

```
2701     outofloop: for i in a b c d e
2702     do
2703         for j in 0 1 2 3 4 5 6 7 8 9
2704         do
2705             if test -r "${i}${j}"
2706             then break outofloop
2707             fi
2708         done
2709     done
```

2710 and that this might be standardized after implementation experience is achieved.

**2711 FUTURE DIRECTIONS**

2712 None.

**2713 SEE ALSO**

2714 Section 2.14 (on page 2266)

**2715 CHANGE HISTORY**

2716 None.

2717 **NAME**

2718           colon — null utility

2719 **SYNOPSIS**2720           : [*argument* ...]2721 **DESCRIPTION**2722           This utility shall only expand command *arguments*. It is used when a command is needed, as in  
2723           the *then* condition of an **if** command, but nothing is to be done by the command.2724 **OPTIONS**

2725           None.

2726 **OPERANDS**

2727           None.

2728 **STDIN**

2729           None.

2730 **INPUT FILES**

2731           None.

2732 **ENVIRONMENT VARIABLES**

2733           None.

2734 **ASYNCHRONOUS EVENTS**

2735           None.

2736 **STDOUT**

2737           None.

2738 **STDERR**

2739           None.

2740 **OUTPUT FILES**

2741           None.

2742 **EXTENDED DESCRIPTION**

2743           None.

2744 **EXIT STATUS**

2745           Zero.

2746 **CONSEQUENCES OF ERRORS**

2747           None.

2748 **APPLICATION USAGE**

2749           None.

2750 **EXAMPLES**2751           : \${X=abc}  
2752           if       false  
2753           then     :  
2754           else     echo \$X  
2755           fi  
2756           **abc**2757           As with any of the special built-ins, the null utility can also have variable assignments and  
2758           redirections associated with it, such as:

- 2759            `x=y : > z`
- 2760            which sets variable `x` to the value `y` (so that it persists after the null utility completes) and creates  
2761            or truncates file `z`.
- 2762 **RATIONALE**
- 2763            None.
- 2764 **FUTURE DIRECTIONS**
- 2765            None.
- 2766 **SEE ALSO**
- 2767            Section 2.14 (on page 2266)
- 2768 **CHANGE HISTORY**
- 2769            None.

2770 **NAME**

2771 continue — continue for, while, or until loop

2772 **SYNOPSIS**2773 continue [*n*]2774 **DESCRIPTION**

2775 The *continue* utility shall return to the top of the smallest enclosing **for**, **while**, or **until** loop, or to  
2776 the top of the *n*th enclosing loop, if *n* is specified. This involves repeating the condition list of a  
2777 **while** or **until** loop or performing the next assignment of a **for** loop, and reexecuting the loop if  
2778 appropriate.

2779 The value of *n* is a decimal integer greater than or equal to 1. The default shall be equivalent to |  
2780 *n*=1. If *n* is greater than the number of enclosing loops, the outermost enclosing loop shall be |  
2781 used. |

2782 **OPTIONS**

2783 None.

2784 **OPERANDS**

2785 None.

2786 **STDIN**

2787 None.

2788 **INPUT FILES**

2789 None.

2790 **ENVIRONMENT VARIABLES**

2791 None.

2792 **ASYNCHRONOUS EVENTS**

2793 None.

2794 **STDOUT**

2795 None.

2796 **STDERR**

2797 None.

2798 **OUTPUT FILES**

2799 None.

2800 **EXTENDED DESCRIPTION**

2801 None.

2802 **EXIT STATUS**

2803 0 Successful completion.

2804 >0 The *n* value was not an unsigned decimal integer greater than or equal to 1.2805 **CONSEQUENCES OF ERRORS**

2806 None.

**2807 APPLICATION USAGE**

2808 None.

**2809 EXAMPLES**

```
2810     for i in *
2811     do
2812         if test -d "$i"
2813         then continue
2814         fi
2815         echo "\"$i\" is not a directory.
2816     done
```

**2817 RATIONALE**

2818 None.

**2819 FUTURE DIRECTIONS**

2820 None.

**2821 SEE ALSO**

2822 Section 2.14 (on page 2266)

**2823 CHANGE HISTORY**

2824 None.

2825 **NAME**

2826 dot — execute commands in current environment

2827 **SYNOPSIS**2828 . *file*2829 **DESCRIPTION**2830 The shell shall execute commands from the *file* in the current environment.

2831 If *file* does not contain a slash, the shell shall use the search path specified by *PATH* to find the  
2832 directory containing *file*. Unlike normal command search, however, the file searched for by the  
2833 *dot* utility need not be executable. If no readable file is found, a non-interactive shell shall abort;  
2834 an interactive shell shall write a diagnostic message to standard error, but this condition shall  
2835 not be considered a syntax error.

2836 **OPTIONS**

2837 None.

2838 **OPERANDS**

2839 None.

2840 **STDIN**

2841 None.

2842 **INPUT FILES**

2843 None.

2844 **ENVIRONMENT VARIABLES**

2845 None.

2846 **ASYNCHRONOUS EVENTS**

2847 None.

2848 **STDOUT**

2849 None.

2850 **STDERR**

2851 None.

2852 **OUTPUT FILES**

2853 None.

2854 **EXTENDED DESCRIPTION**

2855 None.

2856 **EXIT STATUS**

2857 Returns the value of the last command executed, or a zero exit status if no command is executed. |

2858 **CONSEQUENCES OF ERRORS**

2859 None. |

2860 **APPLICATION USAGE**

2861 None.

2862 **EXAMPLES**2863 `cat foobar`2864 `foo=hello bar=world`2865 `. foobar`2866 `echo $foo $bar`2867 `hello world`2868 **RATIONALE**

2869 Some older implementations searched the current directory for the *file*, even if the value of *PATH*  
2870 disallowed it. This behavior was omitted from this volume of IEEE Std 1003.1-200x due to  
2871 concerns about introducing the susceptibility to trojan horses that the user might be trying to  
2872 avoid by leaving *dot* out of *PATH*.

2873 The KornShell version of *dot* takes optional arguments that are set to the positional parameters.

2874 This is a valid extension that allows a *dot* script to behave identically to a function.

2875 **FUTURE DIRECTIONS**

2876 None.

2877 **SEE ALSO**

2878 Section 2.14 (on page 2266)

2879 **CHANGE HISTORY**

2880 None.

2881 **NAME**  
2882       eval — construct command by concatenating arguments

2883 **SYNOPSIS**  
2884       eval [*argument* ...]

2885 **DESCRIPTION**  
2886       The *eval* utility shall construct a command by concatenating *arguments* together, separating each  
2887       with a <space>. The constructed command shall be read and executed by the shell.

2888 **OPTIONS**  
2889       None.

2890 **OPERANDS**  
2891       None.

2892 **STDIN**  
2893       None.

2894 **INPUT FILES**  
2895       None.

2896 **ENVIRONMENT VARIABLES**  
2897       None.

2898 **ASYNCHRONOUS EVENTS**  
2899       None.

2900 **STDOUT**  
2901       None.

2902 **STDERR**  
2903       None.

2904 **OUTPUT FILES**  
2905       None.

2906 **EXTENDED DESCRIPTION**  
2907       None.

2908 **EXIT STATUS**  
2909       If there are no *arguments*, or only null arguments, *eval* shall return a zero exit status; otherwise, it  
2910       shall return the exit status of the command defined by the string of concatenated *arguments*  
2911       separated by spaces.

2912 **CONSEQUENCES OF ERRORS**  
2913       None.

2914 **APPLICATION USAGE**  
2915       None.

2916 **EXAMPLES**  
2917       foo=10 x=foo  
2918       y=' '\$x  
2919       echo \$y  
2920       **\$foo**  
2921       eval y=' '\$x  
2922       echo \$y  
2923       **10**

2924 **RATIONALE**

2925       None.

2926 **FUTURE DIRECTIONS**

2927       None.

2928 **SEE ALSO**

2929       Section 2.14 (on page 2266)

2930 **CHANGE HISTORY**

2931       None.

2932 **NAME**

2933           exec — execute commands and open, close, or copy file descriptors

2934 **SYNOPSIS**

2935           exec [*command* [*argument* ...]]

2936 **DESCRIPTION**

2937           The *exec* utility shall open, close, and/or copy file descriptors as specified by any redirections as  
2938           part of the command.

2939           If *exec* is specified without *command* or *arguments*, and any file descriptors with numbers greater  
2940           than 2 are opened with associated redirection statements, it is unspecified whether those file  
2941           descriptors remain open when the shell invokes another utility. Scripts concerned that child  
2942           shells could misuse open file descriptors can always close them explicitly, as shown in one of the  
2943           following examples.

2944           If *exec* is specified with *command*, it shall replace the shell with *command* without creating a new  
2945           process. If *arguments* are specified, they shall be arguments to *command*. Redirection affects the  
2946           current shell execution environment.

2947 **OPTIONS**

2948           None.

2949 **OPERANDS**

2950           None.

2951 **STDIN**

2952           None.

2953 **INPUT FILES**

2954           None.

2955 **ENVIRONMENT VARIABLES**

2956           None.

2957 **ASYNCHRONOUS EVENTS**

2958           None.

2959 **STDOUT**

2960           None.

2961 **STDERR**

2962           None.

2963 **OUTPUT FILES**

2964           None.

2965 **EXTENDED DESCRIPTION**

2966           None.

2967 **EXIT STATUS**

2968           If *command* is specified, *exec* shall not return to the shell; rather, the exit status of the process shall  
2969           be the exit status of the program implementing *command*, which overlaid the shell. If *command* is  
2970           not found, the exit status shall be 127. If *command* is found, but it is not an executable utility, the  
2971           exit status shall be 126. If a redirection error occurs (see Section 2.8.1 (on page 2247)), the shell  
2972           shall exit with a value in the range 1–125. Otherwise, *exec* shall return a zero exit status.

2973 **CONSEQUENCES OF ERRORS**

2974 None.

2975 **APPLICATION USAGE**

2976 None.

2977 **EXAMPLES**2978 Open *readfile* as file descriptor 3 for reading:2979 `exec 3< readfile`2980 Open *writefile* as file descriptor 4 for writing:2981 `exec 4> writefile`

2982 Make file descriptor 5 a copy of file descriptor 0:

2983 `exec 5<&0`

2984 Close file descriptor 3:

2985 `exec 3<&-`2986 Cat the file **maggie** by replacing the current shell with the *cat* utility:2987 `exec cat maggie`2988 **RATIONALE**

2989 Most historical implementations were not conformant in that:

2990 `foo=bar exec cmd`2991 did not pass **foo** to **cmd**.2992 **FUTURE DIRECTIONS**

2993 None.

2994 **SEE ALSO**

2995 Section 2.14 (on page 2266)

2996 **CHANGE HISTORY**

2997 None.

2998 **NAME**

2999           exit — cause the shell to exit

3000 **SYNOPSIS**3001           exit [*n*]3002 **DESCRIPTION**

3003           The *exit* utility shall cause the shell to exit with the exit status specified by the unsigned decimal  
3004           integer *n*. If *n* is specified, but its value is not between 0 and 255 inclusively, the exit status is  
3005           undefined.

3006           A *trap* on **EXIT** shall be executed before the shell terminates, except when the *exit* utility is  
3007           invoked in that *trap* itself, in which case the shell shall exit immediately.

3008 **OPTIONS**

3009           None.

3010 **OPERANDS**

3011           None.

3012 **STDIN**

3013           None.

3014 **INPUT FILES**

3015           None.

3016 **ENVIRONMENT VARIABLES**

3017           None.

3018 **ASYNCHRONOUS EVENTS**

3019           None.

3020 **STDOUT**

3021           None.

3022 **STDERR**

3023           None.

3024 **OUTPUT FILES**

3025           None.

3026 **EXTENDED DESCRIPTION**

3027           None.

3028 **EXIT STATUS**

3029           The exit status shall be *n*, if specified. Otherwise, the value shall be the exit value of the last  
3030           command executed, or zero if no command was executed. When *exit* is executed in a *trap* action,  
3031           the last command is considered to be the command that executed immediately preceding the  
3032           *trap* action. |

3033 **CONSEQUENCES OF ERRORS**

3034           None. |

**3035 APPLICATION USAGE**

3036 None.

**3037 EXAMPLES**

3038 Exit with a *true* value:

3039 `exit 0`

3040 Exit with a *false* value:

3041 `exit 1`

**3042 RATIONALE**

3043 As explained in other sections, certain exit status values have been reserved for special uses and  
3044 should be used by applications only for those purposes:

3045     126     A file to be executed was found, but it was not an executable utility.

3046     127     A utility to be executed was not found.

3047     >128    A command was interrupted by a signal.

**3048 FUTURE DIRECTIONS**

3049 None.

**3050 SEE ALSO**

3051 Section 2.14 (on page 2266)

**3052 CHANGE HISTORY**

3053 None.

3054 **NAME**

3055        export — set export attribute for variables

3056 **SYNOPSIS**3057        export name[=*word*]...

3058        export -p

3059 **DESCRIPTION**3060        The shell shall give the export attribute to the variables corresponding to the specified *names*,  
3061        which shall cause them to be in the environment of subsequently executed commands.3062        The *export* special built-in shall support the Base Definitions volume of IEEE Std 1003.1-200x,  
3063        Section 12.2, Utility Syntax Guidelines.3064        When **-p** is specified, *export* shall write to the standard output the names and values of all  
3065        exported variables, in the following format:3066        "export %s=%s\n", <*name*>, <*value*>3067        if *name* is set, and: |3068        "export %s\n", <*name*> |3069        if *name* is unset. |3070        The shell shall format the output, including the proper use of quoting, so that it is suitable for |  
3071        reinput to the shell as commands that achieve the same exporting results, except: |

3072            1. Read-only variables with values cannot be reset. |

3073            2. Variables that were unset at the time they were output need not be reset to the unset state |  
3074            if a value is assigned to the variable between the time the state was saved and the time at |  
3075            which the saved output is reinput to the shell. |

3076        When no arguments are given, the results are unspecified. |

3077 **OPTIONS**

3078        None.

3079 **OPERANDS**

3080        None.

3081 **STDIN**

3082        None.

3083 **INPUT FILES**

3084        None.

3085 **ENVIRONMENT VARIABLES**

3086        None.

3087 **ASYNCHRONOUS EVENTS**

3088        None.

3089 **STDOUT**

3090        None.

3091 **STDERR**

3092        None.

3093 **OUTPUT FILES**

3094 None.

3095 **EXTENDED DESCRIPTION**

3096 None.

3097 **EXIT STATUS**

3098 Zero.

3099 **CONSEQUENCES OF ERRORS**

3100 None.

3101 **APPLICATION USAGE**

3102 None.

3103 **EXAMPLES**3104 Export *PWD* and *HOME* variables:3105 `export PWD HOME`3106 Set and export the *PATH* variable:3107 `export PATH=/local/bin:$PATH`

3108 Save and restore all exported variables:

3109 `export -p > temp-file`3110 `unset a lot of variables`3111 `... processing`3112 `. temp-file`3113 **RATIONALE**

3114 Some historical shells use the no-argument case as the functional equivalent of what is required  
3115 here with `-p`. This feature was left unspecified because it is not historical practice in all shells,  
3116 and some scripts may rely on the now-unspecified results on their implementations. Attempts to  
3117 specify the `-p` output as the default case were unsuccessful in achieving consensus. The `-p`  
3118 option was added to allow portable access to the values that can be saved and then later restored  
3119 using; for example, a *dot* script.

3120 **FUTURE DIRECTIONS**

3121 None.

3122 **SEE ALSO**

3123 Section 2.14 (on page 2266)

3124 **CHANGE HISTORY**3125 **Issue 6**

3126 IEEE PASC Interpretation 1003.2 #203 is applied, clarifying the format when a variable is unset.

3127 **NAME**

3128       readonly — set read-only attribute for variables

3129 **SYNOPSIS**3130       readonly name[=*word*]...

3131       readonly -p

3132 **DESCRIPTION**

3133       The variables whose *names* are specified shall be given the *readonly* attribute. The values of  
 3134       variables with the *readonly* attribute cannot be changed by subsequent assignment, nor can those  
 3135       variables be unset by the *unset* utility.

3136       The *readonly* special built-in shall support the Base Definitions volume of IEEE Std 1003.1-200x,  
 3137       Section 12.2, Utility Syntax Guidelines.

3138       When **-p** is specified, *readonly* writes to the standard output the names and values of all read-  
 3139       only variables, in the following format:

3140       "readonly %s=%s\n", <*name*>, <*value*>3141       if *name* is set, and |3142       "readonly %s\n", <*name*> |3143       if *name* is unset. |

3144       The shell shall format the output, including the proper use of quoting, so that it is suitable for |  
 3145       reinput to the shell as commands that achieve the same value and read-only attribute-setting |  
 3146       results in a shell execution environment in which: |

3147       1. Variables with values at the time they were output do not have the read-only attribute set. |

3148       2. Variables that were unset at the time they were output do not have a value at the time at |  
 3149       which the saved output is reinput to the shell. |

3150       When no arguments are given, the results are unspecified. |

3151 **OPTIONS**

3152       None.

3153 **OPERANDS**

3154       None.

3155 **STDIN**

3156       None.

3157 **INPUT FILES**

3158       None.

3159 **ENVIRONMENT VARIABLES**

3160       None.

3161 **ASYNCHRONOUS EVENTS**

3162       None.

3163 **STDOUT**

3164       None.

3165 **STDERR**

3166 None.

3167 **OUTPUT FILES**

3168 None.

3169 **EXTENDED DESCRIPTION**

3170 None.

3171 **EXIT STATUS**

3172 Zero.

3173 **CONSEQUENCES OF ERRORS**

3174 None.

3175 **APPLICATION USAGE**

3176 None.

3177 **EXAMPLES**3178 `readonly HOME PWD`3179 **RATIONALE**

3180 Some historical shells preserve the read-only attribute across separate invocations. This volume  
3181 of IEEE Std 1003.1-200x allows this behavior, but does not require it.

3182 The `-p` option allows portable access to the values that can be saved and then later restored  
3183 using; for example, a *dot* script. Also see the RATIONALE for *export* (on page 2281) for a  
3184 description of the no-argument and `-p` output cases and a related example.

3185 Read-only functions were considered, but they were omitted as not being historical practice or  
3186 particularly useful. Furthermore, functions must not be *readonly* across invocations to preclude  
3187 *spoofing* (spoofing is the term for the practice of creating a program that acts like a well-known  
3188 utility with the intent of subverting the real intent of the user) of administrative or security-  
3189 relevant (or security-conscious) shell scripts.

3190 **FUTURE DIRECTIONS**

3191 None.

3192 **SEE ALSO**

3193 Section 2.14 (on page 2266)

3194 **CHANGE HISTORY**3195 **Issue 6**

3196 IEEE PASC Interpretation 1003.2 #203 is applied, clarifying the format when a variable is unset.

3197 **NAME**

3198           return — return from a function

3199 **SYNOPSIS**3200           return [*n*]3201 **DESCRIPTION**3202           The *return* utility shall cause the shell to stop executing the current function or *dot* script. If the  
3203           shell is not currently executing a function or *dot* script, the results are unspecified.3204 **OPTIONS**

3205           None.

3206 **OPERANDS**

3207           None.

3208 **STDIN**

3209           None.

3210 **INPUT FILES**

3211           None.

3212 **ENVIRONMENT VARIABLES**

3213           None.

3214 **ASYNCHRONOUS EVENTS**

3215           None.

3216 **STDOUT**

3217           None.

3218 **STDERR**

3219           None.

3220 **OUTPUT FILES**

3221           None.

3222 **EXTENDED DESCRIPTION**

3223           None.

3224 **EXIT STATUS**3225           The value of the special parameter '*?*' shall be set to *n*, an unsigned decimal integer, or to the  
3226           exit status of the last command executed if *n* is not specified. If the value of *n* is greater than 255,  
3227           the results are undefined. When *return* is executed in a *trap* action, the last command is  
3228           considered to be the command that executed immediately preceding the *trap* action.3229 **CONSEQUENCES OF ERRORS**

3230           None.

3231 **APPLICATION USAGE**

3232           None.

3233 **EXAMPLES**

3234           None.

3235 **RATIONALE**3236           The behavior of *return* when not in a function or *dot* script differs between the System V shell  
3237           and the KornShell. In the System V shell this is an error, whereas in the KornShell, the effect is  
3238           the same as *exit*.

3239           The results of returning a number greater than 255 are undefined because of differing practices  
3240           in the various historical implementations. Some shells AND out all but the low-order 8 bits;  
3241           others allow larger values, but not of unlimited size.

3242           See the discussion of appropriate exit status values under *exit* (on page 2279).

3243 **FUTURE DIRECTIONS**

3244           None.

3245 **SEE ALSO**

3246           Section 2.14 (on page 2266)

3247 **CHANGE HISTORY**

3248           None.

3249 **NAME**

3250 set — set or unset options and positional parameters

3251 **SYNOPSIS**3252 xSI set [-abCefmnuvx] [-h] [-o *option*] [*argument...*]3253 xSI set [+abCefmnuvx] [+h] [+o *option*] [*argument...*]3254 set --[*argument...*]

3255 set -o

3256 set +o

3257 **DESCRIPTION**

3258 If no options or *arguments* are specified, *set* shall write the names and values of all shell variables  
 3259 in the collation sequence of the current locale. Each *name* shall start on a separate line, using the  
 3260 format:

3261 "%s=%s\n", <*name*>, <*value*>

3262 The *value* string shall be written with appropriate quoting so that it is suitable for reinput to the  
 3263 shell, setting or resetting, as far as possible, the variables that are currently set. Read-only  
 3264 variables cannot be reset; see the description of shell quoting in Section 2.2 (on page 2232).

3265 When options are specified, they shall set or unset attributes of the shell, as described below.  
 3266 When *arguments* are specified, they cause positional parameters to be set or unset, as described  
 3267 below. Setting or unsetting attributes and positional parameters are not necessarily related  
 3268 actions, but they can be combined in a single invocation of *set*.

3269 The *set* special built-in shall support the Base Definitions volume of IEEE Std 1003.1-200x,  
 3270 Section 12.2, Utility Syntax Guidelines except that options can be specified with either a leading  
 3271 hyphen (meaning enable the option) or plus sign (meaning disable it).

3272 Implementations shall support the options in the following list in both their hyphen and plus-  
 3273 sign forms. These options can also be specified as options to *sh*.

3274 **-a** When this option is on, the export attribute shall be set for each variable to which an |  
 3275 assignment is performed; see the Base Definitions volume of IEEE Std 1003.1-200x, Section |  
 3276 4.21, Variable Assignment. If the assignment precedes a utility name in a command, the |  
 3277 export attribute shall not persist in the current execution environment after the utility |  
 3278 completes, with the exception that preceding one of the special built-in utilities causes the |  
 3279 export attribute to persist after the built-in has completed. If the assignment does not |  
 3280 precede a utility name in the command, or if the assignment is a result of the operation of |  
 3281 the *getopts* or *read* utilities, the export attribute shall persist until the variable is unset.

3282 **-b** This option shall be supported if the implementation supports the User Portability Utilities |  
 3283 option. It shall cause the shell to notify the user asynchronously of background job |  
 3284 completions. The following message is written to standard error: |

3285 "[%d]%c %s%s\n", <*job-number*>, <*current*>, <*status*>, <*job-name*>

3286 where the fields shall be as follows:

3287 <*current*> The character '+' identifies the job that would be used as a default for  
 3288 the *fg* or *bg* utilities; this job can also be specified using the *job\_id* "%+" or  
 3289 "%%". The character '-' identifies the job that would become the default  
 3290 if the current default job were to exit; this job can also be specified using  
 3291 the *job\_id* "%-". For other jobs, this field is a <*space*>. At most one job  
 3292 can be identified with '+' and at most one job can be identified with '-'.

- 3293 If there is any suspended job, then the current job shall be a suspended  
 3294 job. If there are at least two suspended jobs, then the previous job also  
 3295 shall be a suspended job.
- 3296 <*job-number*> A number that can be used to identify the process group to the *wait*, *fg*, *bg*,  
 3297 and *kill* utilities. Using these utilities, the job can be identified by  
 3298 prefixing the job number with '*%*'.
- 3299 <*status*> Unspecified.
- 3300 <*job-name*> Unspecified.
- 3301 When the shell notifies the user a job has been completed, it may remove the job's process  
 3302 ID from the list of those known in the current shell execution environment; see Section  
 3303 2.9.3.1 (on page 2252). Asynchronous notification shall not be enabled by default.
- 3304 **-C** (Uppercase C.) Prevent existing files from being overwritten by the shell's '*>*' redirection  
 3305 operator (see Section 2.7.2 (on page 2245)); the "*>|*" redirection operator shall override this  
 3306 *noclobber* option for an individual file.
- 3307 **-e** When this option is on, if a simple command fails for any of the reasons listed in Section  
 3308 2.8.1 (on page 2247) or returns an exit status value *>0*, and is not part of the compound list  
 3309 following a **while**, **until**, or **if** keyword, and is not a part of an AND or OR list, and is not a  
 3310 pipeline preceded by the **!** reserved word, then the shell shall immediately exit.
- 3311 **-f** The shell shall disable pathname expansion.
- 3312 XSI **-h** Locate and remember utilities invoked by functions as those functions are defined (the  
 3313 utilities are normally located when the function is executed).
- 3314 **-m** This option shall be supported if the implementation supports the User Portability Utilities |  
 3315 option. All jobs shall be run in their own process groups. Immediately before the shell |  
 3316 issues a prompt after completion of the background job, a message reporting the exit status |  
 3317 of the background job shall be written to standard error. If a foreground job stops, the shell |  
 3318 shall write a message to standard error to that effect, formatted as described by the *jobs* |  
 3319 utility. In addition, if a job changes status other than exiting (for example, if it stops for  
 3320 input or output or is stopped by a SIGSTOP signal), the shell shall write a similar message  
 3321 immediately prior to writing the next prompt. This option is enabled by default for  
 3322 interactive shells.
- 3323 **-n** The shell shall read commands but does not execute them; this can be used to check for  
 3324 shell script syntax errors. An interactive shell may ignore this option.
- 3325 **-o** Write the current settings of the options to standard output in an unspecified format.
- 3326 **+o** Write the current option settings to standard output in a format that is suitable for reinput  
 3327 to the shell as commands that achieve the same options settings.
- 3328 **-o option**  
 3329 This option is supported if the system supports the User Portability Utilities option. It shall  
 3330 set various options, many of which shall be equivalent to the single option letters. The  
 3331 following values of *option* shall be supported:
- 3332 *allexport* Equivalent to **-a**.
- 3333 *errexit* Equivalent to **-e**.
- 3334 *ignoreeof* Prevent an interactive shell from exiting on end-of-file. This setting prevents  
 3335 accidental logouts when *<control>-D* is entered. A user shall explicitly *exit* to  
 3336 leave the interactive shell.

3337	<i>monitor</i>	Equivalent to <b>-m</b> . This option is supported if the system supports the User Portability Utilities option.
3338		
3339	<i>noclobber</i>	Equivalent to <b>-C</b> (uppercase C).
3340	<i>noglob</i>	Equivalent to <b>-f</b> .
3341	<i>noexec</i>	Equivalent to <b>-n</b> .
3342	<i>nolog</i>	Prevent the entry of function definitions into the command history; see <b>Command History List</b> (on page 3052).
3343		
3344	<i>notify</i>	Equivalent to <b>-b</b> .
3345	<i>nounset</i>	Equivalent to <b>-u</b> .
3346	<i>verbose</i>	Equivalent to <b>-v</b> .
3347	<i>vi</i>	Allow shell command line editing using the built-in <i>vi</i> editor. Enabling <i>vi</i> mode shall disable any other command line editing mode provided as an implementation extension.
3348		
3349		
3350		It need not be possible to set <i>vi</i> mode on for certain block-mode terminals.
3351	<i>xtrace</i>	Equivalent to <b>-x</b> .
3352	<b>-u</b>	The shell shall write a message to standard error when it tries to expand a variable that is not set and immediately exit. An interactive shell shall not exit.
3353		
3354	<b>-v</b>	The shell shall write its input to standard error as it is read.
3355	<b>-x</b>	The shell shall write to standard error a trace for each command after it expands the command and before it executes it. It is unspecified whether the command that turns tracing off is traced.
3356		
3357		
3358		The default for all these options shall be off (unset) unless the shell was invoked with them on; see <i>sh</i> .
3359		
3360		The remaining arguments shall be assigned in order to the positional parameters. The special parameter '# ' shall be set to reflect the number of positional parameters. All positional parameters shall be unset before any new values are assigned.
3361		
3362		
3363		The special argument "--" immediately following the <i>set</i> command name can be used to delimit the arguments if the first argument begins with '+' or '-', or to prevent inadvertent listing of all shell variables when there are no arguments. The command <i>set--</i> without <i>argument</i> shall unset all positional parameters and set the special parameter '# ' to zero.
3364		
3365		
3366		
3367	<b>OPTIONS</b>	
3368		None.
3369	<b>OPERANDS</b>	
3370		None.
3371	<b>STDIN</b>	
3372		None.
3373	<b>INPUT FILES</b>	
3374		None.

## 3375 ENVIRONMENT VARIABLES

3376 None.

## 3377 ASYNCHRONOUS EVENTS

3378 None.

## 3379 STDOUT

3380 None.

## 3381 STDERR

3382 None.

## 3383 OUTPUT FILES

3384 None.

## 3385 EXTENDED DESCRIPTION

3386 None.

## 3387 EXIT STATUS

3388 Zero.

## 3389 CONSEQUENCES OF ERRORS

3390 None.

## 3391 APPLICATION USAGE

3392 None.

## 3393 EXAMPLES

3394 Write out all variables and their values:

3395 `set`

3396 Set \$1, \$2, and \$3 and set "\$#" to 3:

3397 `set c a b`3398 Turn on the `-x` and `-v` options:3399 `set -xv`

3400 Unset all positional parameters:

3401 `set --`3402 Set \$1 to the value of `-x`, even if `x` begins with `'-'` or `'+'`:3403 `set -- "$x"`3404 Set the positional parameters to the expansion of `x`, even if `x` expands with a leading `'-'` or `'+'`:3405 `set -- $x`

## 3406 RATIONALE

3407 The `set --` form is listed specifically in the SYNOPSIS even though this usage is implied by the  
3408 Utility Syntax Guidelines. The explanation of this feature removes any ambiguity about whether  
3409 the `set --` form might be misinterpreted as being equivalent to `set` without any options or  
3410 arguments. The functionality of this form has been adopted from the KornShell. In System V, `set`  
3411 `--` only unsets parameters if there is at least one argument; the only way to unset all parameters  
3412 is to use *shift*. Using the KornShell version should not affect System V scripts because there  
3413 should be no reason to issue it without arguments deliberately; if it were issued as, for example:

3414 `set -- "$@"`

3415 and there were in fact no arguments resulting from "\$@", unsetting the parameters would have  
3416 no result.

3417 The *set +* form in early proposals was omitted as being an unnecessary duplication of *set* alone  
3418 and not widespread historical practice.

3419 The *noclobber* option was changed to allow *set -C* as well as the *set -o noclobber* option. The  
3420 single-letter version was added so that the historical "\$-" paradigm would not be broken; see  
3421 Section 2.5.2 (on page 2235).

3422 The *-h* flag is related to command name hashing and is only required on XSI-conformant  
3423 systems.

3424 The following *set* flags were omitted intentionally with the following rationale:

3425 **-k** The *-k* flag was originally added by the author of the Bourne shell to make it easier for  
3426 users of pre-release versions of the shell. In early versions of the Bourne shell the construct  
3427 *set name=value*, had to be used to assign values to shell variables. The problem with *-k* is  
3428 that the behavior affects parsing, virtually precluding writing any compilers. To explain the  
3429 behavior of *-k*, it is necessary to describe the parsing algorithm, which is implementation-  
3430 defined. For example:

```
3431 set -k; echo name=value
```

3432 and:

```
3433 set x--k  
3434 echo name=value
```

3435 behave differently. The interaction with functions is even more complex. What is more, the  
3436 *-k* flag is never needed, since the command line could have been reordered.

3437 **-t** The *-t* flag is hard to specify and almost never used. The only known use could be done  
3438 with here-documents. Moreover, the behavior with *ksh* and *sh* differs. The reference page  
3439 says that it exits after reading and executing one command. What is one command? If the  
3440 input is *date;date*, *sh* executes both *date* commands while *ksh* does only the first.

3441 Consideration was given to rewriting *set* to simplify its confusing syntax. A specific suggestion  
3442 was that the *unset* utility should be used to unset options instead of using the non-*getopt()*-able  
3443 *+option* syntax. However, the conclusion was reached that the historical practice of using *+option*  
3444 was satisfactory and that there was no compelling reason to modify such widespread historical  
3445 practice.

3446 The *-o* option was adopted from the KornShell to address user needs. In addition to its generally  
3447 friendly interface, *-o* is needed to provide the *vi* command line editing mode, for which  
3448 historical practice yields no single-letter option name. (Although it might have been possible to  
3449 invent such a letter, it was recognized that other editing modes would be developed and *-o*  
3450 provides ample name space for describing such extensions.)

3451 Historical implementations are inconsistent in the format used for *-o* option status reporting.  
3452 The *+o* format without an option-argument was added to allow portable access to the options  
3453 that can be saved and then later restored using, for instance, a dot script.

3454 Historically, *sh* did trace the command *set +x*, but *ksh* did not.

3455 The *ignoreeof* setting prevents accidental logouts when the end-of-file character (typically  
3456 <control>-D) is entered. A user shall explicitly *exit* to leave the interactive shell.

3457 The *set -m* option was added to apply only to the UPE because it applies primarily to interactive  
3458 use, not shell script applications.

3459 The ability to do asynchronous notification became available in the 1988 version of the  
3460 KornShell. To have it occur, the user had to issue the command:

```
3461 trap "jobs -n" CLD
```

3462 The C shell provides two different levels of an asynchronous notification capability. The  
3463 environment variable *notify* is analogous to what is done in *set -b* or *set -o notify*. When set, it  
3464 notifies the user immediately of background job completions. When unset, this capability is  
3465 turned off.

3466 The other notification ability comes through the built-in utility *notify*. The syntax is:

```
3467 notify [%job ... ]
```

3468 By issuing *notify* with no operands, it causes the C shell to notify the user asynchronously when  
3469 the state of the current job changes. If given operands, *notify* asynchronously informs the user of  
3470 changes in the states of the specified jobs.

3471 To add asynchronous notification to the POSIX shell, neither the KornShell extensions to *trap*,  
3472 nor the C shell *notify* environment variable seemed appropriate (*notify* is not a proper POSIX  
3473 environment variable name).

3474 The *set -b* option was selected as a compromise.

3475 The *notify* built-in was considered to have more functionality than was required for simple  
3476 asynchronous notification.

#### 3477 **FUTURE DIRECTIONS**

3478 None.

#### 3479 **SEE ALSO**

3480 Section 2.14 (on page 2266)

#### 3481 **CHANGE HISTORY**

##### 3482 **Issue 6**

3483 The obsolescent *set* command name followed by ‘-’ has been removed.

3484 The following new requirements on POSIX implementations derive from alignment with the  
3485 Single UNIX Specification:

- 3486 • The *nolog* option is added to *set -o*.

3487 **NAME**

3488           shift — shift positional parameters

3489 **SYNOPSIS**3490           shift [*n*]3491 **DESCRIPTION**

3492           The positional parameters shall be shifted. Positional parameter 1 shall be assigned the value of  
3493           parameter (1+*n*), parameter 2 shall be assigned the value of parameter (2+*n*), and so on. The  
3494           parameters represented by the numbers "\$#" down to "\$#-*n*+1" shall be unset, and the  
3495           parameter '#' is updated to reflect the new number of positional parameters.

3496           The value *n* shall be an unsigned decimal integer less than or equal to the value of the special  
3497           parameter '#'. If *n* is not given, it shall be assumed to be 1. If *n* is 0, the positional and special  
3498           parameters are not changed.

3499 **OPTIONS**

3500           None.

3501 **OPERANDS**

3502           None.

3503 **STDIN**

3504           None.

3505 **INPUT FILES**

3506           None.

3507 **ENVIRONMENT VARIABLES**

3508           None.

3509 **ASYNCHRONOUS EVENTS**

3510           None.

3511 **STDOUT**

3512           None.

3513 **STDERR**

3514           None.

3515 **OUTPUT FILES**

3516           None.

3517 **EXTENDED DESCRIPTION**

3518           None.

3519 **EXIT STATUS**3520           The exit status is >0 if *n*>\$#; otherwise, it is zero.3521 **CONSEQUENCES OF ERRORS**

3522           None.

3523 **APPLICATION USAGE**

3524       None.

3525 **EXAMPLES**

3526       \$ set a b c d e

3527       \$ shift 2

3528       \$ echo \$\*

3529       c d e

3530 **RATIONALE**

3531       None.

3532 **FUTURE DIRECTIONS**

3533       None.

3534 **SEE ALSO**

3535       Section 2.14 (on page 2266)

3536 **CHANGE HISTORY**

3537       None.

3538 **NAME**  
3539 times — write process times

3540 **SYNOPSIS**  
3541 times

3542 **DESCRIPTION**  
3543 Write the accumulated user and system times for the shell and for all of its child processes, in the  
3544 following POSIX locale format:

3545 "%dm%fs %dm%fs\n%dm%fs %dm%fs\n", <shell user minutes>,  
3546 <shell user seconds>, <shell system minutes>,  
3547 <shell system seconds>, <children user minutes>,  
3548 <children user seconds>, <children system minutes>,  
3549 <children system seconds>

3550 The four pairs of times shall correspond to the members of the <sys/times.h> **tms** structure |  
3551 (defined in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 13, Headers) as  
3552 returned by *times()*: *tms\_utime*, *tms\_stime*, *tms\_cutime*, and *tms\_cstime*, respectively.

3553 **OPTIONS**  
3554 None.

3555 **OPERANDS**  
3556 None.

3557 **STDIN**  
3558 None.

3559 **INPUT FILES**  
3560 None.

3561 **ENVIRONMENT VARIABLES**  
3562 None.

3563 **ASYNCHRONOUS EVENTS**  
3564 None.

3565 **STDOUT**  
3566 None.

3567 **STDERR**  
3568 None.

3569 **OUTPUT FILES**  
3570 None.

3571 **EXTENDED DESCRIPTION**  
3572 None.

3573 **EXIT STATUS**  
3574 Zero.

3575 **CONSEQUENCES OF ERRORS**  
3576 None.

3577 **APPLICATION USAGE**

3578       None.

3579 **EXAMPLES**

3580       \$ times

3581       **0m0.43s 0m1.11s**3582       **8m44.18s 1m43.23s**3583 **RATIONALE**3584       The *times* special built-in from the Single UNIX Specification is now required for all conforming  
3585       shells.3586 **FUTURE DIRECTIONS**

3587       None.

3588 **SEE ALSO**

3589       Section 2.14 (on page 2266)

3590 **CHANGE HISTORY**

3591       None.

3592 **NAME**

3593 trap — trap signals

3594 **SYNOPSIS**3595 trap [*action condition ...*]3596 **DESCRIPTION**

3597 If *action* is '-', the shell shall reset each *condition* to the default value. If *action* is null (" "), the  
 3598 shell shall ignore each specified *condition* if it arises. Otherwise, the argument *action* shall be read  
 3599 and executed by the shell when one of the corresponding conditions arises. The action of *trap*  
 3600 shall override a previous action (either default action or one explicitly set). The value of "\$?"  
 3601 after the *trap* action completes shall be the value it had before *trap* was invoked.

3602 The condition can be EXIT, 0 (equivalent to EXIT), or a signal specified using a symbolic name,  
 3603 without the SIG prefix, as listed in the tables of signal names in the <signal.h> header defined in  
 3604 the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 13, Headers; for example, HUP,  
 3605 INT, QUIT, TERM. Implementations may permit lowercase signal names or names with the SIG  
 3606 prefix as an extension. Setting a trap for SIGKILL or SIGSTOP produces undefined results.

3607 The environment in which the shell executes a *trap* on EXIT shall be identical to the environment  
 3608 immediately after the last command executed before the *trap* on EXIT was taken.

3609 Each time *trap* is invoked, the *action* argument shall be processed in a manner equivalent to:

3610 eval "\$action"

3611 Signals that were ignored on entry to a non-interactive shell cannot be trapped or reset, although  
 3612 no error need be reported when attempting to do so. An interactive shell may reset or catch  
 3613 signals ignored on entry. Traps shall remain in place for a given shell until explicitly changed  
 3614 with another *trap* command.

3615 When a subshell is entered, traps that are not being ignored are set to the default actions. This  
 3616 does not imply that the *trap* command cannot be used within the subshell to set new traps.

3617 The *trap* command with no arguments shall write to standard output a list of commands  
 3618 associated with each condition. The format shall be:

3619 "trap -- %s %s ... \n", &lt;action&gt;, &lt;condition&gt; ...

3620 The shell shall format the output, including the proper use of quoting, so that it is suitable for  
 3621 reinput to the shell as commands that achieve the same trapping results. For example:

3622 save\_traps=\$(trap)

3623 ...

3624 eval "\$save\_traps"

3625 XSI the following signal names:  
 3626

3627

3628

Signal Number	Signal Name
1	SIGHUP
2	SIGINT
3	SIGQUIT
6	SIGABRT
9	SIGKILL
14	SIGALRM
15	SIGTERM

3629 XSI

3630 XSI

3631 XSI

3632 XSI

3633 XSI

3634 XSI

3635 XSI

3636 The *trap* special built-in shall conform to the Base Definitions volume of IEEE Std 1003.1-200x,  
 3637 Section 12.2, Utility Syntax Guidelines.

3638 **OPTIONS**

3639 None.

3640 **OPERANDS**

3641 None.

3642 **STDIN**

3643 None.

3644 **INPUT FILES**

3645 None.

3646 **ENVIRONMENT VARIABLES**

3647 None.

3648 **ASYNCHRONOUS EVENTS**

3649 None.

3650 **STDOUT**

3651 None.

3652 **STDERR**

3653 None.

3654 **OUTPUT FILES**

3655 None.

3656 **EXTENDED DESCRIPTION**

3657 None.

3658 **EXIT STATUS**

3659 XSI If the trap name or number is invalid, a non-zero exit status shall be returned; otherwise, zero  
 3660 XSI shall be returned. For both interactive and non-interactive shells, invalid signal names or  
 3661 numbers shall not be considered a syntax error and do not cause the shell to abort. |

3662 **CONSEQUENCES OF ERRORS**

3663 None. |

3664 **APPLICATION USAGE**

3665 None.

3666 **EXAMPLES**

3667 Write out a list of all traps and actions:

3668 trap

3669 Set a trap so the *logout* utility in the directory referred to by the *HOME* environment variable  
3670 executes when the shell terminates:

3671 trap '\$HOME/logout' EXIT

3672 or:

3673 trap '\$HOME/logout' 0

3674 Unset traps on INT, QUIT, TERM, and EXIT:

3675 trap - INT QUIT TERM EXIT

3676 **RATIONALE**3677 Implementations may permit lowercase signal names as an extension. Implementations may  
3678 also accept the names with the SIG prefix; no known historical shell does so. The *trap* and *kill*  
3679 utilities in this volume of IEEE Std 1003.1-200x are now consistent in their omission of the SIG  
3680 prefix for signal names. Some *kill* implementations do not allow the prefix, and *kill -l* lists the  
3681 signals without prefixes.3682 Trapping SIGKILL or SIGSTOP is syntactically accepted by some historical implementations, but  
3683 it has no effect. Portable POSIX applications cannot attempt to trap these signals.3684 The output format is not historical practice. Since the output of historical *trap* commands is not  
3685 portable (because numeric signal values are not portable) and had to change to become so, an  
3686 opportunity was taken to format the output in a way that a shell script could use to save and  
3687 then later reuse a trap if it wanted.3688 The KornShell uses an **ERR** trap that is triggered whenever *set -e* would cause an exit. This is  
3689 allowable as an extension, but was not mandated, as other shells have not used it.3690 The text about the environment for the EXIT trap invalidates the behavior of some historical  
3691 versions of interactive shells which, for example, close the standard input before executing a  
3692 trap on 0. For example, in some historical interactive shell sessions the following trap on 0 would  
3693 always print "--":

3694 trap 'read foo; echo "--\$foo--"' 0

3695 **FUTURE DIRECTIONS**

3696 None.

3697 **SEE ALSO**

3698 Section 2.14 (on page 2266)

3699 **CHANGE HISTORY**3700 **Issue 6**3701 XSI-conforming implementations provide the mapping of signal names to numbers given above  
3702 (previously this had been marked obsolescent). Other implementations need not provide this  
3703 optional mapping.

3704 **NAME**

3705           unset — unset values and attributes of variables and functions

3706 **SYNOPSIS**

3707           unset [-fv] *name* ...

3708 **DESCRIPTION**

3709           Each variable or function specified by *name* shall be unset.

3710           If **-v** is specified, *name* refers to a variable name and the shell shall unset it and remove it from  
3711           the environment. Read-only variables cannot be unset.

3712           If **-f** is specified, *name* refers to a function and the shell shall unset the function definition.

3713           If neither **-f** nor **-v** is specified, *name* refers to a variable; if a variable by that name does not  
3714           exist, it is unspecified whether a function by that name, if any, shall be unset.

3715           Unsetting a variable or function that was not previously set shall not be considered an error and  
3716           does not cause the shell to abort.

3717           The *unset* special built-in shall support the Base Definitions volume of IEEE Std 1003.1-200x,  
3718           Section 12.2, Utility Syntax Guidelines.

3719           Note that:

3720           VARIABLE=

3721           is not equivalent to an *unset* of **VARIABLE**; in the example, **VARIABLE** is set to " ". Also, the  
3722           variables that can be *unset* should not be misinterpreted to include the special parameters (see  
3723           Section 2.5.2 (on page 2235)).

3724 **OPTIONS**

3725           None.

3726 **OPERANDS**

3727           None.

3728 **STDIN**

3729           None.

3730 **INPUT FILES**

3731           None.

3732 **ENVIRONMENT VARIABLES**

3733           None.

3734 **ASYNCHRONOUS EVENTS**

3735           None.

3736 **STDOUT**

3737           None.

3738 **STDERR**

3739           None.

3740 **OUTPUT FILES**

3741           None.

3742 **EXTENDED DESCRIPTION**

3743           None.

3744 **EXIT STATUS**

3745           0 All *name* operands were successfully unset.

3746           >0 At least one *name* could not be unset.

3747 **CONSEQUENCES OF ERRORS**

3748           None.

3749 **APPLICATION USAGE**

3750           None.

3751 **EXAMPLES**

3752           Unset *VISUAL* variable:

3753           unset -v VISUAL

3754           Unset the functions **foo** and **bar**:

3755           unset -f foo bar

3756 **RATIONALE**

3757           Consideration was given to omitting the **-f** option in favor of an *unfunction* utility, but the standard developers decided to retain historical practice.

3759           The **-v** option was introduced because System V historically used one name space for both variables and functions. When *unset* is used without options, System V historically unset either a function or a variable, and there was no confusion about which one was intended. A portable POSIX application can use *unset* without an option to unset a variable, but not a function; the **-f** option must be used.

3764 **FUTURE DIRECTIONS**

3765           None.

3766 **SEE ALSO**

3767           Section 2.14 (on page 2266)

3768 **CHANGE HISTORY**

3769           None.



# Batch Environment Services

3771

3772 BE This chapter describes the services and utilities that shall be implemented on all systems that  
 3773 claim conformance to the Batch Environment option. This functionality is dependent on support  
 3774 of this option (and the rest of this section is not further shaded for this option).

## 3775 3.1 General Concepts

### 3776 3.1.1 Batch Client-Server Interaction

3777 Batch jobs are created and managed by batch servers. A batch client interacts with a batch server  
 3778 to access batch services on behalf of the user. In order to use batch services, a user must have  
 3779 access to a batch client.

3780 A batch server is a computational entity, such as a daemon process, that provides batch services.  
 3781 Batch servers route, queue, modify, and execute batch jobs on behalf of batch clients.

3782 The batch utilities described in this volume of IEEE Std 1003.1-200x (and listed in Table 3-1) are  
 3783 clients of batch services; they allow users to perform actions on the job such as creating,  
 3784 modifying, and deleting batch jobs from a shell command line. Although these batch utilities  
 3785 may be said to accomplish certain services, they actually obtain services on behalf of a user by  
 3786 means of requests to batch servers.

3787

**Table 3-1** Batch Utilities

3788	<i>qalter</i>	<i>qmove</i>	<i>qrls</i>	<i>qstat</i>
3789	<i>qdel</i>	<i>qmsg</i>	<i>qselect</i>	<i>qsub</i>
3790	<i>qhold</i>	<i>qrerun</i>	<i>qsig</i>	

3791 Client-server interaction takes place by means of the batch requests defined in this chapter.  
 3792 Because direct access to batch jobs and queues is limited to batch servers, clients and servers of  
 3793 different implementations can interoperate, since dependencies on private structures for batch  
 3794 jobs and queues are limited to batch servers. Also, batch servers may be clients of other batch  
 3795 servers.

### 3796 3.1.2 Batch Queues

3797 Two types of batch queue are described: *routing queues* and *execution queues*. When a batch job is  
 3798 placed in a routing queue, it is a candidate for routing. A batch job is removed from routing  
 3799 queues under the following conditions:

- 3800 • The batch job has been routed to another queue.
- 3801 • The batch job has been deleted from the batch queue.
- 3802 • The batch job has been aborted.

3803 When a batch job is placed in an execution queue, it is a candidate for execution.

3804 A batch job is removed from an execution queue under the following conditions:

- 3805 • The batch job has been executed and exited.

- 3806           • The batch job has been aborted.
- 3807           • The batch job has been deleted from the batch queue.
- 3808           • The batch job has been moved to another queue.

3809           Access to a batch queue is limited to the batch server that manages the batch queue. Clients  
3810           never access a batch queue or a batch job directly, either to read or write information; all client  
3811           access to batch queues or jobs takes place through batch servers.

### 3812   **3.1.3   Batch Job Creation**

3813           When a batch server creates a batch job on behalf of a client, it shall assign a batch job identifier |  
3814           to the job. A batch job identifier consists of both a sequence number that is unique among the |  
3815           sequence numbers issued by that server and the name of the server. Since the batch server name |  
3816           is unique within a name space, the job identifier is likewise unique within the name space.

3817           The batch server that creates a batch job shall return the batch server-assigned job identifier to |  
3818           the client that requested the job creation. If the batch server routes or moves the job to another |  
3819           server, it sends the job identifier with the job. Once assigned, the job identifier of a batch job shall |  
3820           never change. |

### 3821   **3.1.4   Batch Job Tracking**

3822           Since a batch job may be moved after creation, the batch server name component of the job |  
3823           identifier need not indicate the location of the job. An implementation may provide a batch job |  
3824           tracking mechanism, in which case the user generally does not need to know the location of the |  
3825           job. However, an implementation need not provide a batch job tracking mechanism, in which |  
3826           case the user must find routed jobs by probing the possible destinations. |

### 3827   **3.1.5   Batch Job Routing**

3828           To route a batch job, a batch server either moves the job to some other queue that is managed by  
3829           the batch server, or requests that some other batch server accept the job.

3830           Each routing queue has one or more queues to which it can route batch jobs. The batch server  
3831           administrator creates routing queues.

3832           A batch server may route a batch job from a routing queue to another routing queue. Batch  
3833           servers shall prevent or otherwise handle cases of circular routing paths. As a deferred service, a  
3834           batch server routes jobs from the routing queues that it manages. The algorithm by which a  
3835           batch server selects a batch queue to which to route a batch job is implementation-defined.

3836           A batch job need not be eligible for routing to all the batch queues fed by the routing queue from  
3837           which it is routed. A batch server that has been asked to accept the job may reject the request if  
3838           the job requires resources that are unavailable to that batch server, or if the client is not  
3839           authorized to access the batch server.

3840           Batch servers may route high-priority jobs before low-priority jobs, but, on other than  
3841           overloaded systems, the effect may be imperceptible to the user. If all the batch servers fed by a  
3842           routing queue reject requests to accept the job for reasons that are permanent, the batch server |  
3843           that manages the job shall abort the job. If all or some rejections are temporary, the batch server |  
3844           should try to route the job again at some later point. |

3845           The reasons for rejecting a batch job are implementation-defined. The reasons for which the |  
3846           routing should be retried later and the reasons for which the job should be aborted are also |  
3847           implementation-defined. |

**3848 3.1.6 Batch Job Execution**

3849 To execute a batch job is to create a session leader (a process) that runs the shell program  
3850 indicated by the *Shell\_Path* attribute of the job. The script shall be passed to the program as its  
3851 standard input. An implementation may pass the script to the program by other  
3852 implementation-defined means. At the time a batch job begins execution, it is defined to enter  
3853 the RUNNING state. The primary program that is executed by a batch job is typically, though  
3854 not necessarily, a shell program.

3855 A batch server shall execute eligible jobs as a deferred service—no client request is necessary  
3856 once the batch job is created and eligible. However, the attributes of a batch job, such as the job  
3857 hold type, may render the job ineligible. A batch server shall scan the execution queues that it  
3858 manages for jobs that are eligible for execution. The algorithm by which the batch server selects  
3859 eligible jobs for execution is implementation-defined.

3860 As part of creating the process for the batch job, the batch server shall open the standard output  
3861 and standard error streams of the session.

3862 The attributes of a batch job may indicate that the batch server executing the job shall send mail  
3863 to a list of users at the time it begins execution of the job.

**3864 3.1.7 Batch Job Exit**

3865 When the session leader of an executing job terminates, the job exits. As part of exiting a batch  
3866 job, the batch server that manages the job shall remove the job from the batch queue in which it  
3867 resides. The server shall transfer output files of the job to a location described by the attributes of  
3868 the job.

3869 The attributes of a batch job may indicate that the batch server managing the job shall send mail  
3870 to a list of users at the time the job exits.

**3871 3.1.8 Batch Job Abort**

3872 A batch server shall abort jobs for which a required deferred service cannot be performed. The  
3873 attributes of a batch job may indicate that the batch server that aborts the job shall send mail to a  
3874 list of users at the time it aborts the job.

**3875 3.1.9 Batch Authorization**

3876 Clients, such as the batch environment utilities (marked BE), access batch services by means of  
3877 requests to one or more batch servers. To acquire the services of any given batch server, the user  
3878 identifier under which the client runs must be authorized to use that batch server.

3879 The user with an associated user name that creates a batch job shall own the job and can perform  
3880 actions such as read, modify, delete, and move.

3881 A user identifier of the same value at a different host need not be the same user. For example,  
3882 user name *smith* at host **alpha** may or may not represent the same person as user name *smith* at  
3883 host **beta**. Likewise, the same person may have access to different user names on different hosts.

3884 An implementation may optionally provide an authorization mechanism that permits one user  
3885 name to access jobs under another user name.

3886 A process on a client host may be authorized to run processes under multiple user names at a  
3887 batch server host. Where appropriate, the utilities defined in this volume of IEEE Std 1003.1-200x  
3888 provide a means for a user to choose from among such user names when creating or modifying a  
3889 batch job.

**3890 3.1.10 Batch Administration**

3891 The processing of a batch job by a batch server is affected by the attributes of the job. The  
3892 processing of a batch job may also be affected by the attributes of the batch queue in which the  
3893 job resides and by the status of the batch server that manages the job. See also the Base |  
3894 Definitions volume of IEEE Std 1003.1-200x, Section 3.43, Batch Administrator and the Base |  
3895 Definitions volume of IEEE Std 1003.1-200x, Section 3.58, Batch Operator. |

**3896 3.1.11 Batch Notification**

3897 Whereas batch servers are persistent entities, clients are often transient. For example, the *qsub*  
3898 utility creates a batch job and exits. For this reason, batch servers notify users of batch job events  
3899 by sending mail to the user that owns the job, or to other designated users.

**3900 3.2 Batch Services**

3901 The presence of Batch Environment option services is indicated by the configuration variable  
3902 POSIX2\_PBS. A conforming batch server provides services as defined in this section.

3903 A batch server shall provide batch services in two ways: |

- 3904 1. The batch server provides a service at the request of a client.
- 3905 2. The batch server provides a deferred service as a result of a change in conditions  
3906 monitored by the batch server.

3907 If a batch server cannot complete a request, it shall reject the request. If a batch server cannot |  
3908 complete a deferred service for a batch job, the batch server shall abort the batch job. Table 3-2 |  
3909 (on page 2307) is a summary of environment variables that shall be supported by an  
3910 implementation of the batch server and utilities.

3911

**Table 3-2** Environment Variable Summary

3912

3913

3914

3915

3916

3917

3918

3919

3920

3921

3922

3923

3924

3925

3926

3927

3928

3929

3930

Variable	Description
<i>PBS_DPREFIX</i>	Defines the directive prefix (see <i>qsub</i> )
<i>PBS_ENVIRONMENT</i>	Batch Job is batch or interactive (see Section 3.2.2.1 (on page 2308))
<i>PBS_JOBID</i>	The <i>job_identifier</i> attribute of job (see Section 3.2.3.8 (on page 2320))
<i>PBS_JOBNAME</i>	The <i>job_name</i> attribute of job (see Section 3.2.3.8 (on page 2320))
<i>PBS_O_HOME</i>	Defines the <i>HOME</i> of the batch client (see <i>qsub</i> )
<i>PBS_O_HOST</i>	Defines the host name of the batch client (see <i>qsub</i> )
<i>PBS_O_LANG</i>	Defines the <i>LANG</i> of the batch client (see <i>qsub</i> )
<i>PBS_O_LOGNAME</i>	Defines the <i>LOGNAME</i> of the batch client (see <i>qsub</i> )
<i>PBS_O_MAIL</i>	Defines the <i>MAIL</i> of the batch client (see <i>qsub</i> )
<i>PBS_O_PATH</i>	Defines the <i>PATH</i> of the batch client (see <i>qsub</i> )
<i>PBS_O_QUEUE</i>	Defines the submit queue of the batch client (see <i>qsub</i> )
<i>PBS_O_SHELL</i>	Defines the <i>SHELL</i> of the batch client (see <i>qsub</i> )
<i>PBS_O_TZ</i>	Defines the <i>TZ</i> of the batch client (see <i>qsub</i> )
<i>PBS_O_WORKDIR</i>	Defines the working directory of the batch client (see <i>qsub</i> )
<i>PBS_QUEUE</i>	Defines the initial execution queue (see Section 3.2.2.1 (on page 2308))

### 3931 3.2.1 Batch Job States

3932

3933

3934

3935

3936

A batch job shall always be in one of the following states: QUEUED, RUNNING, HELD, WAITING, EXITING, or TRANSITING. The state of a batch job determines the types of requests that the batch server that manages the batch job can accept for the batch job. A batch server shall change the state of a batch job either in response to service requests from clients or as a result of deferred services, such as job execution or job routing.

3937

3938

A batch job that is in the QUEUED state resides in a queue but is still pending either execution or routing, depending on the queue type.

3939

3940

3941

3942

3943

A batch server that queues a batch job in a routing queue shall put the batch job in the QUEUED state. A batch server that puts a batch job in an execution queue, but has not yet executed the batch job, shall put the batch job in the QUEUED state. A batch job that resides in an execution queue and is executing is defined to be in the RUNNING state. While a batch job is in the RUNNING state, a session leader is associated with the batch job.

3944

3945

A batch job that resides in an execution queue, but is ineligible to run because of a hold attribute, is defined to be in the HELD state.

3946

3947

A batch job that is not held, but must wait until a future date and time before executing, is defined to be in the WAITING state.

3948

3949

When the session leader associated with a running job exits, the batch job shall be placed in the EXITING state.

3950

3951

3952

3953

3954

A batch job for which the session leader has terminated is defined to be in the EXITING state, and the batch server that manages such a batch job cannot accept job modification requests that affect the batch job. While a batch job is in the EXITING state, the batch server that manages the batch job is staging output files and notifying clients of job completion. Once a batch job has exited, it no longer exists as an object managed by a batch server.

3955 A batch job that is being moved from a routing queue to another queue is defined to be in the  
3956 TRANSITING state.

3957 When a batch job in a routing queue has been selected to be moved to a new destination, then  
3958 the batch job shall be in either the QUEUED state or the TRANSITING state, depending on the  
3959 batch server implementation.

3960 Batch jobs with either a *Execution\_Time* attribute value set in the future or a *Hold\_Types* attribute  
3961 of value not equal to NO\_HOLD, or both, may be routed or held in the routing queue. The  
3962 treatment of jobs with the *Execution\_Time* or *Hold\_Types* attributes in a routing queue is  
3963 implementation-defined.

3964 When a batch job in a routing queue has not been selected to be moved to a new destination and  
3965 the batch job has a *Hold\_Types* attribute value of other than NO\_HOLD, then the job should be in  
3966 the HELD state.

3967 **Note:** The effect of a hold upon a batch job in a routing queue is implementation-defined. The  
3968 implementation should use the state that matches whether the batch job can route with a hold  
3969 or not.

3970 When a batch job in a routing queue has not been selected to be moved to a new destination and  
3971 the batch job has:

- 3972 • A *Hold\_Types* attribute value of NO\_HOLD
- 3973 • An *Execution\_Time* attribute in the past

3974 then the batch job shall be in the QUEUED state.

3975 When a batch job in a routing queue has not been selected to be moved to a new destination and  
3976 the batch job has:

- 3977 • A *Hold\_Types* attribute value of NO\_HOLD
- 3978 • An *Execution\_Time* attribute in the future

3979 then the batch job may be in the WAITING state.

3980 **Note:** The effect of a future execution time upon a batch job in a routing queue is implementation-  
3981 defined. The implementation should use the state that matches whether the batch job can route  
3982 with a hold or not.

3983 Table 3-3 (on page 2309) describes the next state of a batch job, given the current state of the  
3984 batch job and the type of request. Table 3-4 (on page 2310) describes the response of a batch  
3985 server to a request, given the current state of the batch job and the type of request.

### 3986 3.2.2 Deferred Batch Services

3987 This section describes the deferred services performed by batch servers: job execution, job  
3988 routing, job exit, job abort, and the rerunning of jobs after a restart.

#### 3989 3.2.2.1 Batch Job Execution

3990 To execute a batch job is to create a session leader (a process) that runs the shell program  
3991 indicated by the *Shell\_Path\_List* attribute of the batch job. The script is passed to the program as  
3992 its standard input. An implementation may pass the script to the program by other  
3993 implementation-defined means. At the time a batch job begins execution, it is defined to enter  
3994 the RUNNING state.

3995

Table 3-3 Next State Table

3996

3997

3998

3999

4000

4001

4002

4003

4004

4005

4006

4007

4008

4009

4010

4011

4012

Request Type	Current State						
	X	Q	R	H	W	E	T
Queue Batch Job Request	Q	e	e	e	e	e	e
Modify Batch Job Request	e	Q	R	H	W	e	T
Delete Batch Job Request	e	X	E	X	X	E	X
Batch Job Message Request	e	Q	R	H	W	E	T
Rerun Batch Job Request	e	e	Q	e	e	e	e
Signal Batch Job Request	e	e	R	H	W	e	e
Batch Job Status Request	e	Q	R	H	W	E	T
Batch Queue Status Request	X	Q	R	H	W	E	T
Server Status Request	X	Q	R	H	W	E	T
Select Batch Jobs Request	X	Q	R	H	W	E	T
Move Batch Job Request	e	Q	R	H	W	e	T
Hold Batch Job Request	e	H	R/H	H	H	e	T
Release Batch Job Request	Q	R	Q/W/H	W	e	T	
Server Shutdown Request	X	Q	Q	H	W	E	T
Locate Batch Job Request	e	Q	R	H	W	E	T

4013

**Legend**

4014

X Nonexistent

4015

Q QUEUED

4016

R RUNNING

4017

H HELD

4018

W WAITING

4019

E EXITING

4020

T TRANSITING

4021

e Error

4022

4023

4024

4025

4026

A batch server that has an execution queue containing jobs is said to own the queue and manage the batch jobs in that queue. A batch server that has been started shall execute the batch jobs in the execution queues owned by the batch server. The batch server shall schedule for execution those jobs in the execution queues that are in the QUEUED state. The algorithm for scheduling jobs is implementation-defined.

4027

4028

4029

A batch server that executes a batch job shall create, in the environment of the session leader of the batch job, an environment variable named *PBS\_ENVIRONMENT*, the value of which is the string *PBS\_BATCH* encoded in the portable character set.

4030

4031

4032

A batch server that executes a batch job shall create, in the environment of the session leader of the batch job, an environment variable named *PBS\_QUEUE*, the value of which is the name of the execution queue of the batch job encoded in the portable character set.

4033

4034

4035

4036

4037

4038

To rerun a batch job is to requeue a batch job that is currently executing and then kill the session leader of the executing job by sending a SIGKILL prior to completion; see Section 3.2.3.11 (on page 2322). A batch server that reruns a batch job shall append the standard output and standard error files of the batch job to the corresponding files of the previous execution, if they exist, with appropriate annotation. If either file does not exist, that file shall be created as in normal execution.

4039

Table 3-4 Results/Output Table

4040

4041

4042

4043

4044

4045

4046

4047

4048

4049

4050

4051

4052

4053

4054

4055

4056

Request Type	Current State						
	X	Q	R	H	W	E	T
Queue Batch Job Request	O	e	e	e	e	e	e
Modify Batch Job Request	e	O	e	O	O	e	e
Delete Batch Job Request	e	O	O	O	O	e	O
Batch Job Message Request	e	e	O	e	e	e	e
Rerun Batch Job Request	e	e	O	e	e	e	e
Signal Batch Job Request	e	e	O	e	e	e	e
Batch Job Status Request	e	O	O	O	O	O	O
Batch Queue Status Request	O	O	O	O	O	O	O
Server Status Request	O	O	O	O	O	O	O
Select Batch Job Request	e	O	O	O	O	O	O
Move Batch Job Request	e	O	O	O	O	e	e
Hold Batch Job Request	e	O	O	O	O	e	e
Release Batch Job Request	e	O	e	O	O	e	e
Server Shutdown Request	O	O	e	O	O	e	e
Locate Batch Job Request	e	O	O	O	O	O	O

4057

**Legend**

4058

O OK

4059

e Error message

4060

4061

The execution of a batch job by a batch server shall be controlled by job, queue, and server attributes, as defined in this section.

4062

**Account\_Name Attribute**

4063

4064

4065

Batch accounting is an optional feature of batch servers. If a batch server implements accounting, the statements in this section apply and the configuration variable `POSIX2_PBS_ACCOUNTING` shall be set to 1.

4066

4067

A batch server that executes a batch job shall charge the account named in the *Account\_Name* attribute of the batch job for resources consumed by the batch job.

4068

4069

If the *Account\_Name* attribute of the batch job is absent from the batch job attribute list or is altered while the batch job is in execution, the batch server action is implementation-defined.

4070

**Checkpoint Attribute**

4071

4072

4073

Batch checkpointing is an optional feature of batch servers. If a batch server implements checkpointing, the statements in this section apply and the configuration variable `POSIX2_PBS_CHECKPOINT` shall be set to 1.

4074

4075

4076

4077

4078

There are two attributes associated with the checkpointing feature: *Checkpoint* and *Minimum\_Cpu\_Interval*. *Checkpoint* is a batch job attribute, while *Minimum\_Cpu\_Interval* is a queue attribute. An implementation that does not support checkpointing shall support the *Checkpoint* job attribute to the extent that the batch server shall maintain and pass this attribute to other servers.

4079

4080

4081

The behavior of a batch server that executes a batch job for which the value of the *Checkpoint* attribute is `CHECKPOINT_UNSPECIFIED` is implementation-defined. A batch server that executes a batch job for which the value of the *Checkpoint* attribute is `NO_CHECKPOINT` shall

4082 not checkpoint the batch job.

4083 A batch server that executes a batch job for which the value of the *Checkpoint* attribute is  
4084 CHECKPOINT\_AT\_SHUTDOWN shall checkpoint the batch job only when the batch server  
4085 accepts a request to shut down during the time when the batch job is in the RUNNING state.

4086 A batch server that executes a batch job for which the value of the *Checkpoint* attribute is  
4087 CHECKPOINT\_AT\_MIN\_CPU\_INTERVAL shall checkpoint the batch job at the interval  
4088 specified by the *Minimum\_Cpu\_Interval* attribute of the queue for which the batch job has been  
4089 selected. The *Minimum\_Cpu\_Interval* attribute shall be specified in units of CPU minutes.

4090 A batch server that executes a batch job for which the value of the *Checkpoint* attribute is an  
4091 unsigned integer shall checkpoint the batch job at an interval that is the value of either the  
4092 *Checkpoint* attribute, or the *Minimum\_Cpu\_Interval* attribute of the queue for which the batch job  
4093 has been selected, whichever is greater. Both intervals shall be in units of CPU minutes. When  
4094 the *Minimum\_Cpu\_Interval* attribute is greater than the *Checkpoint* attribute, the batch job shall  
4095 write a warning message to the standard error stream of the batch job.

#### 4096 **Error\_Path Attribute**

4097 The *Error\_Path* attribute of a running job cannot be changed by a *Modify Batch Job Request*. When  
4098 the *Join\_Path* attribute of the batch job is set to the value FALSE and the *Keep\_Files* attribute of  
4099 the batch job does not contain the value KEEP\_STD\_ERROR, a batch server that executes a batch  
4100 job shall perform one of the following actions:

- 4101 • Set the standard error stream of the session leader of the batch job to the path described by  
4102 the value of the *Error\_Path* attribute of the batch job.
- 4103 • Buffer the standard error of the session leader of the batch job until completion of the batch  
4104 job, and when the batch job exits return the contents to the destination described by the value  
4105 of the *Error\_Path* attribute of the batch job.

4106 Applications shall not rely on having access to the standard error of a batch job prior to the  
4107 completion of the batch job.

4108 When the *Error\_Path* attribute does not specify a host name, then the batch server shall retain the  
4109 standard error of the batch job on the host of execution.

4110 When the *Error\_Path* attribute does specify a host name and the *Keep\_Files* attribute does not  
4111 contain the value KEEP\_STD\_ERROR, then the final destination of the standard error of the  
4112 batch job shall be on the host whose host name is specified.

4113 If the path indicated by the value of the *Error\_Path* attribute of the batch job is a relative path, the  
4114 batch server shall expand the path relative to the home directory of the user on the host to which  
4115 the file is being returned.

4116 When the batch server buffers the standard error of the batch job and the file cannot be opened  
4117 for write upon completion of the batch job, then the server shall place the standard error in an  
4118 implementation-defined location and notify the user of the location via mail. It shall be possible  
4119 for the user to process this mail using the *mailx* utility.

4120 If a batch server that does not buffer the standard error cannot open the standard error path of  
4121 the batch job for write access, then the batch server shall abort the batch job.

**4122 Execution\_Time Attribute**

4123 A batch server shall not execute a batch job before the time represented by the value of the  
4124 *Execution\_Time* attribute of the batch job. The *Execution\_Time* attribute is defined in seconds since  
4125 the Epoch.

**4126 Hold\_Types Attribute**

4127 A batch server shall support the following hold types:

4128 s Can be set or released by a user with at least a privilege level of batch administrator  
4129 (SYSTEM).

4130 o Can be set or released by a user with at least a privilege level of batch operator  
4131 (OPERATOR).

4132 u Can be set or released by the user with at least a privilege level of user, where the user is  
4133 defined in the *Job\_Owner* attribute (USER).

4134 n Indicates that none of the *Hold\_Types* attributes are set (NO\_HOLD).

4135 An implementation may define other hold types. Any additional hold types, how they are |  
4136 specified, their internal representation, their behavior, and how they affect the behavior of other |  
4137 utilities are implementation-defined. |

4138 The value of the *Hold\_Types* attribute shall be the union of the valid hold types ('s', 'o', 'u',  
4139 and any implementation-defined hold types), or 'n'.

4140 A batch server shall not execute a batch job if the *Hold\_Types* attribute of the batch job has a  
4141 value other than NO\_HOLD. If the *Hold\_Types* attribute of the batch job has a value other than  
4142 NO\_HOLD, the batch job shall be in the HELD state.

**4143 Job\_Owner Attribute**

4144 The *Job\_Owner* attribute consists of a pair of user name and host name values of the form:

4145 username@hostname |

4146 A batch server that accepts a *Queue Batch Job Request* shall set the *Job\_Owner* attribute to a string  
4147 that is the *username@hostname* of the user who submitted the job.

**4148 Join\_Path Attribute**

4149 A batch server that executes a batch job for which the value of the *Join\_Path* attribute is TRUE  
4150 shall ignore the value of the *Error\_Path* attribute and merge the standard error of the batch job  
4151 with the standard output of the batch job.

**4152 Keep\_Files Attribute**

4153 A batch server that executes a batch job for which the value of the *Keep\_Files* attribute includes  
4154 the value KEEP\_STD\_OUTPUT shall retain the standard output of the batch job on the host  
4155 where execution occurs. The standard output shall be retained in the home directory of the user  
4156 under whose user ID the batch job is executed and the filename shall be the default filename for  
4157 the standard output as defined under the *-o* option of the *qsub* utility. The *Output\_Path* attribute  
4158 is not modified.

4159 A batch server that executes a batch job for which the value of the *Keep\_Files* attribute includes  
4160 the value KEEP\_STD\_ERROR shall retain the standard error of the batch job on the host where  
4161 execution occurs. The standard error shall be retained in the home directory of the user under  
4162 whose user ID the batch job is executed and the filename shall be the default filename for

4163 standard error as defined under the `-e` option of the `qsub` utility. The `Error_Path` attribute is not  
4164 modified.

4165 A batch server that executes a batch job for which the value of the `Keep_Files` attribute includes  
4166 values other than `KEEP_STD_OUTPUT` and `KEEP_STD_ERROR` shall retain these other files on  
4167 the host where execution occurs. These files (with implementation-defined names) shall be |  
4168 retained in the home directory of the user under whose user identifier the batch job is executed. |

#### 4169 **Mail\_Points and Mail\_Users Attributes**

4170 A batch server that executes a batch job for which one of the values of the `Mail_Points` attribute is  
4171 the value `MAIL_AT_BEGINNING` shall send a mail message to each user account listed in the  
4172 `Mail_Users` attribute of the batch job.

4173 The mail message shall contain at least the batch job identifier, queue, and server at which the  
4174 batch job currently resides, and the `Job_Owner` attribute.

#### 4175 **Output\_Path Attribute**

4176 The `Output_Path` attribute of a running job cannot be changed by a *Modify Batch Job Request*.  
4177 When the `Keep_Files` attribute of the batch job does not contain the value `KEEP_STD_OUTPUT`, a  
4178 batch server that executes a batch job shall either:

4179 • Set the standard output stream of the session leader of the batch job to the destination  
4180 described by the value of the `Output_Path` attribute of the batch job.

4181 or:

4182 • Buffer the standard output of the session leader of the batch job until completion of the batch  
4183 job, and when the batch job exits return the contents to the destination described by the value  
4184 of the `Output_Path` attribute of the batch job.

4185 When the `Output_Path` attribute does not specify a host name, then the batch server shall retain  
4186 the standard output of the batch job on the host of execution.

4187 When the `Keep_Files` attribute does not contain the value `KEEP_STD_OUTPUT` and the  
4188 `Output_Path` attribute does specify a host name, then the final destination of the standard output  
4189 of the batch job shall be on the host specified.

4190 If the path specified in the `Output_Path` attribute of the batch job is a relative path, the batch  
4191 server shall expand the path relative to the home directory of the user on the host to which the  
4192 file is being returned.

4193 Whether or not the batch server buffers the standard output of the batch job until completion of  
4194 the batch job is implementation-defined. Applications shall not rely on having access to the  
4195 standard output of a batch job prior to the completion of the batch job.

4196 When the batch server does buffer the standard output of the batch job and the file cannot be  
4197 opened for write upon completion of the batch job, then the batch server shall place the standard  
4198 output in an implementation-defined location and notify the user of the location via mail. It shall  
4199 be possible for the user to process this mail using the `mailx` utility.

4200 If a batch server that does not buffer the standard output cannot open the standard output path  
4201 of the batch job for write access, then the batch server shall abort the batch job.

**4202 Priority Attribute**

4203 A batch server implementation may choose to preferentially execute a batch job based on the  
4204 *Priority* attribute. The interpretation of the batch job *Priority* attribute by a batch server is  
4205 implementation-defined. If an implementation uses the *Priority* attribute, it shall interpret larger  
4206 values of the *Priority* attribute to mean the batch job shall be preferentially selected for execution.

**4207 Rerunable Attribute**

4208 A batch job that began execution but did not complete, because the batch server either shut  
4209 down or terminated abnormally, shall be requeued if the *Rerunable* attribute of the batch job has  
4210 the value TRUE.

4211 If a batch job, which was requeued after beginning execution but prior to completion, has a valid  
4212 checkpoint file and the batch server supports checkpointing, then the batch job shall be restarted  
4213 from the last valid checkpoint.

4214 If the batch job cannot be restarted from a checkpoint, then when a batch job has a *Rerunable*  
4215 attribute value of TRUE and was requeued after beginning execution but prior to completion,  
4216 the batch server shall place the batch job into execution at the beginning of the job.

4217 When a batch job has a *Rerunable* attribute value other than TRUE and was requeued after  
4218 beginning execution but prior to completion, and the batch job cannot be restarted from a  
4219 checkpoint, then the batch server shall abort the batch job.

**4220 Resource\_List Attribute**

4221 A batch server that executes a batch job shall establish the resource limits of the session leader of  
4222 the batch job according to the values of the *Resource\_List* attribute of the batch job. Resource  
4223 limits shall be enforced by an implementation-defined method.

**4224 Shell\_Path\_List Attribute**

4225 The *Shell\_Path\_List* job attribute consists of a list of pairs of pathname and host name values. The  
4226 host name component can be omitted, in which case the pathname serves as the default  
4227 pathname when a batch server cannot find the name of the host on which it is running in the list.

4228 A batch server that executes a batch job shall select, from the value of the *Shell\_Path\_List*  
4229 attribute of the batch job, a pathname where the shell to execute the batch job shall be found. The  
4230 batch server shall select the pathname, in order of preference, according to the following  
4231 methods:

- 4232 • Select the pathname that contains the name of the host on which the batch server is running.
- 4233 • Select the pathname for which the host name has been omitted.
- 4234 • Select the pathname for the login shell of the user under which the batch job is to execute.

4235 If the shell path value selected is an invalid pathname, the batch server shall abort the batch job.

4236 If the value of the selected pathname from the *Shell\_Path\_List* attribute of the batch job  
4237 represents a partial path, the batch server shall expand the path relative to a path that is  
4238 implementation-defined.

4239 The batch server that executes the batch job shall execute the program that was selected from the  
4240 *Shell\_Path\_List* attribute of the batch job. The batch server shall pass the path to the script of the  
4241 batch job as the first argument to the shell program.

4242 **User\_List Attribute**

4243 The *User\_List* job attribute consists of a list of pairs of user name and host name values. The host  
 4244 name component can be omitted, in which case the user name serves as a default when a batch  
 4245 server cannot find the name of the host on which it is running in the list.

4246 A batch server that executes a batch job shall select, from the value of the *User\_List* attribute of  
 4247 the batch job, a user name under which to create the session leader. The server shall select the  
 4248 user name, in order of preference, according to the following methods:

- 4249 • Select the user name of a value that contains the name of the host on which the batch server  
 4250 executes.
- 4251 • Select the user name of a value for which the host name has been omitted.
- 4252 • Select the user name from the *Job\_Owner* attribute of the batch job.

4253 **Variable\_List Attribute**

4254 A batch server that executes a batch job shall create, in the environment of the session leader of  
 4255 the batch job, each environment variable listed in the *Variable\_List* attribute of the batch job, and  
 4256 set the value of each such environment variable to that of the corresponding variable in the  
 4257 variable list.

4258 **3.2.2.2 Batch Job Routing**

4259 To route a batch job is to select a queue from a list and move the batch job to that queue.

4260 A batch server that has routing queues, which have been started, shall route the jobs in the  
 4261 routing queues owned by the batch server. A batch server may delay the routing of a batch job. |  
 4262 The algorithm for selecting a batch job and the queue to which it will be routed is |  
 4263 implementation-defined.

4264 When a routing queue has multiple possible destinations specified, then the precedence of the |  
 4265 destinations is implementation-defined. |

4266 A batch server that routes a batch job to a queue at another server shall move the batch job into  
 4267 the target queue with a *Queue Batch Job Request*.

4268 If the target server rejects the *Queue Batch Job Request*, the routing server shall retry routing the  
 4269 batch job or abort the batch job. A batch server that retries failed routings shall provide a means  
 4270 for the batch administrator to specify the number of retries and the minimum period of time  
 4271 between retries. The means by which an administrator specifies the number of retries and the  
 4272 delay between retries is implementation-defined. When the number of retries specified by the  
 4273 batch administrator has been exhausted, the batch server shall abort the batch job and perform  
 4274 the functions of *Batch Job Exit*; see Section 3.2.2.3.

4275 **3.2.2.3 Batch Job Exit**

4276 For each job in the EXITING state, the batch server that exited the batch job shall perform the  
 4277 following deferred services in the order specified:

- 4278 1. If buffering standard error, move that file into the location specified by the *Error\_Path*  
 4279 attribute of the batch job.
- 4280 2. If buffering standard output, move that file into the location specified by the *Output\_Path*  
 4281 attribute of the batch job.
- 4282 3. If the *Mail\_Points* attribute of the batch job includes MAIL\_AT\_EXIT, send mail to the users  
 4283 listed in the *Mail\_Users* attribute of the batch job. The mail message shall contain at least

4284 the batch job identifier, queue, and server at which the batch job currently resides, and the  
4285 *Job\_Owner* attribute.

4286 4. Remove the batch job from the queue.

4287 If a batch server that buffers the standard error output cannot return the standard error file to  
4288 the standard error path at the time the batch job exits, the batch server shall do one of the  
4289 following:

- 4290 • Mail the standard error file to the batch job owner.
- 4291 • Save the standard error file and mail the location and name of the file where the standard  
4292 error is stored to the batch job owner.
- 4293 • Save the standard error file and notify the user by other implementation-defined means. |

4294 If a batch server that buffers the standard output cannot return the standard output file to the  
4295 standard output path at the time the batch job exits, the batch server shall do one of the  
4296 following:

- 4297 • Mail the standard output file to the batch job owner.
- 4298 • Save the standard output file and mail the location and name of the file where the standard  
4299 output is stored to the batch job owner.
- 4300 • Save the standard output file and notify the user by other implementation-defined means. |

4301 At the conclusion of job exit processing, the batch job is no longer managed by a batch server.

#### 4302 3.2.2.4 *Batch Server Restart*

4303 A batch server that has been either shutdown or terminated abnormally, and has returned to  
4304 operation, is said to have *restarted*.

4305 Upon restarting, a batch server shall requeue those jobs managed by the batch server that were  
4306 in the RUNNING state at the time the batch server shut down and for which the *Rerunable*  
4307 attribute of the batch job has the value TRUE.

4308 Queues are defined to be non-volatile. A batch server shall store the content of queues that it  
4309 controls in such a way that server and system shutdowns do not erase the content of the queues.

#### 4310 3.2.2.5 *Batch Job Abort*

4311 A batch server that cannot perform a deferred service for a batch job shall abort the batch job.

4312 A batch server that aborts a batch job shall perform the following services:

- 4313 • Delete the batch job from the queue in which it resides.
- 4314 • If the *Mail\_Points* attribute of the batch job includes the value MAIL\_AT\_ABORT, send mail  
4315 to the users listed in the value of the *Mail\_Users* attribute of the job. The mail message shall  
4316 contain at least the batch job identifier, queue, and server at which the batch job currently  
4317 resides, the *Job\_Owner* attribute, and the reason for the abort.
- 4318 • If the batch job was in the RUNNING state, terminate the session leader of the executing job  
4319 by sending the session leader a SIGKILL, place the batch job in the EXITING state, and  
4320 perform the actions of *Batch Job Exit*. |

4321 **3.2.3 Requested Batch Services**

4322 This section describes the services provided by batch servers in response to requests from  
 4323 clients. Table 3-5 summarizes the current set of batch service requests and for each gives its type  
 4324 (deferred or not) and whether it is an optional function.

4325 **Table 3-5** Batch Services Summary

Batch Service	Deferred	Optional
<i>Batch Job Execution</i>	Yes	No
<i>Batch Job Routing</i>	Yes	No
<i>Batch Job Exit</i>	Yes	No
<i>Batch Server Restart</i>	Yes	No
<i>Batch Job Abort</i>	Yes	No
<i>Delete Batch Job Request</i>	No	No
<i>Hold Batch Job Request</i>	No	No
<i>Batch Job Message Request</i>	No	Yes
<i>Batch Job Status Request</i>	No	No
<i>Locate Batch Job Request</i>	No	Yes
<i>Modify Batch Job Request</i>	No	No
<i>Move Batch Job Request</i>	No	No
<i>Queue Batch Job Request</i>	No	No
<i>Batch Queue Status Request</i>	No	No
<i>Release Batch Job Request</i>	No	No
<i>Rerun Batch Job Request</i>	No	No
<i>Select Batch Jobs Request</i>	No	No
<i>Server Shutdown Request</i>	No	No
<i>Server Status Request</i>	No	No
<i>Signal Batch Job Request</i>	No	No
<i>Track Batch Job Request</i>	No	Yes

4348 If a request is rejected because the batch client is not authorized to perform the action, the batch  
 4349 server shall return the same status as when the batch job does not exist.

4350 **3.2.3.1 Delete Batch Job Request**

4351 A batch job is defined to have been deleted when it has been removed from the queue in which it  
 4352 resides and not instantiated in another queue. A client requests that the server that manages a  
 4353 batch job delete the batch job. Such a request is called a *Delete Batch Job Request*.

4354 A batch server shall reject a *Delete Batch Job Request* if any of the following statements are true:

- 4355 • The user of the batch client is not authorized to delete the designated job.
- 4356 • The designated job is not managed by the batch server.
- 4357 • The designated job is in a state inconsistent with the delete request.

4358 A batch server may reject a *Delete Batch Job Request* for other implementation-defined reasons.  
 4359 The method used to determine whether the user of a client is authorized to perform the  
 4360 requested action is implementation-defined.

4361 A batch server requested to delete a batch job shall delete the batch job if the batch job exists and  
 4362 is not in the EXITING state.

4363 A batch server that deletes a batch job in the RUNNING state shall send a SIGKILL signal to the  
 4364 session leader of the batch job. It is implementation-defined whether additional signals are sent

- 4365 to the session leader of the job prior to sending the SIGKILL signal. |
- 4366 A batch server that deletes a batch job in the RUNNING state shall place the batch job in the  
4367 EXITING state after it has killed the session leader of the batch job and shall perform the actions |  
4368 of *Batch Job Exit*. |
- 4369 **3.2.3.2 Hold Batch Job Request**
- 4370 A batch client can request that the batch server add one or more holds to a batch job. Such a  
4371 request is called a *Hold Batch Job Request*.
- 4372 A batch server shall reject a *Hold Batch Job Request* if any of the following statements are true:
- 4373 • The batch server does not support one or more of the requested holds to be added to the  
4374 batch job.
  - 4375 • The user of the batch client is not authorized to add one or more of the requested holds to the  
4376 batch job.
  - 4377 • The batch server does not manage the specified job.
  - 4378 • The designated job is in the EXITING state.
- 4379 A batch server may reject a *Hold Batch Job Request* for other implementation-defined reasons. The |  
4380 method used to determine whether the user of a client is authorized to perform the requested |  
4381 action is implementation-defined. |
- 4382 A batch server that accepts a *Hold Batch Job Request* for a batch job in the RUNNING state shall  
4383 place a hold on the batch job. The effects, if any, the hold will have on a batch job in the |  
4384 RUNNING state are implementation-defined. |
- 4385 A batch server that accepts a *Hold Batch Job Request* shall add each type of hold listed in the *Hold*  
4386 *Batch Job Request*, that is not already present, to the value of the *Hold\_Types* attribute of the batch  
4387 job.
- 4388 **3.2.3.3 Batch Job Message Request**
- 4389 *Batch Job Message Request* is an optional feature of batch servers. If an implementation supports  
4390 *Batch Job Message Request*, the statements in this section apply and the configuration variable  
4391 POSIX2\_PBS\_MESSAGE shall be set to 1.
- 4392 A batch client can request that a batch server write a message into certain output files of a batch  
4393 job. Such a request is called a *Batch Job Message Request*.
- 4394 A batch server shall reject a *Batch Job Message Request* if any of the following statements are true:
- 4395 • The batch server does not support sending messages to jobs.
  - 4396 • The user of the batch client is not authorized to post a message to the designated job.
  - 4397 • The designated job does not exist on the batch server.
  - 4398 • The designated job is not in the RUNNING state.
- 4399 A batch server may reject a *Batch Job Message Request* for other implementation-defined reasons. |  
4400 The method used to determine whether the user of a client is authorized to perform the |  
4401 requested action is implementation-defined. |
- 4402 A batch server that accepts a *Batch Job Message Request* shall write the message sent by the batch  
4403 client into the files indicated by the batch client.

4404 3.2.3.4 *Batch Job Status Request*

4405 A batch client can request that a batch server respond with the status and attributes of a batch  
4406 job. Such a request is called a *Batch Job Status Request*.

4407 A batch server shall reject a *Batch Job Status Request* if any of the following statements are true:

- 4408 • The user of the batch client is not authorized to query the status of the designated job.
- 4409 • The designated job is not managed by the batch server.

4410 A batch server may reject a *Batch Job Status Request* for other implementation-defined reasons. |  
4411 The method used to determine whether the user of a client is authorized to perform the |  
4412 requested action is implementation-defined. |

4413 A batch server that accepts a *Batch Job Status Request* shall return a *Batch Job Status Message* to the  
4414 batch client.

4415 A batch server may return other information in response to a *Batch Job Status Request*.

4416 3.2.3.5 *Locate Batch Job Request*

4417 *Locate Batch Job Request* is an optional feature of batch servers. If an implementation supports  
4418 *Locate Batch Job Request*, the statements in this section apply and the configuration variable  
4419 POSIX2\_PBS\_LOCATE shall be set to 1.

4420 A batch client can ask a batch server to respond with the location of a batch job that was created  
4421 by the batch server. Such a request is called a *Locate Batch Job Request*.

4422 A batch server that accepts a *Locate Batch Job Request* shall return a *Batch Job Location Message* to  
4423 the batch client.

4424 A batch server may reject a *Locate Batch Job Request* for a batch job that was not created by that  
4425 server.

4426 A batch server may reject a *Locate Batch Job Request* for a batch job that is no longer managed by  
4427 that server; that is, for a batch job that is not in a queue owned by that server.

4428 A batch server may reject a *Locate Batch Job Request* for other implementation-defined reasons. |

4429 3.2.3.6 *Modify Batch Job Request*

4430 Batch clients modify (alter) the attributes of a batch job by making a request to the server that  
4431 manages the batch job. Such a request is called a *Modify Batch Job Request*.

4432 A batch server shall reject a *Modify Batch Job Request* if any of the following statements are true:

- 4433 • The user of the batch client is not authorized to make the requested modification to the batch  
4434 job.
- 4435 • The designated job is not managed by the batch server.
- 4436 • The requested modification is inconsistent with the state of the batch job.
- 4437 • An unrecognized resource is requested for a batch job in an execution queue.

4438 A batch server may reject a *Modify Batch Job Request* for other implementation-defined reasons. |  
4439 The method used to determine whether the user of a client is authorized to perform the |  
4440 requested action is implementation-defined. |

4441 A batch server that accepts a *Modify Batch Job Request* shall modify all the specified attributes of  
4442 the batch job. A batch server that rejects a *Modify Batch Job Request* shall modify none of the  
4443 attributes of the batch job.

4444 If the servicing by a batch server of an otherwise valid request would result in no change, then  
4445 the batch server shall indicate successful completion of the request.

#### 4446 3.2.3.7 *Move Batch Job Request*

4447 A batch client can request that a batch server move a batch job to another destination. Such a  
4448 request is called a *Move Batch Job Request*.

4449 A batch server shall reject a *Move Batch Job Request* if any of the following statements are true:

- 4450 • The user of the batch client is not authorized to remove the designated job from the queue in  
4451 which the batch job resides.
- 4452 • The user of the batch client is not authorized to move the designated job to the destination.
- 4453 • The designated job is not managed by the batch server.
- 4454 • The designated job is in the EXITING state.
- 4455 • The destination is inaccessible.

4456 A batch server can reject a *Move Batch Job Request* for other implementation-defined reasons. The  
4457 method used to determine whether the user of a client is authorized to perform the requested  
4458 action is implementation-defined.

4459 A batch server that accepts a *Move Batch Job Request* shall perform the following services:

- 4460 • Queue the designated job at the destination.
- 4461 • Remove the designated job from the queue in which the batch job resides.

4462 If the destination resides on another batch server, the batch server shall queue the batch job at  
4463 the destination by sending a *Queue Batch Job Request* to the other server. If the *Queue Batch Job*  
4464 *Request* fails, the batch server shall reject the *Move Batch Job Request*. If the *Queue Batch Job Request*  
4465 succeeds, the batch server shall remove the batch job from its queue.

4466 The batch server shall not modify any attributes of the batch job.

#### 4467 3.2.3.8 *Queue Batch Job Request*

4468 A batch queue is controlled by one and only one batch server. A batch server is said to own the  
4469 queues that it controls. Batch clients make requests of batch servers to have jobs queued. Such a  
4470 request is called a *Queue Batch Job Request*.

4471 A batch server requested to queue a batch job for which the queue is not specified shall select an  
4472 implementation-defined queue for the batch job. Such a queue is called the *default queue* of the  
4473 batch server. The implementation shall provide the means for a batch administrator to specify  
4474 the default queue. The queue, whether specified or defaulted, is called the *target queue*.

4475 A batch server shall reject a *Queue Batch Job Request* if any of the following statements are true:

- 4476 • The client is not authorized to create a batch job in the target queue.
- 4477 • The request specifies a queue that does not exist on the batch server.
- 4478 • The target queue is an execution queue and the batch server cannot satisfy a resource  
4479 requirement of the batch job.
- 4480 • The target queue is an execution queue and an unrecognized resource is requested.
- 4481 • The target queue is an execution queue, the batch server does not support checkpointing, and  
4482 the value of the *Checkpoint* attribute of the batch job is not NO\_CHECKPOINT.

- 4483           • The job requires access to a user identifier that the batch client is not authorized to access.

4484           A batch server may reject a *Queue Batch Job Request* for other implementation-defined reasons. |

4485           A batch server that accepts a *Queue Batch Job Request* for a batch job for which the  
4486           PBS\_O\_QUEUE value is missing from the value of the *Variable\_List* attribute of the batch job  
4487           shall add that variable to the list and set the value to the name of the target queue. Once set, no  
4488           server shall change the value of PBS\_O\_QUEUE, even if the batch job is moved to another  
4489           queue.

4490           A batch server that accepts a *Queue Batch Job Request* for a batch job for which the PBS\_JOBID  
4491           value is missing from the value of the *Variable\_List* attribute shall add that variable to the list and  
4492           set the value to the batch job identifier assigned by the server in the format:

4493                 sequence\_number.server |

4494           A batch server that accepts a *Queue Batch Job Request* for a batch job for which the  
4495           PBS\_JOBNAME value is missing from the value of the *Variable\_List* attribute of the batch job  
4496           shall add that variable to the list and set the value to the *Job\_Name* attribute of the batch job.

### 4497 3.2.3.9 *Batch Queue Status Request*

4498           A batch client can request that a batch server respond with the status and attributes of a queue.  
4499           Such a request is called a *Batch Queue Status Request*.

4500           A batch server shall reject a *Batch Queue Status Request* if any of the following statements are true:

- 4501           • The user of the batch client is not authorized to query the status of the designated queue.  
4502           • The designated queue does not exist on the batch server.

4503           A batch server may reject a *Batch Queue Status Request* for other implementation-defined reasons. |  
4504           The method used to determine whether the user of a client is authorized to perform the |  
4505           requested action is implementation-defined. |

4506           A batch server that accepts a *Batch Queue Status Request* shall return a *Batch Queue Status Reply* to  
4507           the batch client.

### 4508 3.2.3.10 *Release Batch Job Request*

4509           A batch client can request that server remove one or more holds from a batch job. Such a request  
4510           is called a *Release Batch Job Request*.

4511           A batch server shall reject a *Release Batch Job Request* if any of the following statements are true:

- 4512           • The user of the batch client is not authorized to remove one or more of the requested holds  
4513           from the batch job.  
4514           • The batch server does not manage the specified job.

4515           A batch server may reject a *Release Batch Job Request* for other implementation-defined reasons. |  
4516           The method used to determine whether the user of a client is authorized to perform the |  
4517           requested action is implementation-defined. |

4518           A batch server that accepts a *Release Batch Job Request* shall remove each type of hold listed in the  
4519           *Release Batch Job Request*, that is present, from the value of the *Hold\_Types* attribute of the batch  
4520           job.

4521 3.2.3.11 *Rerun Batch Job Request*

4522 To rerun a batch job is to kill the session leader of the batch job and leave the batch job eligible  
4523 for re-execution. A batch client can request that a batch server rerun a batch job. Such a request is  
4524 called *Rerun Batch Job Request*.

4525 A batch server shall reject a *Rerun Batch Job Request* if any of the following statements are true:

- 4526 • The user of the batch client is not authorized to rerun the designated job.
- 4527 • The *Rerunable* attribute of the designated job has the value FALSE.
- 4528 • The designated job is not in the RUNNING state.
- 4529 • The batch server does not manage the designated job.

4530 A batch server may reject a *Rerun Batch Job Request* for other implementation-defined reasons. |  
4531 The method used to determine whether the user of a client is authorized to perform the |  
4532 requested action is implementation-defined. |

4533 A batch server that rejects a *Rerun Batch Job Request* shall in no way modify the execution of the  
4534 batch job.

4535 A batch server that accepts a request to rerun a batch job shall perform the following services:

- 4536 • Requeue the batch job in the execution queue in which it was executing.
- 4537 • Send a SIGKILL signal to the process group of the session leader of the batch job.

4538 An implementation may indicate to the batch job owner that the batch job has been rerun. |  
4539 Whether and how the batch job owner is notified that a batch job is rerun is implementation- |  
4540 defined. |

4541 A batch server that reruns a batch job may send other implementation-defined signals to the |  
4542 session leader of the batch job prior to sending the SIGKILL signal. |

4543 A batch server may preferentially select a rerun job for execution. Whether rerun jobs shall be |  
4544 selected for execution before other jobs is implementation-defined. |

4545 3.2.3.12 *Select Batch Jobs Request*

4546 A batch client can request from a batch server a list of jobs managed by that server that match a  
4547 list of selection criteria. Such a request is called a *Select Batch Jobs Request*. All the batch jobs  
4548 managed by the batch server that receives the request are candidates for selection.

4549 A batch server that accepts a *Select Batch Jobs Request* shall return a list of zero or more job  
4550 identifiers that correspond to jobs that meet the selection criteria.

4551 If the batch client is not authorized to query the status of a batch job, the batch server shall not  
4552 select the batch job.

4553 3.2.3.13 *Server Shutdown Request*

4554 A batch server is defined to have shut down when it does not respond to requests from clients  
4555 and does not perform deferred services for jobs. A batch client can request that a batch server  
4556 shut down. Such a request is called a *Server Shutdown Request*.

4557 A batch server shall reject a *Server Shutdown Request* from a client that is not authorized to shut |  
4558 down the batch server. The method used to determine whether the user of a client is authorized |  
4559 to perform the requested action is implementation-defined. |

4560 A batch server may reject a *Server Shutdown Request* for other implementation-defined reasons. |  
4561 The reasons for which a *Server Shutdown Request* may be rejected are implementation-defined. |

4562 At server shutdown, a batch server shall do, in order of preference, one of the following:

- 4563 • If checkpointing is implemented and the batch job is checkpointable, then checkpoint the  
4564 batch job and requeue it.
- 4565 • If the batch job is rerunnable, then requeue the batch job to be rerun (restarted from the  
4566 beginning).
- 4567 • Abort the batch job.

#### 4568 3.2.3.14 *Server Status Request*

4569 A batch client can request that a batch server respond with the status and attributes of the batch  
4570 server. Such a request is called a *Server Status Request*.

4571 A batch server shall reject a *Server Status Request* if the following statement is true:

- 4572 • The user of the batch client is not authorized to query the status of the designated server.

4573 A batch server may reject a *Server Status Request* for other implementation-defined reasons. The |  
4574 method used to determine whether the user of a client is authorized to perform the requested |  
4575 action is implementation-defined. |

4576 A batch server that accepts a *Server Status Request* shall return a *Server Status Reply* to the batch  
4577 client.

#### 4578 3.2.3.15 *Signal Batch Job Request*

4579 A batch client can request that a batch server signal the session leader of a batch job. Such a  
4580 request is called a *Signal Batch Job Request*.

4581 A batch server shall reject a *Signal Batch Job Request* if any of the following statements are true:

- 4582 • The user of the batch client is not authorized to signal the batch job.
- 4583 • The job is not in the RUNNING state.
- 4584 • The batch server does not manage the designated job.
- 4585 • The requested signal is not supported by the implementation.

4586 A batch server may reject a *Signal Batch Job Request* for other implementation-defined reasons. |  
4587 The method used to determine whether the user of a client is authorized to perform the |  
4588 requested action is implementation-defined. |

4589 A batch server that accepts a request to signal a batch job shall send the signal requested by the  
4590 batch client to the process group of the session leader of the batch job.

#### 4591 3.2.3.16 *Track Batch Job Request*

4592 *Track Batch Job Request* is an optional feature of batch servers. If an implementation supports  
4593 *Track Batch Job Request*, the statements in this section apply and the configuration variable  
4594 POSIX2\_PBS\_TRACK shall be set to 1.

4595 *Track Batch Job Request* provides a method for tracking the current location of a batch job. Clients  
4596 may use the tracking information to determine the batch server that should receive a batch  
4597 server request.

4598 If *Track Batch Job Request* is supported by a batch server, then when the batch server queues a  
 4599 batch job as a result of a *Queue Batch Job Request*, and the batch server is not the batch server that  
 4600 created the batch job, the batch server shall send a *Track Batch Job Request* to the batch server that  
 4601 created the job.

4602 If *Track Batch Job Request* is supported by a batch server, then the *Track Batch Job Request* may also  
 4603 be sent to other servers as a backup to the primary server. The method by which backup servers  
 4604 are specified is implementation-defined.

4605 If *Track Batch Job Request* is supported by a batch server that receives a *Track Batch Job Request*,  
 4606 then the batch server shall record the current location of the batch job as contained in the  
 4607 request.

### 4608 **3.3 Common Behavior for Batch Environment Utilities**

#### 4609 **3.3.1 Batch Job Identifier**

4610 A utility shall recognize *job\_identifiers* of the format:

```
4611 [sequence_number][.server_name][@server]
```

4612 where:

4613 *sequence\_number* An integer that, when combined with *server\_name*, provides a batch job  
 4614 identifier that is unique within the batch system.

4615 *server\_name* The name of the batch server to which the batch job was originally submitted.

4616 *server* The name of the batch server that is currently managing the batch job.

4617 If the application omits the batch *server\_name* portion of a batch job identifier, a utility shall use  
 4618 the name of a default batch server.

4619 If the application omits the batch *server* portion of a batch job identifier, a utility shall use:

- 4620 • The batch server indicated by *server\_name*, if present.
- 4621 • The name of the default batch server.
- 4622 • The name of the batch server that is currently managing the batch job.

4623 If only *@server* is specified, then the status of all jobs owned by the user on the requested server  
 4624 is listed.

4625 The means by which a utility determines the default batch server is implementation-defined.

4626 If the application presents the batch *server* portion of a batch job identifier to a utility, the utility  
 4627 shall send the request to the specified server.

4628 A strictly conforming application shall use the syntax described for the job identifier. Whenever  
 4629 a batch job identifier is specified whose syntax is not recognized by an implementation, then a  
 4630 message for each error that occurs shall be written to standard error and the utility shall exit  
 4631 with an exit status greater than zero.

4632 When a batch job identifier is supplied as an argument to a batch utility and the *server\_name*  
 4633 portion of the batch job identifier is omitted, then the utility shall use the name of the default  
 4634 batch server.

4635 When a batch job identifier is supplied as an argument to a batch utility and the batch *server*  
 4636 portion of the batch job identifier is omitted, then the utility shall use either:

4637           • The name of the default batch server

4638           or:

4639           • The name of the batch server that is currently managing the batch job

4640           When a batch job identifier is supplied as an argument to a batch utility and the *batch server*  
4641           portion of the batch job identifier is specified, then the utility shall send the required *Batch Server*  
4642           *Request* to the specified server.

### 4643 3.3.2 Destination

4644           The utility shall recognize a *destination* of the format:

4645           [*queue*][*@server*]

4646           where:

4647           *queue*           The name of a valid execution or routing queue at the batch server denoted by  
4648           *@server*, defined as a string of up to 15 alphanumeric characters in the portable  
4649           character set (see the Base Definitions volume of IEEE Std 1003.1-200x, Section  
4650           6.1, Portable Character Set) where the first character is alphabetic.

4651           *server*           The name of a batch server, defined as a string of alphanumeric characters in  
4652           the portable character set.

4653           If the application omits the *batch server* portion of a destination, then the utility shall use either:

4654           • The name of the default batch server

4655           or:

4656           • The name of the batch server that is currently managing the batch job

4657           The means by which a utility determines the default batch server is implementation-defined.

4658           If the application omits the *queue* portion of a destination, then the utility shall use the name of  
4659           the default queue at the batch server chosen. The means by which a batch server determines its  
4660           default queue is implementation-defined. If a destination is specified in the *queue@server* form,  
4661           then the utility shall use the specified queue at the specified server.

4662           A strictly conforming application shall use the syntax described for a destination. Whenever a  
4663           destination is specified whose syntax is not recognized by an implementation, then a message  
4664           shall be written to standard error and the utility shall exit with an exit status greater than zero.

### 4665 3.3.3 Multiple Keyword-Value Pairs

4666           For each option that can have multiple keyword-value pair arguments, the following rules shall  
4667           apply. Examples of options that can have list-oriented option-arguments are *-u value@keyword*  
4668           and *-l keyword=value*.

4669           1. If a batch utility is presented with a list-oriented option-argument for which a keyword has  
4670           a corresponding value that begins with a single or double quote, then the utility shall stop  
4671           interpreting the input stream for delimiters until a second single or double quote,  
4672           respectively, is encountered. This feature allows some flexibility for a comma (',') or  
4673           equals sign ('=') to be part of the value string for a particular keyword; for example:

4674           keywd1='val1,val2',keywd2="val3,val4"

4675           **Note:**       This may require the user to escape the quotes as in the following command:

4676                                   foo -xkeywd1=\ 'val1, val2\ ',keywd2=\"val3, val4\"

- 4677                                   2. If a batch server is presented with a list-oriented attribute that has a keyword that was  
4678                                   encountered earlier in the list, then the later entry for that keyword shall replace the earlier  
4679                                   entry.
- 4680                                   3. If a batch server is presented with a list-oriented attribute that has a keyword without any  
4681                                   corresponding value of the form *keyword=* or *@keyword* and the same keyword was  
4682                                   encountered earlier in the list, then the prior entry for that keyword shall be ignored by the  
4683                                   batch server.
- 4684                                   4. If a batch utility is expecting a list-oriented option-argument entry of the form  
4685                                   *keyword=value*, but is presented with an entry of the form *keyword* without any  
4686                                   corresponding *value*, then the entry shall be treated as though a default value of NULL was  
4687                                   assigned (that is, *keyword=NULL*) for entry parsing purposes. The utility shall include only  
4688                                   the keyword, not the NULL value, in the associated job attribute.
- 4689                                   5. If a batch utility is expecting a list-oriented option-argument entry of the form  
4690                                   *value@keyword*, but is presented with an entry of the form *value* without any corresponding  
4691                                   *keyword*, then the entry shall be treated as though a keyword of NULL was assigned (that  
4692                                   is, *value@NULL*) for entry parsing purposes. The utility shall include only the value, not  
4693                                   the NULL keyword, in the associated job attribute.
- 4694                                   6. A batch server shall accept a list-oriented attribute that has multiple occurrences of the  
4695                                   same keyword, interpreting the keywords, in order, with the last value encountered taking  
4696                                   precedence over prior instances of the same keyword. This rule allows, but does not  
4697                                   require, a batch utility to preprocess the attribute to remove duplicate keywords.
- 4698                                   7. If a batch utility is presented with multiple list-oriented option-arguments on the  
4699                                   command line or in script directives, or both, for a single option, then the utility shall  
4700                                   concatenate, in order, any command line keyword and value pairs to the end of any  
4701                                   directive keyword and value pairs separated by a single comma to produce a single string  
4702                                   that is an equivalent, valid option-argument. The resulting string shall be assigned to the  
4703                                   associated attribute of the batch job (after optionally removing duplicate entries as  
4704                                   described in item 6. |

# *Chapter 4* *Utilities*

4705

4706

This chapter contains the definitions of the utilities, as follows:

4707

- Mandatory utilities that are present on every conformant system

4708

4709

4710

- Optional utilities that are present only on systems supporting the associated option; see Section 1.8.1 (on page 2210) for information on the options in this volume of IEEE Std 1003.1-200x

## 4711 NAME

4712 admin — create and administer SCCS files (DEVELOPMENT)

## 4713 SYNOPSIS

4714 xSI admin -i[name][-n][-a login][-d flag][-e login][-f flag][-m mrlist] |  
4715 [-r rel][-t[name][-y[comment]]] newfile |4716 admin -n[-a login][-d flag][-e login][-f flag][-m mrlist][-t[name]] |  
4717 [-y[comment]] newfile ... |

4718 admin [-a login][-d flag][-m mrlist][-r rel][-t[name]] file ... |

4719 admin -h file ...

4720 admin -z file ...

4721

## 4722 DESCRIPTION

4723 The *admin* utility shall create new SCCS files or change parameters of existing ones. If a named  
4724 file does not exist, it shall be created, and its parameters shall be initialized according to the  
4725 specified options. Parameters not initialized by an option shall be assigned a default value. If a  
4726 named file does exist, parameters corresponding to specified options shall be changed, and other  
4727 parameters shall be left as is.4728 All SCCS filenames supplied by the application shall be of the form *s.filename*. New SCCS files  
4729 shall be given read-only permission mode. Write permission in the parent directory is required  
4730 to create a file. All writing done by *admin* shall be to a temporary *x-file*, named *x.filename* (see *get*)  
4731 created with read-only mode if *admin* is creating a new SCCS file, or created with the same mode  
4732 as that of the SCCS file if the file already exists. After successful execution of *admin*, the SCCS file  
4733 shall be removed (if it exists), and the *x-file* shall be renamed with the name of the SCCS file. This  
4734 ensures that changes are made to the SCCS file only if no errors occur.4735 The *admin* utility shall also use a transient lock file (named *z.filename*), which is used to prevent  
4736 simultaneous updates to the SCCS file; see *get* (on page 2675).

## 4737 OPTIONS

4738 The *admin* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section  
4739 12.2, Utility Syntax Guidelines, except that the *-i*, *-t*, and *-y* options have optional option-  
4740 arguments. These optional option-arguments shall not be presented as separate arguments. The  
4741 following options are supported:4742 **-n** Create a new SCCS file. When *-n* is used without *-i*, the SCCS file shall be created  
4743 with control information but without any file data.4744 **-i[name]** Specify the *name* of a file from which the text for a new SCCS file shall be taken.  
4745 The text constitutes the first delta of the file (see the *-r* option for delta numbering  
4746 scheme). If the *-i* option is used, but the *name* option-argument is omitted, the text  
4747 shall be obtained by reading the standard input. If this option is omitted, the SCCS  
4748 file shall be created with control information but without any file data. The *-i*  
4749 option implies the *-n* option. |4750 **-r SID** Specify the SID of the initial delta to be inserted. This SID shall be a trunk SID; that |  
4751 is, the branch and sequence numbers shall be zero or missing. The level number is |  
4752 optional, and defaults to 1. |4753 **-t[name]** Specify the *name* of a file from which descriptive text for the SCCS file shall be  
4754 taken. In the case of existing SCCS files (neither *-i* nor *-n* is specified):

- 4755                   • A **-t** option without a *name* option-argument shall cause the removal of  
4756                   descriptive text (if any) currently in the SCCS file.
- 4757                   • A **-t** option with a *name* option-argument shall cause the text (if any) in the  
4758                   named file to replace the descriptive text (if any) currently in the SCCS file.
- 4759           **-f flag**   Specify a *flag*, and, possibly, a value for the *flag*, to be placed in the SCCS file.  
4760                   Several **-f** options may be supplied on a single *admin* command line. |  
4761                   Implementations shall recognize the following flags and associated values: |
- 4762                   **b**           Allow use of the **-b** option on a *get* command to create branch deltas.
- 4763                   **cceil**       Specify the highest release (that is, ceiling), a number less than or equal to  
4764                   9 999, which may be retrieved by a *get* command for editing. The default  
4765                   value for an unspecified **c** flag shall be 9 999.
- 4766                   **ffloor**      Specify the lowest release (that is, floor), a number greater than 0 but less  
4767                   than 9 999, which may be retrieved by a *get* command for editing. The  
4768                   default value for an unspecified **f** flag shall be 1.
- 4769                   **dSID**       Specify the default delta number (SID) to be used by a *get* command.
- 4770                   **istr**       Treat the “No ID keywords” message issued by *get* or *delta* as a fatal  
4771                   error. In the absence of this flag, the message is only a warning. The  
4772                   message is issued if no SCCS identification keywords (see *get* (on page  
4773                   2675)) are found in the text retrieved or stored in the SCCS file. If a value  
4774                   is supplied, the application shall ensure that the keywords exactly match  
4775                   the given string; however, the string shall contain a keyword, and no  
4776                   embedded <newline>s.
- 4777                   **j**           Allow concurrent *get* commands for editing on the same SID of an SCCS  
4778                   file. This allows multiple concurrent updates to the same version of the  
4779                   SCCS file.
- 4780                   **l!list**      Specify a *list* of releases to which deltas can no longer be made (that is, *get*  
4781                   **-e** against one of these locked releases fails). Conforming applications  
4782                   shall use the following syntax to specify a *list*. Implementations may  
4783                   accept additional forms as an extension:
- 4784                   <list> ::= a | <range-list>  
4785                   <range-list> ::= <range> | <range-list>, <range>  
4786                   <range> ::= <SID>
- 4787                   The character *a* in the *list* shall be equivalent to specifying all releases for  
4788                   the named SCCS file. The non-terminal <SID> in range shall be the delta  
4789                   number of an existing delta associated with the SCCS file.
- 4790                   **n**           Cause *delta* to create a null delta in each of those releases (if any) being  
4791                   skipped when a delta is made in a new release (for example, in making  
4792                   delta 5.1 after delta 2.7, releases 3 and 4 are skipped). These null deltas |  
4793                   shall serve as anchor points so that branch deltas may later be created |  
4794                   from them. The absence of this flag shall cause skipped releases to be |  
4795                   nonexistent in the SCCS file, preventing branch deltas from being created |  
4796                   from them in the future. During the initial creation of an SCCS file, the **n** |  
4797                   flag may be ignored; that is, if the **-r** option is used to set the release |  
4798                   number of the initial SID to a value greater than 1, null deltas need not be |  
4799                   created for the “skipped” releases. |

4800	<b>qtext</b>	Substitute user-definable <i>text</i> for all occurrences of the %Q% keyword in the SCCS file text retrieved by <i>get</i> .
4801		
4802	<b>mmod</b>	Specify the module name of the SCCS file substituted for all occurrences of the %M% keyword in the SCCS file text retrieved by <i>get</i> . If the <b>m</b> flag is not specified, the value assigned shall be the name of the SCCS file with the leading ' . ' removed.
4803		
4804		
4805		
4806	<b>ttype</b>	Specify the <i>type</i> of module in the SCCS file substituted for all occurrences of the %Y% keyword in the SCCS file text retrieved by <i>get</i> .
4807		
4808	<b>vpgm</b>	Cause <i>delta</i> to prompt for modification request (MR) numbers as the reason for creating a delta. The optional value specifies the name of an MR number validation program. (If this flag is set when creating an SCCS file, the application shall ensure that the <b>m</b> option is also used even if its value is null.)
4809		
4810		
4811		
4812		
4813	<b>-d flag</b>	Remove (delete) the specified <i>flag</i> from an SCCS file. Several <b>-d</b> options may be supplied on a single <i>admin</i> command. See the <b>-f</b> option for allowable <i>flag</i> names. (The <b>l</b> list flag gives a <i>list</i> of releases to be unlocked. See the <b>-f</b> option for further description of the <b>l</b> flag and the syntax of a <i>list</i> .)
4814		
4815		
4816		
4817	<b>-a login</b>	Specify a <i>login</i> name, or numerical group ID, to be added to the list of users who may make deltas (changes) to the SCCS file. A group ID shall be equivalent to specifying all <i>login</i> names common to that group ID. Several <b>-a</b> options may be used on a single <i>admin</i> command line. As many <i>logins</i> , or numerical group IDs, as desired may be on the list simultaneously. If the list of users is empty, then anyone may add deltas. If <i>login</i> or group ID is preceded by a '!', the users so specified shall be denied permission to make deltas.
4818		
4819		
4820		
4821		
4822		
4823		
4824	<b>-e login</b>	Specify a <i>login</i> name, or numerical group ID, to be erased from the list of users allowed to make deltas (changes) to the SCCS file. Specifying a group ID is equivalent to specifying all <i>login</i> names common to that group ID. Several <b>-e</b> options may be used on a single <i>admin</i> command line.
4825		
4826		
4827		
4828	<b>-y[comment]</b>	Insert the <i>comment</i> text into the SCCS file as a comment for the initial delta in a manner identical to that of <i>delta</i> . In the POSIX locale, omission of the <b>-y</b> option shall result in a default comment line being inserted in the form:
4829		
4830		
4831		"date and time created %s %s by %s", <date>, <time>, <login>
4832		where <date> is expressed in the format of the <i>date</i> utility's %Y/%m/%d conversion specification, <time> in the format of the <i>date</i> utility's %T conversion specification format, and <login> is the login name of the user creating the file.
4833		
4834		
4835	<b>-m mrlist</b>	Insert the list of modification request (MR) numbers into the SCCS file as the reason for creating the initial delta in a manner identical to <i>delta</i> . The application shall ensure that the <b>v</b> flag is set and the MR numbers are validated if the <b>v</b> flag has a value (the name of an MR number validation program). A diagnostic message shall be written if the <b>v</b> flag is not set or MR validation fails.
4836		
4837		
4838		
4839		
4840	<b>-h</b>	Check the structure of the SCCS file and compare the newly computed checksum (the sum of all the characters in the SCCS file except those in the first line) with the checksum that is stored in the first line of the SCCS file. If the newly computed checksum does not match the checksum in the SCCS file, a diagnostic message shall be written.
4841		
4842		
4843		
4844		

4845            **-z**            Recompute the SCCS file checksum and store it in the first line of the SCCS file (see  
 4846                            the **-h** option above). Note that use of this option on a truly corrupted file may  
 4847                            prevent future detection of the corruption.

#### 4848 **OPERANDS**

4849            The following operands shall be supported:

4850            **file**            A pathname of an existing SCCS file or a directory. If *file* is a directory, the *admin*  
 4851                            utility shall behave as though each file in the directory were specified as a named  
 4852                            file, except that non-SCCS files (last component of the pathname does not begin  
 4853                            with **s**.) and unreadable files shall be silently ignored.

4854            **newfile**        A pathname of an SCCS file to be created.

4855            If exactly one *file* or *newfile* operand appears, and it is **'-'**, the standard input shall be read; each  
 4856                            line of the standard input shall be taken to be the name of an SCCS file to be processed. Non-  
 4857                            SCCS files and unreadable files shall be silently ignored.

#### 4858 **STDIN**

4859            The standard input shall be a text file used only if the **-i** is specified without an option-argument  
 4860                            or if a *file* or *newfile* operand is specified as **'-'**. If the first character of any standard input line is  
 4861                            <SOH> in the POSIX locale, the results are unspecified.

#### 4862 **INPUT FILES**

4863            The existing SCCS files shall be text files of an unspecified format. |

4864            The application shall ensure that the file named by the **-i** option's *name* option-argument shall be |  
 4865                            a text file; if the first character of any line in this file is <SOH> in the POSIX locale, the results are |  
 4866                            unspecified. If this file contains more than 99 999 lines, the number of lines recorded in the |  
 4867                            header for this file shall be 99 999 for this delta. |

#### 4868 **ENVIRONMENT VARIABLES**

4869            The following environment variables shall affect the execution of *admin*:

4870            **LANG**            Provide a default value for the internationalization variables that are unset or null.  
 4871                            (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
 4872                            Internationalization Variables for the precedence of internationalization variables  
 4873                            used to determine the values of locale categories.)

4874            **LC\_ALL**        If set to a non-empty string value, override the values of all the other  
 4875                            internationalization variables.

4876            **LC\_CTYPE**     Determine the locale for the interpretation of sequences of bytes of text data as  
 4877                            characters (for example, single-byte as opposed to multi-byte characters in  
 4878                            arguments and input files).

#### 4879 **LC\_MESSAGES**

4880                            Determine the locale that should be used to affect the format and contents of  
 4881                            diagnostic messages written to standard error and the contents of the default **-y**  
 4882                            comment.

4883            **NLSPATH**     Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

#### 4884 **ASYNCHRONOUS EVENTS**

4885            Default.

4886 **STDOUT**

4887 Not used.

4888 **STDERR**

4889 The standard error shall be used only for diagnostic messages. |

4890 **OUTPUT FILES**4891 Any SCCS files created shall be text files of an unspecified format. During processing of a *file*, a  
4892 locking *z-file*, as described in *get* (on page 2675), may be created and deleted.4893 **EXTENDED DESCRIPTION**

4894 None.

4895 **EXIT STATUS**

4896 The following exit values shall be returned:

4897 0 Successful completion.

4898 &gt;0 An error occurred.

4899 **CONSEQUENCES OF ERRORS**

4900 Default.

4901 **APPLICATION USAGE**4902 It is recommended that directories containing SCCS files be writable by the owner only, and that  
4903 SCCS files themselves be read-only. The mode of the directories should allow only the owner to  
4904 modify SCCS files contained in the directories. The mode of the SCCS files prevents any  
4905 modification at all except by SCCS commands.4906 **EXAMPLES**

4907 None.

4908 **RATIONALE**

4909 None.

4910 **FUTURE DIRECTIONS**

4911 None.

4912 **SEE ALSO**4913 *delta, get, prs, what*4914 **CHANGE HISTORY**

4915 First released in Issue 2.

4916 **Issue 6**

4917 The normative text is reworded to avoid use of the term “must” for application requirements.

4918 The normative text is reworded to emphasize the term “shall” for implementation requirements.

4919 The grammar is updated. |

4920 The Open Group Base Resolution bwg 2001-007 is applied, adding new text to the INPUT FILES |  
4921 section warning that the maximum lines recorded in the file is 99 999. |

4922 **NAME**

4923 alias — define or display aliases

4924 **SYNOPSIS**4925 UP alias [*alias-name*[=*string*] ...]

4926

4927 **DESCRIPTION**

4928 The *alias* utility shall create or redefine alias definitions or write the values of existing alias  
 4929 definitions to standard output. An alias definition provides a string value that shall replace a  
 4930 command name when it is encountered; see Section 2.3.1 (on page 2234).

4931 An alias definition shall affect the current shell execution environment and the execution  
 4932 environments of the subshells of the current shell. When used as specified by this volume of  
 4933 IEEE Std 1003.1-200x, the alias definition shall not affect the parent process of the current shell  
 4934 nor any utility environment invoked by the shell; see Section 2.12 (on page 2263).

4935 **OPTIONS**

4936 None.

4937 **OPERANDS**

4938 The following operands shall be supported:

4939 *alias-name* Write the alias definition to standard output.4940 *alias-name=string*4941 Assign the value of *string* to the alias *alias-name*.

4942 If no operands are given, all alias definitions shall be written to standard output.

4943 **STDIN**

4944 Not used.

4945 **INPUT FILES**

4946 None.

4947 **ENVIRONMENT VARIABLES**4948 The following environment variables shall affect the execution of *alias*:

4949 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 4950 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
 4951 Internationalization Variables for the precedence of internationalization variables  
 4952 used to determine the values of locale categories.)

4953 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 4954 internationalization variables.

4955 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 4956 characters (for example, single-byte as opposed to multi-byte characters in  
 4957 arguments).

4958 *LC\_MESSAGES*

4959 Determine the locale that should be used to affect the format and contents of  
 4960 diagnostic messages written to standard error.

4961 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

4962 **ASYNCHRONOUS EVENTS**

4963 Default.

4964 **STDOUT**4965 The format for displaying aliases (when no operands or only *name* operands are specified) shall  
4966 be:4967 "%s=%s\n", *name*, *value*4968 The *value* string shall be written with appropriate quoting so that it is suitable for reinput to the  
4969 shell. See the description of shell quoting in Section 2.2 (on page 2232).4970 **STDERR**

4971 The standard error shall be used only for diagnostic messages.

4972 **OUTPUT FILES**

4973 None.

4974 **EXTENDED DESCRIPTION**

4975 None.

4976 **EXIT STATUS**

4977 The following exit values shall be returned:

4978 0 Successful completion.

4979 >0 One of the *name* operands specified did not have an alias definition, or an error occurred.4980 **CONSEQUENCES OF ERRORS**

4981 Default.

4982 **APPLICATION USAGE**

4983 None.

4984 **EXAMPLES**4985 1. Change *ls* to give a columnated, more annotated output:4986 `alias ls="ls -CF"`

4987 2. Create a simple “redo” command to repeat previous entries in the command history file:

4988 `alias r='fc -s'`4989 3. Use 1K units for *du*:4990 `alias du=du\ -k`4991 4. Set up *nohup* so that it can deal with an argument that is itself an alias name:4992 `alias nohup="nohup "`4993 **RATIONALE**4994 The *alias* description is based on historical KornShell implementations. Known differences exist  
4995 between that and the C shell. The KornShell version was adopted to be consistent with all the  
4996 other KornShell features in this volume of IEEE Std 1003.1-200x, such as command line editing.4997 Since *alias* affects the current shell execution environment, it is generally provided as a shell  
4998 regular built-in.4999 Historical versions of the KornShell have allowed aliases to be exported to scripts that are  
5000 invoked by the same shell. This is triggered by the *alias* `-x` flag; it is allowed by this volume of  
5001 IEEE Std 1003.1-200x only when an explicit extension such as `-x` is used. The standard  
5002 developers considered that aliases were of use primarily to interactive users and that they

5003 should normally not affect shell scripts called by those users; functions are available to such  
5004 scripts.

5005 Historical versions of the KornShell had not written aliases in a quoted manner suitable for  
5006 reentry to the shell, but this volume of IEEE Std 1003.1-200x has made this a requirement for all  
5007 similar output. Therefore, consistency with this volume of IEEE Std 1003.1-200x was chosen over  
5008 this detail of historical practice.

5009 **FUTURE DIRECTIONS**

5010 None.

5011 **SEE ALSO**

5012 Section 2.9.5 (on page 2256)

5013 **CHANGE HISTORY**

5014 First released in Issue 4.

5015 **Issue 6**

5016 This utility is now marked as part of the User Portability Utilities option.

5017 The APPLICATION USAGE section is added.

## 5018 NAME

5019 ar — create and maintain library archives

## 5020 SYNOPSIS

5021 SD ar -d[-v] archive file ...

5022

5023 XSI ar -m[-abiv][posname] archive file ...

5024

5025 XSI ar -p[-v][-s]archive [file ...]

5026 XSI ar -q[-cv] archive file ...

5027

5028 XSI ar -r[-cuv][-abi][posname]archive file ...

5029 XSI ar -t[-v][-s]archive [file ...]

5030 XSI ar -x[-v][-sCT]archive [file ...]

## 5031 DESCRIPTION

5032 The *ar* utility can be used to create and maintain groups of files combined into an archive. Once  
 5033 an archive has been created, new files can be added, and existing files in an archive can be  
 5034 extracted, deleted, or replaced. When an archive consists entirely of valid object files, the  
 5035 implementation shall format the archive so that it is usable as a library for link editing (see *c99*  
 5036 and *fort77*). When some of the archived files are not valid object files, the suitability of the  
 5037 XSI archive for library use is undefined. If an archive consists entirely of printable files, the entire  
 5038 archive shall be printable.

5039 When *ar* creates an archive, it creates administrative information indicating whether a symbol  
 5040 table is present in the archive. When there is at least one object file that *ar* recognizes as such in  
 5041 the archive, an archive symbol table shall be created in the archive and maintained by *ar*; it is  
 5042 used by the link editor to search the archive. Whenever the *ar* utility is used to create or update  
 5043 the contents of such an archive, the symbol table shall be rebuilt. The *-s* option shall force the  
 5044 symbol table to be rebuilt.

5045 All *file* operands can be pathnames. However, files within archives shall be named by a filename,  
 5046 which is the last component of the pathname used when the file was entered into the archive.  
 5047 The comparison of *file* operands to the names of files in archives shall be performed by  
 5048 comparing the last component of the operand to the name of the file in the archive.

5049 It is unspecified whether multiple files in the archive may be identically named. In the case of  
 5050 XSI such files, however, each *file* and *posname* operand shall match only the first file in the archive  
 5051 having a name that is the same as the last component of the operand.

## 5052 OPTIONS

5053 The *ar* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2,  
 5054 Utility Syntax Guidelines.

5055 The following options shall be supported:

5056 XSI **-a** Position new files in the archive after the file named by the *posname* operand.

5057 XSI **-b** Position new files in the archive before the file named by the *posname* operand.

5058 **-c** Suppress the diagnostic message that is written to standard error by default when  
 5059 the archive *archive* is created.

5060 XSI **-C** Prevent extracted files from replacing like-named files in the file system. This  
 5061 option is useful when **-T** is also used, to prevent truncated filenames from

5062		replacing files with the same prefix.	
5063	<b>-d</b>	Delete one or more <i>files</i> from <i>archive</i> .	
5064 XSI	<b>-i</b>	Position new files in the archive before the file in the archive named by the <i>posname</i>	
5065		operand (equivalent to <b>-b</b> ).	
5066 XSI	<b>-m</b>	Move the named files in the archive. The <b>-a</b> , <b>-b</b> , or <b>-i</b> options with the <i>posname</i>	
5067		operand indicate the position; otherwise, move the names files in the archive to the	
5068		end of the archive.	
5069	<b>-p</b>	Write the contents of the <i>files</i> in the archive named by <i>file</i> operands from <i>archive</i> to	
5070		the standard output. If no <i>file</i> operands are specified, the contents of all files in the	
5071		archive shall be written in the order of the archive.	
5072 XSI	<b>-q</b>	Append the named files to the end of the archive. In this case <i>ar</i> does not check	
5073		whether the added files are already in the archive. This is useful to bypass the	
5074		searching otherwise done when creating a large archive piece by piece.	
5075	<b>-r</b>	Replace or add <i>files</i> to <i>archive</i> . If the archive named by <i>archive</i> does not exist, a	
5076		new archive shall be created and a diagnostic message shall be written to standard	
5077		error (unless the <b>-c</b> option is specified). If no <i>files</i> are specified and the <i>archive</i>	
5078		exists, the results are undefined. Files that replace existing files in the archive shall	
5079		not change the order of the archive. Files that do not replace existing files in the	
5080 XSI		archive shall be appended to the archive unless a <b>-a</b> , <b>-b</b> , or <b>-i</b> option specifies	
5081		another position.	
5082 XSI	<b>-s</b>	Force the regeneration of the archive symbol table even if <i>ar</i> is not invoked with an	
5083		option that modifies the archive contents. This option is useful to restore the	
5084		archive symbol table after it has been stripped; see <i>strip</i> .	
5085	<b>-t</b>	Write a table of contents of <i>archive</i> to the standard output. The files specified by the	
5086		<i>file</i> operands shall be included in the written list. If no <i>file</i> operands are specified,	
5087		all files in <i>archive</i> shall be included in the order of the archive.	
5088 XSI	<b>-T</b>	Allow filename truncation of extracted files whose archive names are longer than	
5089		the file system can support. By default, extracting a file with a name that is too	
5090		long shall be an error; a diagnostic message shall be written and the file shall not	
5091		be extracted.	
5092	<b>-u</b>	Update older files in the archive. When used with the <b>-r</b> option, files in the archive	
5093		shall be replaced only if the corresponding <i>file</i> has a modification time that is at	
5094		least as new as the modification time of the file in the archive.	
5095	<b>-v</b>	Give verbose output. When used with the option characters <b>-d</b> , <b>-r</b> , or <b>-x</b> , write a	
5096		detailed file-by-file description of the archive creation and maintenance activity, as	
5097		described in the STDOUT section.	
5098		When used with <b>-p</b> , write the name of the file in the archive to the standard output	
5099		before writing the file in the archive itself to the standard output, as described in	
5100		the STDOUT section.	
5101		When used with <b>-t</b> , include a long listing of information about the files in the	
5102		archive, as described in the STDOUT section.	
5103	<b>-x</b>	Extract the files in the archive named by the <i>file</i> operands from <i>archive</i> . The	
5104		contents of the archive shall not be changed. If no <i>file</i> operands are given, all files	
5105		in the archive shall be extracted. The modification time of each file extracted shall	
5106		be set to the time the file is extracted from the archive.	

5107 **OPERANDS**

5108 The following operands shall be supported:

5109 *archive* A pathname of the archive. |

5110 *file* A pathname. Only the last component shall be used when comparing against the  
5111 names of files in the archive. If two or more *file* operands have the same last  
5112 pathname component (basename), the results are unspecified. The  
5113 implementation's archive format shall not truncate valid filenames of files added  
5114 to or replaced in the archive.

5115 XSI *posname* The name of a file in the archive, used for relative positioning; see options **-m** and |  
5116 **-r**.

5117 **STDIN**

5118 Not used.

5119 **INPUT FILES**5120 The archive named by *archive* shall be a file in the format created by *ar -r*. |5121 **ENVIRONMENT VARIABLES**5122 The following environment variables shall affect the execution of *ar*:

5123 *LANG* Provide a default value for the internationalization variables that are unset or null.  
5124 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
5125 Internationalization Variables for the precedence of internationalization variables  
5126 used to determine the values of locale categories.)

5127 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
5128 internationalization variables.

5129 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
5130 characters (for example, single-byte as opposed to multi-byte characters in  
5131 arguments and input files).

5132 *LC\_MESSAGES*  
5133 Determine the locale that should be used to affect the format and contents of  
5134 diagnostic messages written to standard error.

5135 *LC\_TIME* Determine the format and content for date and time strings written by *ar -tv*.

5136 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

5137 *TMPDIR* Determine the pathname that overrides the default directory for temporary files, if  
5138 any.

5139 *TZ* Determine the timezone used to calculate date and time strings written by *ar -tv*.  
5140 If *TZ* is unset or null, an unspecified default timezone shall be used.

5141 **ASYNCHRONOUS EVENTS**

5142 Default.

5143 **STDOUT**5144 If the **-d** option is used with the **-v** option, the standard output format shall be:

5145 "d - %s\n", &lt;file&gt;

5146 where *file* is the operand specified on the command line.5147 If the **-p** option is used with the **-v** option, *ar* shall precede the contents of each file with:

5148 "\n<%s>\n\n", <file>

5149 where *file* is the operand specified on the command line, if *file* operands were specified, and the  
5150 name of the file in the archive if they were not.

5151 If the **-r** option is used with the **-v** option:

5152 • If *file* is already in the archive, the standard output format shall be:

5153 "r - %s\n", <file>

5154 where <file> is the operand specified on the command line.

5155 • If *file* is not already in the archive, the standard output format shall be:

5156 "a - %s\n", <file>

5157 where <file> is the operand specified on the command line.

5158 If the **-t** option is used, *ar* shall write the names of the files in the archive to the standard output |  
5159 in the format: |

5160 "%s\n", <file>

5161 where *file* is the operand specified on the command line, if *file* operands were specified, or the  
5162 name of the file in the archive if they were not.

5163 If the **-t** option is used with the **-v** option, the standard output format shall be:

5164 "%s %u/%u %u %s %d %d:%d %d %s\n", <member mode>, <user ID>,  
5165 <group ID>, <number of bytes in member>,  
5166 <abbreviated month>, <day-of-month>, <hour>,  
5167 <minute>, <year>, <file>

5168 where:

5169 <file> Shall be the operand specified on the command line, if *file* operands were specified,  
5170 or the name of the file in the archive if they were not.

5171 <member mode>

5172 Shall be formatted the same as the <file mode> string defined in the STDOUT  
5173 section of *ls*, except that the first character, the <entry type>, is not used; the string |  
5174 represents the file mode of the file in the archive at the time it was added to or |  
5175 replaced in the archive. |

5176 The following represent the last-modification time of a file when it was most recently added to  
5177 or replaced in the archive:

5178 <abbreviated month>

5179 Equivalent to the format of the %b conversion specification format in *date*.

5180 <day-of-month>

5181 Equivalent to the format of the %e conversion specification format in *date*.

5182 <hour> Equivalent to the format of the %H conversion specification format in *date*.

5183 <minute> Equivalent to the format of the %M conversion specification format in *date*.

5184 <year> Equivalent to the format of the %Y conversion specification format in *date*.

5185 When *LC\_TIME* does not specify the POSIX locale, a different format and order of presentation  
5186 of these fields relative to each other may be used in a format appropriate in the specified locale.

5187 If the `-x` option is used with the `-v` option, the standard output format shall be:

5188 "x - %s\n", <file>

5189 where *file* is the operand specified on the command line, if *file* operands were specified, or the  
5190 name of the file in the archive if they were not.

#### 5191 **STDERR**

5192 The standard error shall be used only for diagnostic messages. The diagnostic message about  
5193 creating a new archive when `-c` is not specified shall not modify the exit status. |

#### 5194 **OUTPUT FILES**

5195 Archives are files with unspecified formats.

#### 5196 **EXTENDED DESCRIPTION**

5197 None.

#### 5198 **EXIT STATUS**

5199 The following exit values shall be returned:

5200 0 Successful completion.

5201 >0 An error occurred.

#### 5202 **CONSEQUENCES OF ERRORS**

5203 Default.

#### 5204 **APPLICATION USAGE**

5205 None.

#### 5206 **EXAMPLES**

5207 None.

#### 5208 **RATIONALE**

5209 The archive format is not described. It is recognized that there are several known *ar* formats,  
5210 which are not compatible. The *ar* utility is included, however, to allow creation of archives that  
5211 are intended for use only on one machine. The archive is specified as a file, and it can be moved  
5212 as a file. This does allow an archive to be moved from one machine to another machine that uses  
5213 the same implementation of *ar*. |

5214 Utilities such as *pax* (and its forebears *tar* and *cpio*) also provide portable “archives”. This is a not  
5215 a duplication; the *ar* utility is included to provide an interface primarily for *make* and the  
5216 compilers, based on a historical model.

5217 In historical implementations, the `-q` option (available on XSI-conforming systems) is known to  
5218 execute quickly because *ar* does not check on whether the added members are already in the  
5219 archive. This is useful to bypass the searching otherwise done when creating a large archive  
5220 piece-by-piece. These remarks may but need not remain true for a brand new implementation of  
5221 this utility; hence, these remarks have been moved into the RATIONALE.

5222 BSD implementations historically required applications to provide the `-s` option whenever the  
5223 archive was supposed to contain a symbol table. As in this volume of IEEE Std 1003.1-200x,  
5224 System V historically creates or updates an archive symbol table whenever an object file is  
5225 removed from, added to, or updated in the archive.

5226 The OPERANDS section requires what might seem to be true without specifying it: the archive  
5227 cannot truncate the filenames below {NAME\_MAX}. Some historical implementations do so,  
5228 however, causing unexpected results for the application. Therefore, this volume of  
5229 IEEE Std 1003.1-200x makes the requirement explicit to avoid misunderstandings.

5230 According to the System V documentation, the options **-dmpqrtx** are not required to begin with  
 5231 a hyphen ('-'). This volume of IEEE Std 1003.1-200x requires that a conforming application use  
 5232 the leading hyphen.

5233 The archive format used by the 4.4 BSD implementation is documented in this RATIONALE as  
 5234 an example:

5235 A file created by *ar* begins with the "magic" string "!<arch>\n". The rest of the archive is |  
 5236 made up of objects, each of which is composed of a header for a file, a possible filename, and |  
 5237 the file contents. The header is portable between machine architectures, and, if the file |  
 5238 contents are printable, the archive is itself printable. |

5239 The header is made up of six ASCII fields, followed by a two-character trailer. The fields are |  
 5240 the object name (16 characters), the file last modification time (12 characters), the user and |  
 5241 group IDs (each 6 characters), the file mode (8 characters), and the file size (10 characters). All |  
 5242 numeric fields are in decimal, except for the file mode, which is in octal. |

5243 The modification time is the file *st\_mtime* field. The user and group IDs are the file *st\_uid* and |  
 5244 *st\_gid* fields. The file mode is the file *st\_mode* field. The file size is the file *st\_size* field. The |  
 5245 two-byte trailer is the string "<newline>". |

5246 Only the name field has any provision for overflow. If any filename is more than 16 |  
 5247 characters in length or contains an embedded space, the string "#1/" followed by the ASCII |  
 5248 length of the name is written in the name field. The file size (stored in the archive header) is |  
 5249 incremented by the length of the name. The name is then written immediately following the |  
 5250 archive header. |

5251 Any unused characters in any of these fields are written as <space>s. If any fields are their |  
 5252 particular maximum number of characters in length, there is no separation between the |  
 5253 fields. |

5254 Objects in the archive are always an even number of bytes long; files that are an odd number |  
 5255 of bytes long are padded with a <newline>, although the size in the header does not reflect |  
 5256 this. |

5257 The *ar* utility description requires that (when all its members are valid object files) *ar* produce an  
 5258 object code library, which the linkage editor can use to extract object modules. If the linkage  
 5259 editor needs a symbol table to permit random access to the archive, *ar* must provide it; however,  
 5260 *ar* does not require a symbol table.

5261 The BSD **-o** option was omitted. It is a rare conforming application that uses *ar* to extract object |  
 5262 code from a library with concern for its modification time, since this can only be of importance |  
 5263 to *make*. Hence, since this functionality is not deemed important for applications portability, the |  
 5264 modification time of the extracted files is set to the current time. |

5265 There is at least one known implementation (for a small computer) that can accommodate only |  
 5266 object files for that system, disallowing mixed object and other files. The ability to handle any |  
 5267 type of file is not only historical practice for most implementations, but is also a reasonable |  
 5268 expectation. |

5269 Consideration was given to changing the output format of *ar -tv* to the same format as the  
 5270 output of *ls -l*. This would have made parsing the output of *ar* the same as that of *ls*. This was  
 5271 rejected in part because the current *ar* format is commonly used and changes would break  
 5272 historical usage. Second, *ar* gives the user ID and group ID in numeric format separated by a  
 5273 slash. Changing this to be the user name and group name would not be correct if the archive  
 5274 were moved to a machine that contained a different user database. Since *ar* cannot know |  
 5275 whether the archive was generated on the same machine, it cannot tell what to report. |

- 5276 The text on the `-ur` option combination is historical practice—since one filename can easily  
5277 represent two different files (for example, `/a/foo` and `/b/foo`), it is reasonable to replace the file in  
5278 the archive even when the modification time in the archive is identical to that in the file system. |
- 5279 **FUTURE DIRECTIONS**
- 5280 None.
- 5281 **SEE ALSO**
- 5282 *c99*, *pax*, *strip* the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 13, Headers,  
5283 `<unistd.h>` description of {POSIX\_NO\_TRUNC}
- 5284 **CHANGE HISTORY**
- 5285 First released in Issue 2.
- 5286 **Issue 5**
- 5287 FUTURE DIRECTIONS section added.
- 5288 **Issue 6**
- 5289 This utility is now marked as part of the Software Development Utilities option.
- 5290 The STDOUT description is changed for the `-v` option to align with the IEEE P1003.2b draft  
5291 standard.
- 5292 The normative text is reworded to avoid use of the term “must” for application requirements.
- 5293 The *TZ* entry is added to the ENVIRONMENT VARIABLES section. |
- 5294 IEEE PASC Interpretation 1003.2 #198 is applied, changing the description to consistently use |  
5295 “file” to refer to a file in the file system hierarchy, “archive” to refer to the archive being |  
5296 operated upon by the *ar* utility, and “file in the archive” to refer to a copy of a file that is |  
5297 contained in the archive. |

5298 **NAME**

5299           asa — interpret carriage-control characters

5300 **SYNOPSIS**5301 FR       asa [ *file* ... ]

5302

5303 **DESCRIPTION**5304           The *asa* utility shall write its input files to standard output, mapping carriage-control characters  
5305           from the text files to line-printer control sequences in an implementation-defined manner.5306           The first character of every line shall be removed from the input, and the following actions are  
5307           performed:

5308           If the character removed is:

5309           &lt;space&gt; The rest of the line is output without change.

5310           0        A &lt;newline&gt; is output, then the rest of the input line.

5311           1        One or more implementation-defined characters that causes an advance to the next  
5312           page shall be output, followed by the rest of the input line.5313           +        The <newline> of the previous line shall be replaced with one or more  
5314           implementation-defined characters that causes printing to return to column position 1,  
5315           followed by the rest of the input line. If the '+' is the first character in the input, it shall  
5316           be equivalent to <space>. |5317           The action of the *asa* utility is unspecified upon encountering any character other than those  
5318           listed above as the first character in a line.5319 **OPTIONS**

5320           None.

5321 **OPERANDS**5322           *file*        A pathname of a text file used for input. If no *file* operands are specified, the  
5323           standard input shall be used.5324 **STDIN**5325           The standard input shall be used only if no *file* operands are specified; see the INPUT FILES |  
5326           section.5327 **INPUT FILES**

5328           The input files shall be text files.

5329 **ENVIRONMENT VARIABLES**5330           The following environment variables shall affect the execution of *asa*:5331           *LANG*        Provide a default value for the internationalization variables that are unset or null.  
5332           (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
5333           Internationalization Variables for the precedence of internationalization variables  
5334           used to determine the values of locale categories.)5335           *LC\_ALL*       If set to a non-empty string value, override the values of all the other  
5336           internationalization variables.5337           *LC\_CTYPE*   Determine the locale for the interpretation of sequences of bytes of text data as  
5338           characters (for example, single-byte as opposed to multi-byte characters in  
5339           arguments and input files).

5340 *LC\_MESSAGES*

5341 Determine the locale that should be used to affect the format and contents of  
5342 diagnostic messages written to standard error.

5343 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

5344 **ASYNCHRONOUS EVENTS**

5345 Default.

5346 **STDOUT**

5347 The standard output shall be the text from the input file modified as described in the  
5348 DESCRIPTION section.

5349 **STDERR**

5350 None.

5351 **OUTPUT FILES**

5352 None.

5353 **EXTENDED DESCRIPTION**

5354 None.

5355 **EXIT STATUS**

5356 The following exit values shall be returned:

5357 0 All input files were output successfully.

5358 >0 An error occurred.

5359 **CONSEQUENCES OF ERRORS**

5360 Default.

5361 **APPLICATION USAGE**

5362 None.

5363 **EXAMPLES**

5364 1. The following command:

5365 *asa file*

5366 permits the viewing of *file* (created by a program using FORTRAN-style carriage control  
5367 characters) on a terminal.

5368 2. The following command:

5369 *a.out | asa | lp*

5370 formats the FORTRAN output of **a.out** and directs it to the printer.

5371 **RATIONALE**

5372 The *asa* utility is needed to map “standard” FORTRAN 77 output into a form acceptable to  
5373 contemporary printers. Usually, *asa* is used to pipe data to the *lp* utility; see *lp*.

5374 This utility is generally used only by FORTRAN programs. The standard developers decided to  
5375 retain *asa* to avoid breaking the historical large base of FORTRAN applications that put  
5376 carriage-control characters in their output files. There is no requirement that a system have a  
5377 FORTRAN compiler in order to run applications that need *asa*.

5378 Historical implementations have used an ASCII <form-feed> in response to a 1 and an ASCII  
5379 <carriage-return> in response to a '+'. It is suggested that implementations treat characters  
5380 other than 0, 1, and '+' as <space> in the absence of any compelling reason to do otherwise.  
5381 However, the action is listed here as “unspecified”, permitting an implementation to provide

5382 extensions to access fast multiple-line slewing and channel seeking in a non-portable manner.

5383 **FUTURE DIRECTIONS**

5384 None.

5385 **SEE ALSO**

5386 *fort77, lp*

5387 **CHANGE HISTORY**

5388 First released in Issue 4.

5389 **Issue 6**

5390 This utility is now marked as part of the FORTRAN Runtime Utilities option.

5391 The normative text is reworded to avoid use of the term “must” for application requirements.

## 5392 NAME

5393 at — execute commands at a later time

## 5394 SYNOPSIS

5395 UP at [-m][-f *file*][-q *queuename*] -t *time\_arg*5396 at [-m][-f *file*][-q *queuename*] *timespec* ...5397 at -r *at\_job\_id* ...5398 at -l -q *queuename*5399 at -l [*at\_job\_id* ...]

5400

## 5401 DESCRIPTION

5402 The *at* utility shall read commands from standard input and group them together as an *at-job*, to  
5403 be executed at a later time.5404 The *at-job* shall be executed in a separate invocation of the shell, running in a separate process  
5405 group with no controlling terminal, except that the environment variables, current working  
5406 directory, file creation mask, and other implementation-defined execution-time attributes in  
5407 effect when the *at* utility is executed shall be retained and used when the *at-job* is executed.5408 When the *at-job* is submitted, the *at\_job\_id* and scheduled time shall be written to standard error.  
5409 The *at\_job\_id* is an identifier that shall be a string consisting solely of alphanumeric characters  
5410 and the period character. The *at\_job\_id* shall be assigned by the system when the job is scheduled  
5411 such that it uniquely identifies a particular job.5412 User notification and the processing of the job's standard output and standard error are  
5413 described under the *-m* option.5414 XSI Users shall be permitted to use *at* if their name appears in the file */usr/lib/cron/at.allow*. If that  
5415 file does not exist, the file */usr/lib/cron/at.deny* shall be checked to determine whether the user  
5416 shall be denied access to *at*. If neither file exists, only a process with the appropriate privileges  
5417 shall be allowed to submit a job. If only *at.deny* exists and is empty, global usage shall be  
5418 permitted. The *at.allow* and *at.deny* files shall consist of one user name per line. |

## 5419 OPTIONS

5420 The *at* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2,  
5421 Utility Syntax Guidelines.

5422 The following options shall be supported:

5423 *-f file* Specify the pathname of a file to be used as the source of the *at-job*, instead of  
5424 standard input.5425 *-l* (The letter ell.) Report all jobs scheduled for the invoking user if no *at\_job\_id*  
5426 operands are specified. If *at\_job\_ids* are specified, report only information for these  
5427 jobs. The output shall be written to standard output.5428 *-m* Send mail to the invoking user after the *at-job* has run, announcing its completion.  
5429 Standard output and standard error produced by the *at-job* shall be mailed to the  
5430 user as well, unless redirected elsewhere. Mail shall be sent even if the job  
5431 produces no output.5432 If *-m* is not used, the job's standard output and standard error shall be provided to  
5433 the user by means of mail, unless they are redirected elsewhere; if there is no such  
5434 output to provide, the implementation need not notify the user of the job's  
5435 completion.

- 5436           **-q** *queuename*  
 5437                   Specify in which queue to schedule a job for submission. When used with the **-l**  
 5438                   option, limit the search to that particular queue. By default, at-jobs shall be  
 5439                   scheduled in queue *a*. In contrast, queue *b* shall be reserved for batch jobs; see  
 5440                   *batch*. The meanings of all other *queuenames* are implementation-defined. If **-q** is  
 5441                   specified along with either of the **-t** *time\_arg* or *timespec* arguments, the results are  
 5442                   unspecified.
- 5443           **-r**           Remove the jobs with the specified *at\_job\_id* operands that were previously  
 5444                   scheduled by the *at* utility.
- 5445           **-t** *time\_arg*   Submit the job to be run at the time specified by the *time* option-argument, which  
 5446                   the application shall ensure has the format as specified by the *touch -t time* utility.

5447 **OPERANDS**

5448           The following operands shall be supported:

5449           *at\_job\_id*       The name reported by a previous invocation of the *at* utility at the time the job was  
 5450                   scheduled.

5451           *timespec*       Submit the job to be run at the date and time specified. All of the *timespec* operands  
 5452                   are interpreted as if they were separated by <space>s and concatenated, and shall  
 5453                   be parsed as described in the grammar at the end of this section. The date and time  
 5454                   shall be interpreted as being in the timezone of the user (as determined by the *TZ*  
 5455                   variable), unless a timezone name appears as part of *time*, below.

5456                   In the POSIX locale, the following describes the three parts of the time  
 5457                   specification string. All of the values from the *LC\_TIME* categories in the POSIX  
 5458                   locale shall be recognized in a case-insensitive manner.

5459           *time*           The time can be specified as one, two, or four digits. One-digit and  
 5460                   two-digit numbers shall be taken to be hours; four-digit numbers to  
 5461                   be hours and minutes. The time can alternatively be specified as two  
 5462                   numbers separated by a colon, meaning *hour:minute*. An AM/PM  
 5463                   indication (one of the values from the **am\_pm** keywords in the  
 5464                   *LC\_TIME* locale category) can follow the time; otherwise, a 24-hour  
 5465                   clock time shall be understood. A timezone name can also follow to  
 5466                   further qualify the time. The acceptable timezone names are  
 5467                   implementation-defined, except that they shall be case-insensitive  
 5468                   and the string **utc** is supported to indicate the time is in Coordinated  
 5469                   Universal Time. In the POSIX locale, the *time* field can also be one of  
 5470                   the following tokens:

5471                   **midnight**   Indicates the time 12:00 am (00:00).

5472                   **noon**         Indicates the time 12:00 pm.

5473                   **now**         Indicates the current day and time. Invoking *at <now>*  
 5474                   shall submit an at-job for potentially immediate  
 5475                   execution (that is, subject only to unspecified  
 5476                   scheduling delays).

5477           *date*           An optional *date* can be specified as either a month name (one of the  
 5478                   values from the **mon** or **abmon** keywords in the *LC\_TIME* locale  
 5479                   category) followed by a day number (and possibly year number  
 5480                   preceded by a comma), or a day of the week (one of the values from  
 5481                   the **day** or **abday** keywords in the *LC\_TIME* locale category). In the  
 5482                   POSIX locale, two special days shall be recognized:

5483                   **today**       Indicates the current day.

5484                   **tomorrow**   Indicates the day following the current day.

5485                   If no *date* is given, **today** shall be assumed if the given time is greater

5486                   than the current time, and **tomorrow** shall be assumed if it is less. If

5487                   the given month is less than the current month (and no year is given),

5488                   next year shall be assumed.

5489                   *increment*   The optional *increment* shall be a number preceded by a plus sign

5490                   ('+') and suffixed by one of the following: **minutes, hours, days,**

5491                   **weeks, months, or years.** (The singular forms shall be also

5492                   accepted.) The keyword **next** shall be equivalent to an increment

5493                   number of +1. For example, the following are equivalent commands:

5494                   at 2pm + 1 week

5495                   at 2pm next week

5496                   The following grammar describes the precise format of *timespec* in the POSIX locale. The general

5497                   conventions for this style of grammar are described in Section 1.10 (on page 2220). This formal

5498                   syntax shall take precedence over the preceding text syntax description. The longest possible

5499                   token or delimiter shall be recognized at a given point. When used in a *timespec*, white space

5500                   shall also delimit tokens.

```
5501 %token hr24clock_hr_min
5502 %token hr24clock_hour
5503 /*
5504     A hr24clock_hr_min is a one, two, or four-digit number. A one-digit
5505     or two-digit number constitutes a hr24clock_hour. A hr24clock_hour
5506     may be any of the single digits [0,9], or may be double digits, ranging
5507     from [00,23]. If a hr24clock_hr_min is a four digit number, the
5508     first two digits shall be a valid hr24clock_hour, while the last two
5509     represent the number of minutes, from [00,59].
5510 */
5511 %token wallclock_hr_min
5512 %token wallclock_hour
5513 /*
5514     A wallclock_hr_min is a one, two-digit, or four-digit number.
5515     A one-digit or two-digit number constitutes a wallclock_hour.
5516     A wallclock_hour may be any of the single digits [1,9], or may
5517     be double digits, ranging from [01,12]. If a wallclock_hr_min
5518     is a four-digit number, the first two digits shall be a valid
5519     wallclock_hour, while the last two represent the number of
5520     minutes, from [00,59].
5521 */
5522 %token minute
5523 /*
5524     A minute is a one or two-digit number whose values can be [0,9]
5525     or [00,59].
5526 */
5527 %token day_number
5528 /*
5529     A day_number is a number in the range appropriate for the particular
5530     month and year specified by month_name and year_number, respectively.
```

```

5531         If no year_number is given, the current year is assumed if the given
5532         date and time are later this year. If no year_number is given and
5533         the date and time have already occurred this year and the month is
5534         not the current month, next year is the assumed year.
5535     */

5536     %token year_number
5537     /*
5538         A year_number is a four-digit number representing the year A.D., in
5539         which the at_job is to be run.
5540     */

5541     %token inc_number
5542     /*
5543         The inc_number is the number of times the succeeding increment
5544         period is to be added to the specified date and time.
5545     */

5546     %token timezone_name
5547     /*
5548         The name of an optional timezone suffix to the time field, in an
5549         implementation-defined format.
5550     */

5551     %token month_name
5552     /*
5553         One of the values from the mon or abmon keywords in the LC_TIME
5554         locale category.
5555     */

5556     %token day_of_week
5557     /*
5558         One of the values from the day or abday keywords in the LC_TIME
5559         locale category.
5560     */

5561     %token am_pm
5562     /*
5563         One of the values from the am_pm keyword in the LC_TIME locale
5564         category.
5565     */

5566     %start timespec
5567     %%
5568     timespec      : time
5569                   | time date
5570                   | time increment
5571                   | time date increment
5572                   | nowspec
5573                   ;

5574     nowspec      : "now"
5575                   | "now" increment
5576                   ;

5577     time         : hr24clock_hr_min
5578                   | hr24clock_hr_min timezone_name

```

```

5579         | hr24clock_hour ":" minute
5580         | hr24clock_hour ":" minute timezone_name
5581         | wallclock_hr_min am_pm
5582         | wallclock_hr_min am_pm timezone_name
5583         | wallclock_hour ":" minute am_pm
5584         | wallclock_hour ":" minute am_pm timezone_name
5585         | "noon"
5586         | "midnight"
5587         ;

5588     date      : month_name day_number
5589         | month_name day_number "," year_number
5590         | day_of_week
5591         | "today"
5592         | "tomorrow"
5593         ;

5594     increment : "+" inc_number inc_period
5595         | "next" inc_period
5596         ;

5597     inc_period : "minute" | "minutes"
5598         | "hour" | "hours"
5599         | "day" | "days"
5600         | "week" | "weeks"
5601         | "month" | "months"
5602         | "year" | "years"
5603         ;

```

#### 5604 STDIN

5605 The standard input shall be a text file consisting of commands acceptable to the shell command  
5606 language described in Chapter 2 (on page 2231). The standard input shall only be used if no *-f*  
5607 *file* option is specified.

#### 5608 INPUT FILES

5609 See the STDIN section.

5610 XSI The text files `/usr/lib/cron/at.allow` and `/usr/lib/cron/at.deny` shall contain zero or more user  
5611 names, one per line, of users who are, respectively, authorized or denied access to the *at* and  
5612 *batch* utilities.

#### 5613 ENVIRONMENT VARIABLES

5614 The following environment variables shall affect the execution of *at*:

5615 *LANG* Provide a default value for the internationalization variables that are unset or null.  
5616 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
5617 Internationalization Variables for the precedence of internationalization variables  
5618 used to determine the values of locale categories.)

5619 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
5620 internationalization variables.

5621 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
5622 characters (for example, single-byte as opposed to multi-byte characters in  
5623 arguments and input files).

5624 *LC\_MESSAGES*

5625 Determine the locale that should be used to affect the format and contents of

- 5626 diagnostic messages written to standard error and informative messages written to  
5627 standard output.
- 5628 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 5629 **LC\_TIME** Determine the format and contents for date and time strings written and accepted  
5630 by *at*.
- 5631 **SHELL** Determine a name of a command interpreter to be used to invoke the *at*-job. If the  
5632 variable is unset or null, *sh* shall be used. If it is set to a value other than a name for  
5633 *sh*, the implementation shall do one of the following: use that shell; use *sh*; use the  
5634 login shell from the user database; or any of the preceding accompanied by a  
5635 warning diagnostic about which was chosen.
- 5636 **TZ** Determine the timezone. The job shall be submitted for execution at the time  
5637 specified by *timespec* or *-t time* relative to the timezone specified by the *TZ*  
5638 variable. If *timespec* specifies a timezone, it shall override *TZ*. If *timespec* does not  
5639 specify a timezone and *TZ* is unset or null, an unspecified default timezone shall  
5640 be used.
- 5641 **ASYNCHRONOUS EVENTS**
- 5642 Default.
- 5643 **STDOUT**
- 5644 When standard input is a terminal, prompts of unspecified format for each line of the user input  
5645 described in the *STDIN* section may be written to standard output.
- 5646 In the POSIX locale, the following shall be written to the standard output for each job when jobs  
5647 are listed in response to the *-l* option:
- 5648 "%s\t%s\n", *at\_job\_id*, <*date*>
- 5649 where *date* shall be equivalent in format to the output of:
- 5650 date +"%a %b %e %T %Y"
- 5651 The date and time written shall be adjusted so that they appear in the timezone of the user (as  
5652 determined by the *TZ* variable).
- 5653 **STDERR**
- 5654 In the POSIX locale, the following shall be written to standard error when a job has been  
5655 successfully submitted:
- 5656 "job %s at %s\n", *at\_job\_id*, <*date*>
- 5657 where *date* has the same format as is described in the *STDOUT* section. Neither this, nor warning  
5658 messages concerning the selection of the command interpreter, shall be considered a diagnostic  
5659 that changes the exit status.
- 5660 Diagnostic messages, if any, shall be written to standard error.
- 5661 **OUTPUT FILES**
- 5662 None.
- 5663 **EXTENDED DESCRIPTION**
- 5664 None.
- 5665 **EXIT STATUS**
- 5666 The following exit values shall be returned:
- 5667 0 The *at* utility successfully submitted, removed, or listed a job or jobs.

5668 >0 An error occurred.

#### 5669 CONSEQUENCES OF ERRORS

5670 The job shall not be scheduled, removed, or listed.

#### 5671 APPLICATION USAGE

5672 The format of the *at* command line shown here is guaranteed only for the POSIX locale. Other  
5673 cultures may be supported with substantially different interfaces, although implementations are  
5674 encouraged to provide comparable levels of functionality.

5675 Since the commands run in a separate shell invocation, running in a separate process group with  
5676 no controlling terminal, open file descriptors, traps, and priority inherited from the invoking  
5677 environment are lost.

5678 Some implementations do not allow substitution of different shells using *SHELL*. System V  
5679 systems, for example, have used the login shell value for the user in */etc/passwd*. To select  
5680 reliably another command interpreter, the user must include it as part of the script, such as:

```
5681 $ at 1800
5682 myshell myscript
5683 job ... at ...
5684 $
```

#### 5685 EXAMPLES

5686 1. This sequence can be used at a terminal:

```
5687 at -m 0730 tomorrow
5688 sort < file >outfile
5689 EOT
```

5690 2. This sequence, which demonstrates redirecting standard error to a pipe, is useful in a  
5691 command procedure (the sequence of output redirection specifications is significant):

```
5692 at now + 1 hour <<!
5693 diff file1 file2 2>&1 >outfile | mailx mygroup
5694 !
```

5695 3. To have a job reschedule itself, *at* can be invoked from within the *at*-job. For example, this  
5696 daily processing script named **my.daily** runs every day (although *crontab* is a more  
5697 appropriate vehicle for such work):

```
5698 # my.daily runs every day
5699 daily processing
5700 at now tomorrow < my.daily
```

5701 4. The spacing of the three portions of the POSIX locale *timespec* is quite flexible as long as  
5702 there are no ambiguities. Examples of various times and operand presentation include:

```
5703 at 0815am Jan 24
5704 at 8 :15amjan24
5705 at now "+ 1day"
5706 at 5 pm FRIday
5707 at '17
5708 utc+
5709 30minutes'
```

## 5710 RATIONALE

5711 The *at* utility reads from standard input the commands to be executed at a later time. It may be  
 5712 useful to redirect standard output and standard error within the specified commands.

5713 The **-t** *time* option was added as a new capability to support an internationalized way of  
 5714 specifying a time for execution of the submitted job.

5715 Early proposals added a “jobname” concept as a way of giving submitted jobs names that are |  
 5716 meaningful to the user submitting them. The historical, system-specified *at\_job\_id* gives no |  
 5717 indication of what the job is. Upon further reflection, it was decided that the benefit of this was |  
 5718 not worth the change in historical interface. The *at* functionality is useful in simple |  
 5719 environments, but in large or complex situations, the functionality provided by the Batch |  
 5720 Services option is more suitable. |

5721 The **-q** option historically has been an undocumented option, used mainly by the *batch* utility.

5722 The System V **-m** option was added to provide a method for informing users that an at-job had  
 5723 completed. Otherwise, users are only informed when output to standard error or standard  
 5724 output are not redirected.

5725 The behavior of *at* *<now>* was changed in an early proposal from being unspecified to  
 5726 submitting a job for potentially immediate execution. Historical BSD *at* implementations  
 5727 support this. Historical System V implementations give an error in that case, but a change to the  
 5728 System V versions should have no backwards compatibility ramifications.

5729 On BSD-based systems, a **-u** *user* option has allowed those with appropriate privileges to access  
 5730 the work of other users. Since this is primarily a system administration feature and is not  
 5731 universally implemented, it has been omitted. Similarly, a specification for the output format for  
 5732 user with appropriate privileges viewing the queues of other users has been omitted.

5733 The **-f** *file* option from System V is used instead of the BSD method of using the last operand as  
 5734 the pathname. The BSD method is ambiguous—does:

5735 `at 1200 friday`

5736 mean the same thing if there is a file named **friday** in the current directory?

5737 The *at\_job\_id* is composed of a limited character set in historical practice, and it is mandated here  
 5738 to invalidate systems that might try using characters that require shell quoting or that could not  
 5739 be easily parsed by shell scripts.

5740 The *at* utility varies between System V and BSD systems in the way timezones are used. On  
 5741 System V systems, the *TZ* variable affects the at-job submission times and the times displayed  
 5742 for the user. On BSD systems, *TZ* is not taken into account. The BSD behavior is easily achieved  
 5743 with the current specification. If the user wishes to have the timezone default to that of the  
 5744 system, they merely need to issue the *at* command immediately following an unsetting or null  
 5745 assignment to *TZ*. For example:

5746 `TZ= at noon ...`

5747 gives the desired BSD result.

5748 While the *yacc*-like grammar specified in the OPERANDS section is lexically unambiguous with  
 5749 respect to the digit strings, a lexical analyzer would probably be written to look for and return  
 5750 digit strings in those cases. The parser could then check whether the digit string returned is a  
 5751 valid *day\_number*, *year\_number*, and so on, based on the context.

5752 **FUTURE DIRECTIONS**

5753 None.

5754 **SEE ALSO**5755 *batch, crontab*5756 **CHANGE HISTORY**

5757 First released in Issue 2.

5758 **Issue 6**

5759 This utility is now marked as part of the User Portability Utilities option.

5760 The following new requirements on POSIX implementations derive from alignment with the  
5761 Single UNIX Specification:

- 5762
- If **-m** is not used, the job's standard output and standard error are provided to the user by  
5763 mail.

5764 The effects of using the **-q** and **-t** options as defined in the IEEE P1003.2b draft standard are  
5765 specified.

5766 The normative text is reworded to avoid use of the term “must” for application requirements.

5767 **NAME**

5768 awk — pattern scanning and processing language

5769 **SYNOPSIS**5770 awk [-F *ERE*][-v *assignment*] ... *program* [*argument* ...]5771 awk [-F *ERE*] -f *progfile* ... [-v *assignment*] ... [*argument* ...]5772 **DESCRIPTION**

5773 The *awk* utility shall execute programs written in the *awk* programming language, which is  
 5774 specialized for textual data manipulation. An *awk* program is a sequence of patterns and  
 5775 corresponding actions. When input is read that matches a pattern, the action associated with  
 5776 that pattern is carried out.

5777 Input shall be interpreted as a sequence of records. By default, a record is a line, less its  
 5778 terminating <newline>, but this can be changed by using the **RS** built-in variable. Each record of  
 5779 input shall be matched in turn against each pattern in the program. For each pattern matched,  
 5780 the associated action shall be executed.

5781 The *awk* utility shall interpret each input record as a sequence of fields where, by default, a field  
 5782 is a string of non-<blank>s. This default white-space field delimiter can be changed by using the  
 5783 **FS** built-in variable or the **-F *ERE***. The *awk* utility shall denote the first field in a record \$1, the  
 5784 second \$2, and so on. The symbol \$0 shall refer to the entire record; setting any other field causes  
 5785 the re-evaluation of \$0. Assigning to \$0 shall reset the values of all other fields and the **NF** built-  
 5786 in variable.

5787 **OPTIONS**

5788 The *awk* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section  
 5789 12.2, Utility Syntax Guidelines.

5790 The following options shall be supported:

5791 **-F *ERE*** Define the input field separator to be the extended regular expression *ERE*, before  
 5792 any input is read; see **Regular Expressions** (on page 2363).

5793 **-f *progfile*** Specify the pathname of the file *progfile* containing an *awk* program. If multiple  
 5794 instances of this option are specified, the concatenation of the files specified as  
 5795 *progfile* in the order specified shall be the *awk* program. The *awk* program can  
 5796 alternatively be specified in the command line as a single argument.

5797 **-v *assignment***  
 5798 The application shall ensure that the *assignment* argument is in the same form as an  
 5799 *assignment* operand. The specified variable assignment shall occur prior to  
 5800 executing the *awk* program, including the actions associated with **BEGIN** patterns  
 5801 (if any). Multiple occurrences of this option can be specified.

5802 **OPERANDS**

5803 The following operands shall be supported:

5804 *program* If no **-f** option is specified, the first operand to *awk* shall be the text of the *awk*  
 5805 program. The application shall supply the *program* operand as a single argument to  
 5806 *awk*. If the text does not end in a <newline>, *awk* shall interpret the text as if it did.

5807 *argument* Either of the following two types of *argument* can be intermixed:

5808 *file* A pathname of a file that contains the input to be read, which is  
 5809 matched against the set of patterns in the program. If no *file* operands  
 5810 are specified, or if a *file* operand is '-', the standard input shall be  
 5811 used.

5812           *assignment*   An operand that begins with an underscore or alphabetic character  
5813 from the portable character set (see the table in the Base Definitions  
5814 volume of IEEE Std 1003.1-200x, Section 6.1, Portable Character Set),  
5815 followed by a sequence of underscores, digits, and alphabetic characters  
5816 from the portable character set, followed by the '=' character, shall  
5817 specify a variable assignment rather than a pathname. The  
5818 characters before the '=' represent the name of an *awk* variable; if  
5819 that name is an *awk* reserved word (see **Grammar** (on page 2372)) the  
5820 behavior is undefined. The characters following the equal sign shall  
5821 be interpreted as if they appeared in the *awk* program preceded and  
5822 followed by a double-quote ('"') character, as a **STRING** token (see  
5823 **Grammar** (on page 2372)), except that if the last character is an  
5824 unescaped backslash, it shall be interpreted as a literal backslash  
5825 rather than as the first character of the sequence "\ ". The variable  
5826 shall be assigned the value of that **STRING** token and, if  
5827 appropriate, shall be considered a *numeric string* (see **Expressions in**  
5828 **awk** (on page 2358)), the variable shall also be assigned its numeric  
5829 value. Each such variable assignment shall occur just prior to the  
5830 processing of the following *file*, if any. Thus, an assignment before  
5831 the first *file* argument shall be executed after the **BEGIN** actions (if  
5832 any), while an assignment after the last *file* argument shall occur  
5833 before the **END** actions (if any). If there are no *file* arguments,  
5834 assignments shall be executed before processing the standard input.

#### 5835 **STDIN**

5836           The standard input shall be used only if no *file* operands are specified, or if a *file* operand is '-';  
5837 see the INPUT FILES section. If the *awk* program contains no actions and no patterns, but is  
5838 otherwise a valid *awk* program, standard input and any *file* operands shall not be read and *awk*  
5839 shall exit with a return status of zero.

#### 5840 **INPUT FILES**

5841           Input files to the *awk* program from any of the following sources shall be text files:

- 5842           • Any *file* operands or their equivalents, achieved by modifying the *awk* variables **ARGV** and
- 5843           **ARGC**
- 5844           • Standard input in the absence of any *file* operands
- 5845           • Arguments to the **getline** function

5846           Whether the variable **RS** is set to a value other than a <newline> or not, for these files,  
5847 implementations shall support records terminated with the specified separator up to  
5848 {**LINE\_MAX**} bytes and may support longer records.

5849           If **-f progfile** is specified, the application shall ensure that the files named by each of the *progfile*  
5850 option-arguments are text files containing an *awk* program.

#### 5851 **ENVIRONMENT VARIABLES**

5852           The following environment variables shall affect the execution of *awk*:

- 5853           **LANG**           Provide a default value for the internationalization variables that are unset or null.  
5854                               (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
5855                               Internationalization Variables for the precedence of internationalization variables  
5856                               used to determine the values of locale categories.)
- 5857           **LC\_ALL**          If set to a non-empty string value, override the values of all the other  
5858                               internationalization variables.

- 5859            *LC\_COLLATE*  
 5860            Determine the locale for the behavior of ranges, equivalence classes, and multi-  
 5861            character collating elements within regular expressions and in comparisons of  
 5862            string values.
- 5863            *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 5864            characters (for example, single-byte as opposed to multi-byte characters in  
 5865            arguments and input files), the behavior of character classes within regular  
 5866            expressions, the identification of characters as letters, and the mapping of  
 5867            uppercase and lowercase characters for the **toupper** and **tolower** functions.
- 5868            *LC\_MESSAGES*  
 5869            Determine the locale that should be used to affect the format and contents of  
 5870            diagnostic messages written to standard error.
- 5871            *LC\_NUMERIC*  
 5872            Determine the radix character used when interpreting numeric input, performing  
 5873            conversions between numeric and string values, and formatting numeric output.  
 5874            Regardless of locale, the period character (the decimal-point character of the  
 5875            POSIX locale) is the decimal-point character recognized in processing *awk*  
 5876            programs (including assignments in command line arguments).
- 5877 XSI        *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 5878            *PATH* Determine the search path when looking for commands executed by *system(expr)*,  
 5879            or input and output pipes; see the Base Definitions volume of  
 5880            IEEE Std 1003.1-200x, Chapter 8, Environment Variables.
- 5881            In addition, all environment variables shall be visible via the *awk* variable **ENVIRON**.
- 5882 **ASYNCHRONOUS EVENTS**
- 5883            Default.
- 5884 **STDOUT**
- 5885            The nature of the output files depends on the *awk* program.
- 5886 **STDERR**
- 5887            The standard error shall be used only for diagnostic messages. |
- 5888 **OUTPUT FILES**
- 5889            The nature of the output files depends on the *awk* program.
- 5890 **EXTENDED DESCRIPTION**
- 5891            **Overall Program Structure**
- 5892            An *awk* program is composed of pairs of the form:
- 5893            *pattern* { *action* }
- 5894            Either the pattern or the action (including the enclosing brace characters) can be omitted.
- 5895            A missing pattern shall match any record of input, and a missing action shall be equivalent to:
- 5896            { *print* }
- 5897            Execution of the *awk* program shall start by first executing the actions associated with all **BEGIN**  
 5898            patterns in the order they occur in the program. Then each *file* operand (or standard input if no  
 5899            files were specified) shall be processed in turn by reading data from the file until a record  
 5900            separator is seen (<newline> by default). Before the first reference to a field in the record is  
 5901            evaluated, the record shall be split into fields, according to the rules in **Regular Expressions** (on

5902 page 2363), using the value of **FS** that was current at the time the record was read. Each pattern  
 5903 in the program then shall be evaluated in the order of occurrence, and the action associated with  
 5904 each pattern that matches the current record executed. The action for a matching pattern shall be  
 5905 executed before evaluating subsequent patterns. Finally, the actions associated with all **END**  
 5906 patterns shall be executed in the order they occur in the program.

### 5907 Expressions in awk

5908 Expressions describe computations used in *patterns* and *actions*. In the following table, valid  
 5909 expression operations are given in groups from highest precedence first to lowest precedence  
 5910 last, with equal-precedence operators grouped between horizontal lines. In expression  
 5911 evaluation, where the grammar is formally ambiguous, higher precedence operators shall be  
 5912 evaluated before lower precedence operators. In this table *expr*, *expr1*, *expr2*, and *expr3* represent  
 5913 any expression, while *lvalue* represents any entity that can be assigned to (that is, on the left side  
 5914 of an assignment operator). The precise syntax of expressions is given in **Grammar** (on page  
 5915 2372).

5916 **Table 4-1** Expressions in Decreasing Precedence in *awk*

Syntax	Name	Type of Result	Associativity
( <i>expr</i> )	Grouping	Type of <i>expr</i>	N/A
<i>\$expr</i>	Field reference	String	N/A
++ <i>lvalue</i>	Pre-increment	Numeric	N/A
-- <i>lvalue</i>	Pre-decrement	Numeric	N/A
<i>lvalue</i> ++	Post-increment	Numeric	N/A
<i>lvalue</i> --	Post-decrement	Numeric	N/A
<i>expr</i> ^ <i>expr</i>	Exponentiation	Numeric	Right
! <i>expr</i>	Logical not	Numeric	N/A
+ <i>expr</i>	Unary plus	Numeric	N/A
- <i>expr</i>	Unary minus	Numeric	N/A
<i>expr</i> * <i>expr</i>	Multiplication	Numeric	Left
<i>expr</i> / <i>expr</i>	Division	Numeric	Left
<i>expr</i> % <i>expr</i>	Modulus	Numeric	Left
<i>expr</i> + <i>expr</i>	Addition	Numeric	Left
<i>expr</i> - <i>expr</i>	Subtraction	Numeric	Left
<i>expr</i> <i>expr</i>	String concatenation	String	Left
<i>expr</i> < <i>expr</i>	Less than	Numeric	None
<i>expr</i> <= <i>expr</i>	Less than or equal to	Numeric	None
<i>expr</i> != <i>expr</i>	Not equal to	Numeric	None
<i>expr</i> == <i>expr</i>	Equal to	Numeric	None
<i>expr</i> > <i>expr</i>	Greater than	Numeric	None
<i>expr</i> >= <i>expr</i>	Greater than or equal to	Numeric	None
<i>expr</i> ~ <i>expr</i>	ERE match	Numeric	None
<i>expr</i> !~ <i>expr</i>	ERE non-match	Numeric	None

5943  
5944  
5945  
5946  
5947  
5948  
5949  
5950  
5951  
5952  
5953  
5954  
5955  
5956  
5957  
5958

Syntax	Name	Type of Result	Associativity
<code>expr in array</code>	Array membership	Numeric	Left
<code>( index ) in array</code>	Multi-dimension array membership	Numeric	Left
<code>expr &amp;&amp; expr</code>	Logical AND	Numeric	Left
<code>expr    expr</code>	Logical OR	Numeric	Left
<code>expr1 ? expr2 : expr3</code>	Conditional expression	Type of selected <i>expr2</i> or <i>expr3</i>	Right
<code>lvalue ^= expr</code>	Exponentiation assignment	Numeric	Right
<code>lvalue %= expr</code>	Modulus assignment	Numeric	Right
<code>lvalue *= expr</code>	Multiplication assignment	Numeric	Right
<code>lvalue /= expr</code>	Division assignment	Numeric	Right
<code>lvalue += expr</code>	Addition assignment	Numeric	Right
<code>lvalue -= expr</code>	Subtraction assignment	Numeric	Right
<code>lvalue = expr</code>	Assignment	Type of <i>expr</i>	Right

5959  
5960  
5961  
5962

Each expression shall have either a string value, a numeric value, or both. Except as stated for specific contexts, the value of an expression shall be implicitly converted to the type needed for the context in which it is used. A string value shall be converted to a numeric value by the equivalent of the following calls to functions defined by the ISO C standard:

5963  
5964

```
setlocale(LC_NUMERIC, "");
numeric_value = atof(string_value);
```

5965  
5966  
5967  
5968  
5969  
5970  
5971  
5972  
5973  
5974  
5975

A numeric value that is exactly equal to the value of an integer (see Section 1.7.2 (on page 2207)) shall be converted to a string by the equivalent of a call to the **sprintf** function (see **String Functions** (on page 2369)) with the string "%d" as the *fmt* argument and the numeric value being converted as the first and only *expr* argument. Any other numeric value shall be converted to a string by the equivalent of a call to the **sprintf** function with the value of the variable **CONVFMT** as the *fmt* argument and the numeric value being converted as the first and only *expr* argument. The result of the conversion is unspecified if the value of **CONVFMT** is not a floating-point format specification. This volume of IEEE Std 1003.1-200x specifies no explicit conversions between numbers and strings. An application can force an expression to be treated as a number by adding zero to it, or can force it to be treated as a string by concatenating the null string (" ") to it.

5976  
5977  
5978  
5979  
5980  
5981  
5982  
5983  
5984

A string value shall be considered a *numeric string* if it comes from one of the following:

1. Field variables
2. Input from the *getline()* function
3. **FILENAME**
4. **ARGV** array elements
5. **ENVIRON** array elements
6. Array elements created by the *split()* function
7. A command line variable assignment
8. Variable assignment from another numeric string variable

5985  
5986

and after all the following conversions have been applied, the resulting string would lexically be recognized as a **NUMBER** token as described by the lexical conventions in **Grammar** (on page

5987 2372):

- 5988 • All leading and trailing <blank>s are discarded
- 5989 • If the first non-<blank> is ' + ' or ' - ', it is discarded
- 5990 • Changing each occurrence of the decimal point character from the current locale to a period

5991 If a ' - ' character is ignored in the preceding description, the numeric value of the *numeric string*

5992 shall be the negation of the numeric value of the recognized **NUMBER** token. Otherwise, the

5993 numeric value of the *numeric string* shall be the numeric value of the recognized **NUMBER**

5994 token. Whether or not a string is a *numeric string* shall be relevant only in contexts where that

5995 term is used in this section.

5996 When an expression is used in a Boolean context, if it has a numeric value, a value of zero shall

5997 be treated as false and any other value shall be treated as true. Otherwise, a string value of the

5998 null string shall be treated as false and any other value shall be treated as true. A Boolean

5999 context shall be one of the following:

- 6000 • The first subexpression of a conditional expression
- 6001 • An expression operated on by logical NOT, logical AND, or logical OR
- 6002 • The second expression of a **for** statement
- 6003 • The expression of an **if** statement
- 6004 • The expression of the **while** clause in either a **while** or **do...while** statement
- 6005 • An expression used as a pattern (as in Overall Program Structure)

6006 All arithmetic shall follow the semantics of floating-point arithmetic as specified by the ISO C |

6007 standard (see Section 1.7.2 (on page 2207)). |

6008 The value of the expression:

6009 *expr1* ^ *expr2*

6010 shall be equivalent to the value returned by the ISO C standard function call:

6011 `pow(expr1, expr2)`

6012 The expression:

6013 `lvalue ^= expr` |

6014 shall be equivalent to the ISO C standard expression: |

6015 `lvalue = pow(lvalue, expr)` |

6016 except that `lvalue` shall be evaluated only once. The value of the expression: |

6017 *expr1* % *expr2*

6018 shall be equivalent to the value returned by the ISO C standard function call:

6019 `fmod(expr1, expr2)`

6020 The expression:

6021 `lvalue %= expr` |

6022 shall be equivalent to the ISO C standard expression: |

6023 `lvalue = fmod(lvalue, expr)` |

6024 except that lvalue shall be evaluated only once. |

6025 Variables and fields shall be set by the assignment statement:

6026 `lvalue = expression` |

6027 and the type of *expression* shall determine the resulting variable type. The assignment includes |  
 6028 the arithmetic assignments ("`+=`", "`-=`", "`*=`", "`/=`", "`%=`", "`^=`", "`++`", "`--`") all of which |  
 6029 shall produce a numeric result. The left-hand side of an assignment and the target of increment |  
 6030 and decrement operators can be one of a variable, an array with index, or a field selector.

6031 The *awk* language supplies arrays that are used for storing numbers or strings. Arrays need not |  
 6032 be declared. They shall initially be empty, and their sizes shall change dynamically. The |  
 6033 subscripts, or element identifiers, are strings, providing a type of associative array capability. An |  
 6034 array name followed by a subscript within square brackets can be used as an lvalue and thus as |  
 6035 an expression, as described in the grammar; see **Grammar** (on page 2372). Unsubscripted array |  
 6036 names can be used in only the following contexts:

- 6037 • A parameter in a function definition or function call
- 6038 • The **NAME** token following any use of the keyword **in** as specified in the grammar (see |  
 6039 **Grammar** (on page 2372)); if the name used in this context is not an array name, the behavior |  
 6040 is undefined

6041 A valid array *index* shall consist of one or more comma-separated expressions, similar to the way |  
 6042 in which multi-dimensional arrays are indexed in some programming languages. Because *awk* |  
 6043 arrays are really one-dimensional, such a comma-separated list shall be converted to a single |  
 6044 string by concatenating the string values of the separate expressions, each separated from the |  
 6045 other by the value of the **SUBSEP** variable. Thus, the following two index operations shall be |  
 6046 equivalent:

6047 `var[expr1, expr2, ... exprn]`

6048 `var[expr1 SUBSEP expr2 SUBSEP ... SUBSEP exprn]`

6049 The application shall ensure that a multi-dimensioned *index* used with the **in** operator is |  
 6050 parenthesized. The **in** operator, which tests for the existence of a particular array element, shall |  
 6051 not cause that element to exist. Any other reference to a nonexistent array element shall |  
 6052 automatically create it.

6053 Comparisons (with the '`<`', "`<=`", "`!=`", "`==`", '`>`', and "`>=`" operators) shall be made |  
 6054 numerically if both operands are numeric, if one is numeric and the other has a string value that |  
 6055 is a numeric string, or if one is numeric and the other has the uninitialized value. Otherwise, |  
 6056 operands shall be converted to strings as required and a string comparison shall be made using |  
 6057 the locale-specific collation sequence. The value of the comparison expression shall be 1 if the |  
 6058 relation is true, or 0 if the relation is false.

## 6059 **Variables and Special Variables**

6060 Variables can be used in an *awk* program by referencing them. With the exception of function |  
 6061 parameters (see **User-Defined Functions** (on page 2371)), they are not explicitly declared. |  
 6062 Function parameter names shall be local to the function; all other variable names shall be global. |  
 6063 The same name shall not be used as both a function parameter name and as the name of a |  
 6064 function or a special *awk* variable. The same name shall not be used both as a variable name with |  
 6065 global scope and as the name of a function. The same name shall not be used within the same |  
 6066 scope both as a scalar variable and as an array. Uninitialized variables, including scalar |  
 6067 variables, array elements, and field variables, shall have an uninitialized value. An uninitialized |  
 6068 value shall have both a numeric value of zero and a string value of the empty string. Evaluation

6069 of variables with an uninitialized value, to either string or numeric, shall be determined by the  
6070 context in which they are used.

6071 Field variables shall be designated by a '\$' followed by a number or numerical expression. The  
6072 effect of the field number *expression* evaluating to anything other than a non-negative integer is  
6073 unspecified; uninitialized variables or string values need not be converted to numeric values in  
6074 this context. New field variables can be created by assigning a value to them. References to  
6075 nonexistent fields (that is, fields after \$NF), shall evaluate to the uninitialized value. Such  
6076 references shall not create new fields. However, assigning to a nonexistent field (for example,  
6077 \$(NF+2)=5) shall increase the value of NF; create any intervening fields with the uninitialized  
6078 value; and cause the value of \$0 to be recomputed, with the fields being separated by the value  
6079 of OFS. Each field variable shall have a string value or an uninitialized value when created.  
6080 Field variables shall have the uninitialized value when created from \$0 using FS and the variable  
6081 does not contain any characters. If appropriate, the field variable shall be considered a numeric  
6082 string (see **Expressions in awk** (on page 2358)).

6083 Implementations shall support the following other special variables that are set by *awk*:

6084 **ARGC** The number of elements in the **ARGV** array.

6085 **ARGV** An array of command line arguments, excluding options and the *program*  
6086 argument, numbered from zero to **ARGC**-1.

6087 The arguments in **ARGV** can be modified or added to; **ARGC** can be altered. As  
6088 each input file ends, *awk* shall treat the next non-null element of **ARGV**, up to the  
6089 current value of **ARGC**-1, inclusive, as the name of the next input file. Thus,  
6090 setting an element of **ARGV** to null means that it shall not be treated as an input  
6091 file. The name '-' indicates the standard input. If an argument matches the  
6092 format of an *assignment* operand, this argument shall be treated as an assignment  
6093 rather than a *file* argument.

6094 **CONVFMT** The **printf** format for converting numbers to strings (except for output statements,  
6095 where **OFMT** is used); "% . 6g" by default.

6096 **ENVIRON** An array representing the value of the environment, as described in the *exec* |  
6097 functions defined in the System Interfaces volume of IEEE Std 1003.1-200x. The |  
6098 indices of the array shall be strings consisting of the names of the environment |  
6099 variables, and the value of each array element shall be a string consisting of the |  
6100 value of that variable. If appropriate, the environment variable shall be considered |  
6101 a *numeric string* (see **Expressions in awk** (on page 2358)), the array element shall |  
6102 also have its numeric value.

6103 In all cases where the behavior of *awk* is affected by environment variables  
6104 (including the environment of any commands that *awk* executes via the **system**  
6105 function or via pipeline redirections with the **print** statement, the **printf** statement,  
6106 or the **getline** function), the environment used shall be the environment at the time  
6107 *awk* began executing; it is implementation-defined whether any modification of  
6108 **ENVIRON** affects this environment.

6109 **FILENAME** A pathname of the current input file. Inside a **BEGIN** action the value is  
6110 undefined. Inside an **END** action the value shall be the name of the last input file |  
6111 processed. |

6112 **FNR** The ordinal number of the current record in the current file. Inside a **BEGIN** action |  
6113 the value shall be zero. Inside an **END** action the value shall be the number of the |  
6114 last record processed in the last file processed. |

6115	<b>FS</b>	Input field separator regular expression; a <space> by default.
6116	<b>NF</b>	The number of fields in the current record. Inside a <b>BEGIN</b> action, the use of <b>NF</b> is undefined unless a <b>getline</b> function without a <i>var</i> argument is executed previously. Inside an <b>END</b> action, <b>NF</b> shall retain the value it had for the last record read, unless a subsequent, redirected, <b>getline</b> function without a <i>var</i> argument is performed prior to entering the <b>END</b> action.
6117		
6118		
6119		
6120		
6121	<b>NR</b>	The ordinal number of the current record from the start of input. Inside a <b>BEGIN</b> action the value shall be zero. Inside an <b>END</b> action the value shall be the number of the last record processed.
6122		
6123		
6124	<b>OFMT</b>	The <b>printf</b> format for converting numbers to strings in output statements (see <b>Output Statements</b> (on page 2367)); "%. <i>g</i> " by default. The result of the conversion is unspecified if the value of <b>OFMT</b> is not a floating-point format specification.
6125		
6126		
6127		
6128	<b>OFS</b>	The <b>print</b> statement output field separation; <space> by default.
6129	<b>ORS</b>	The <b>print</b> statement output record separator; a <newline> by default.
6130	<b>RLENGTH</b>	The length of the string matched by the <b>match</b> function.
6131	<b>RS</b>	The first character of the string value of <b>RS</b> shall be the input record separator; a <newline> by default. If <b>RS</b> contains more than one character, the results are unspecified. If <b>RS</b> is null, then records are separated by sequences consisting of a <newline> plus one or more blank lines, leading or trailing blank lines shall not result in empty records at the beginning or end of the input, and a <newline> shall always be a field separator, no matter what the value of <b>FS</b> is.
6132		
6133		
6134		
6135		
6136		
6137	<b>RSTART</b>	The starting position of the string matched by the <b>match</b> function, numbering from 1. This shall always be equivalent to the return value of the <b>match</b> function.
6138		
6139	<b>SUBSEP</b>	The subscript separator string for multi-dimensional arrays; the default value is implementation-defined.
6140		
6141	<b>Regular Expressions</b>	
6142	The <i>awk</i> utility shall make use of the extended regular expression notation (see the Base Definitions volume of IEEE Std 1003.1-200x, Section 9.4, Extended Regular Expressions) except that it shall allow the use of C-language conventions for escaping special characters within the EREs, as specified in the table in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 5, File Format Notation ('\\', '\a', '\b', '\f', '\n', '\r', '\t', '\v') and the following table; these escape sequences shall be recognized both inside and outside bracket expressions. Note that records need not be separated by <newline>s and string constants can contain <newline>s, so even the "\n" sequence is valid in <i>awk</i> EREs. Using a slash character within an ERE requires the escaping shown in the following table.	
6143		
6144		
6145		
6146		
6147		
6148		
6149		
6150		

6151

Table 4-2 Escape Sequences in *awk*

6152

6153

6154

6155

6156

6157

6158

6159

6160

6161

6162

6163

6164

6165

6166

6167

6168

6169

6170

6171

6172

Escape Sequence	Description	Meaning
\ "	Backslash quotation-mark	Quotation-mark character
\ /	Backslash slash	Slash character
\ ddd	A backslash character followed by the longest sequence of one, two, or three octal-digit characters (01234567). If all of the digits are 0 (that is, representation of the NUL character), the behavior is undefined.	The character whose encoding is represented by the one, two, or three-digit octal integer. If the size of a byte on the system is greater than nine bits, the valid escape sequence used to represent a byte is implementation-defined. Multi-byte characters require multiple, concatenated escape sequences of this type, including the leading '\ ' for each byte.
\ c	A backslash character followed by any character not described in this table or in the table in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 5, File Format Notation ('\\', '\a', '\b', '\f', '\n', '\r', '\t', '\v')	Undefined

6173

6174

6175

6176

6177

6178

6179

6180

6181

6182

6183

6184

6185

6186

6187

A regular expression can be matched against a specific field or string by using one of the two regular expression matching operators, '*~*' and '!*~*'. These operators shall interpret their right-hand operand as a regular expression and their left-hand operand as a string. If the regular expression matches the string, the '*~*' expression shall evaluate to a value of 1, and the '!*~*' expression shall evaluate to a value of 0. (The regular expression matching operation is as defined by the term *matched* in the Base Definitions volume of IEEE Std 1003.1-200x, Section 9.1, Regular Expression Definitions, where a match occurs on any part of the string unless the regular expression is limited with the circumflex or dollar sign special characters.) If the regular expression does not match the string, the '*~*' expression shall evaluate to a value of 0, and the '!*~*' expression shall evaluate to a value of 1. If the right-hand operand is any expression other than the lexical token **ERE**, the string value of the expression shall be interpreted as an extended regular expression, including the escape conventions described above. Note that these same escape conventions shall also be applied in determining the value of a string literal (the lexical token **STRING**), and thus shall be applied a second time when a string literal is used in this context.

6188

6189

6190

When an **ERE** token appears as an expression in any context other than as the right-hand of the '*~*' or '!*~*' operator or as one of the built-in function arguments described below, the value of the resulting expression shall be the equivalent of:

6191

```
$0 ~ /ere/
```

6192

6193

6194

6195

The *ere* argument to the **gsub**, **match**, **sub** functions, and the *fs* argument to the **split** function (see **String Functions** (on page 2369)) shall be interpreted as extended regular expressions. These can be either **ERE** tokens or arbitrary expressions, and shall be interpreted in the same manner as the right-hand side of the '*~*' or '!*~*' operator.

6196

6197

6198

An extended regular expression can be used to separate fields by using the **-F ERE** option or by assigning a string containing the expression to the built-in variable **FS**. The default value of the **FS** variable shall be a single <space>. The following describes **FS** behavior:

- 6199 1. If **FS** is a null string, the behavior is unspecified.
- 6200 2. If **FS** is a single character:
- 6201 a. If **FS** is <space>, skip leading and trailing <blank>s; fields shall be delimited by sets  
6202 of one or more <blank>s.
- 6203 b. Otherwise, if **FS** is any other character *c*, fields shall be delimited by each single  
6204 occurrence of *c*.
- 6205 3. Otherwise, the string value of **FS** shall be considered to be an extended regular expression.  
6206 Each occurrence of a sequence matching the extended regular expression shall delimit  
6207 fields.

6208 Except for the '*~*' and '!~' operators, and in the **gsub**, **match**, **split**, and **sub** built-in functions,  
6209 ERE matching shall be based on input records; that is, record separator characters (the first  
6210 character of the value of the variable **RS**, <newline> by default) cannot be embedded in the  
6211 expression, and no expression shall match the record separator character. If the record separator  
6212 is not <newline>, <newline>s embedded in the expression can be matched. For the '*~*' and  
6213 '!~' operators, and in those four built-in functions, ERE matching shall be based on text strings;  
6214 that is, any character (including <newline> and the record separator) can be embedded in the  
6215 pattern, and an appropriate pattern shall match any character. However, in all *awk* ERE  
6216 matching, the use of one or more NUL characters in the pattern, input record, or text string  
6217 produces undefined results.

## 6218 **Patterns**

6219 A *pattern* is any valid *expression*, a range specified by two expressions separated by comma, or  
6220 one of the two special patterns **BEGIN** or **END**.

## 6221 **Special Patterns**

6222 The *awk* utility shall recognize two special patterns, **BEGIN** and **END**. Each **BEGIN** pattern  
6223 shall be matched once and its associated action executed before the first record of input is read  
6224 (except possibly by use of the **getline** function—see **Input/Output and General Functions** (on  
6225 page 2370)—in a prior **BEGIN** action) and before command line assignment is done. Each **END**  
6226 pattern shall be matched once and its associated action executed after the last record of input has  
6227 been read. These two patterns shall have associated actions.

6228 **BEGIN** and **END** shall not combine with other patterns. Multiple **BEGIN** and **END** patterns  
6229 shall be allowed. The actions associated with the **BEGIN** patterns shall be executed in the order  
6230 specified in the program, as are the **END** actions. An **END** pattern can precede a **BEGIN** pattern  
6231 in a program.

6232 If an *awk* program consists of only actions with the pattern **BEGIN**, and the **BEGIN** action  
6233 contains no **getline** function, *awk* shall exit without reading its input when the last statement in  
6234 the last **BEGIN** action is executed. If an *awk* program consists of only actions with the pattern  
6235 **END** or only actions with the patterns **BEGIN** and **END**, the input shall be read before the  
6236 statements in the **END** actions are executed.

6237 **Expression Patterns**

6238 An expression pattern shall be evaluated as if it were an expression in a Boolean context. If the  
 6239 result is true, the pattern shall be considered to match, and the associated action (if any) shall be  
 6240 executed. If the result is false, the action shall not be executed.

6241 **Pattern Ranges**

6242 A pattern range consists of two expressions separated by a comma; in this case, the action shall  
 6243 be performed for all records between a match of the first expression and the following match of  
 6244 the second expression, inclusive. At this point, the pattern range can be repeated starting at  
 6245 input records subsequent to the end of the matched range.

6246 **Actions**

6247 An action is a sequence of statements as shown in the grammar in **Grammar** (on page 2372).  
 6248 Any single statement can be replaced by a statement list enclosed in braces. The application shall  
 6249 ensure that statements in a statement list are separated by <newline>s or semicolons. Statements  
 6250 in a statement list shall be executed sequentially in the order that they appear. |

6251 The *expression* acting as the conditional in an **if** statement shall be evaluated and if it is non-zero  
 6252 or non-null, the following *statement* shall be executed; otherwise, if **else** is present, the statement  
 6253 following the **else** shall be executed.

6254 The **if**, **while**, **do...while**, **for**, **break**, and **continue** statements are based on the ISO C standard |  
 6255 (see Section 1.7.2 (on page 2207)), except that the Boolean expressions shall be treated as |  
 6256 described in **Expressions in awk** (on page 2358), and except in the case of: |

6257 `for (variable in array)`

6258 which shall iterate, assigning each *index of array* to *variable* in an unspecified order. The results of  
 6259 adding new elements to *array* within such a **for** loop are undefined. If a **break** or **continue**  
 6260 statement occurs outside of a loop, the behavior is undefined.

6261 The **delete** statement shall remove an individual array element. Thus, the following code deletes  
 6262 an entire array:

```
6263 for (index in array)
6264     delete array[index]
```

6265 The **next** statement shall cause all further processing of the current input record to be  
 6266 abandoned. The behavior is undefined if a **next** statement appears or is invoked in a **BEGIN** or  
 6267 **END** action.

6268 The **exit** statement shall invoke all **END** actions in the order in which they occur in the program  
 6269 source and then terminate the program without reading further input. An **exit** statement inside  
 6270 an **END** action shall terminate the program without further execution of **END** actions. If an  
 6271 expression is specified in an **exit** statement, its numeric value shall be the exit status of **awk**,  
 6272 unless subsequent errors are encountered or a subsequent **exit** statement with an expression is  
 6273 executed.

6274 **Output Statements**

6275 Both **print** and **printf** statements shall write to standard output by default. The output shall be  
 6276 written to the location specified by *output\_redirection* if one is supplied, as follows:

```
6277 > expression
6278 >> expression
6279 | expression
```

6280 In all cases, the *expression* shall be evaluated to produce a string that is used as a pathname into  
 6281 which to write (for '**>**' or "**>>**") or as a command to be executed (for '**|**'). Using the first two  
 6282 forms, if the file of that name is not currently open, it shall be opened, creating it if necessary and  
 6283 using the first form, truncating the file. The output then shall be appended to the file. As long as  
 6284 the file remains open, subsequent calls in which *expression* evaluates to the same string value  
 6285 shall simply append output to the file. The file remains open until the **close** function (see  
 6286 **Input/Output and General Functions** (on page 2370)) is called with an expression that evaluates  
 6287 to the same string value.

6288 The third form shall write output onto a stream piped to the input of a command. The stream  
 6289 shall be created if no stream is currently open with the value of *expression* as its command name.  
 6290 The stream created shall be equivalent to one created by a call to the *popen()* function defined in  
 6291 the System Interfaces volume of IEEE Std 1003.1-200x with the value of *expression* as the  
 6292 *command* argument and a value of *w* as the *mode* argument. As long as the stream remains open,  
 6293 subsequent calls in which *expression* evaluates to the same string value shall write output to the  
 6294 existing stream. The stream shall remain open until the **close** function (see **Input/Output and**  
 6295 **General Functions** (on page 2370)) is called with an expression that evaluates to the same string  
 6296 value. At that time, the stream shall be closed as if by a call to the *pclose()* function defined in  
 6297 the System Interfaces volume of IEEE Std 1003.1-200x.

6298 As described in detail by the grammar in **Grammar** (on page 2372), these output statements shall  
 6299 take a comma-separated list of *expressions* referred to in the grammar by the non-terminal  
 6300 symbols **expr\_list**, **print\_expr\_list**, or **print\_expr\_list\_opt**. This list is referred to here as the  
 6301 *expression list*, and each member is referred to as an *expression argument*.

6302 The **print** statement shall write the value of each expression argument onto the indicated output  
 6303 stream separated by the current output field separator (see variable **OFS** above), and terminated  
 6304 by the output record separator (see variable **ORS** above). All expression arguments shall be  
 6305 taken as strings, being converted if necessary; this conversion shall be as described in  
 6306 **Expressions in awk** (on page 2358), with the exception that the **printf** format in **OFMT** shall be  
 6307 used instead of the value in **CONVFMT**. An empty expression list shall stand for the whole  
 6308 input record (\$0).

6309 The **printf** statement shall produce output based on a notation similar to the File Format  
 6310 Notation used to describe file formats in this volume of IEEE Std 1003.1-200x (see the Base  
 6311 Definitions volume of IEEE Std 1003.1-200x, Chapter 5, File Format Notation). Output shall be  
 6312 produced as specified with the first expression argument as the string *format* and subsequent  
 6313 expression arguments as the strings *arg1* to *argn*, inclusive, with the following exceptions:

- 6314 1. The *format* shall be an actual character string rather than a graphical representation.  
 6315 Therefore, it cannot contain empty character positions. The **<space>** in the *format* string, in  
 6316 any context other than a *flag* of a conversion specification, shall be treated as an ordinary  
 6317 character that is copied to the output.
- 6318 2. If the character set contains a '**Δ**' character and that character appears in the *format* string,  
 6319 it shall be treated as an ordinary character that is copied to the output.

- 6320 3. The *escape sequences* beginning with a backslash character shall be treated as sequences of  
 6321 ordinary characters that are copied to the output. Note that these same sequences shall be  
 6322 interpreted lexically by *awk* when they appear in literal strings, but they shall not be  
 6323 treated specially by the **printf** statement.
- 6324 4. A *field width* or *precision* can be specified as the '\*' character instead of a digit string. In  
 6325 this case the next argument from the expression list shall be fetched and its numeric value  
 6326 taken as the field width or precision.
- 6327 5. The implementation shall not precede or follow output from the *d* or *u* conversion  
 6328 specifications with <blank>s not specified by the *format* string.
- 6329 6. The implementation shall not precede output from the *o* conversion specification with  
 6330 leading zeros not specified by the *format* string.
- 6331 7. For the *c* conversion specification: if the argument has a numeric value, the character  
 6332 whose encoding is that value shall be output. If the value is zero or is not the encoding of  
 6333 any character in the character set, the behavior is undefined. If the argument does not have  
 6334 a numeric value, the first character of the string value shall be output; if the string does not  
 6335 contain any characters, the behavior is undefined.
- 6336 8. For each conversion specification that consumes an argument, the next expression  
 6337 argument shall be evaluated. With the exception of the *c* conversion, the value shall be  
 6338 converted (according to the rules specified in **Expressions in awk** (on page 2358)) to the  
 6339 appropriate type for the conversion specification.
- 6340 9. If there are insufficient expression arguments to satisfy all the conversion specifications in  
 6341 the *format* string, the behavior is undefined.
- 6342 10. If any character sequence in the *format* string begins with a '%' character, but does not  
 6343 form a valid conversion specification, the behavior is unspecified.

6344 Both **print** and **printf** can output at least {LINE\_MAX} bytes.

## 6345 **Functions**

6346 The *awk* language has a variety of built-in functions: arithmetic, string, input/output, and  
 6347 general.

### 6348 **Arithmetic Functions**

6349 The arithmetic functions, except for **int**, shall be based on the ISO C standard (see Section 1.7.2 |  
 6350 (on page 2207)). The behavior is undefined in cases where the ISO C standard specifies that an |  
 6351 error be returned or that the behavior is undefined. Although the grammar (see **Grammar** (on |  
 6352 page 2372)) permits built-in functions to appear with no arguments or parentheses, unless the |  
 6353 argument or parentheses are indicated as optional in the following list (by displaying them |  
 6354 within the "[ ]" brackets), such use is undefined.

- |      |                             |   |  |
|------|-----------------------------|---|--|
| 6355 | <b>atan2</b> ( <i>y,x</i> ) | Return arctangent of $y/x$ in radians in the range $[-\pi,\pi]$ . |  |
| 6356 | <b>cos</b> ( <i>x</i> )     | Return cosine of <i>x</i> , where <i>x</i> is in radians.         |  |
| 6357 | <b>sin</b> ( <i>x</i> )     | Return sine of <i>x</i> , where <i>x</i> is in radians.           |  |
| 6358 | <b>exp</b> ( <i>x</i> )     | Return the exponential function of <i>x</i> .                     |  |
| 6359 | <b>log</b> ( <i>x</i> )     | Return the natural logarithm of <i>x</i> .                        |  |
| 6360 | <b>sqrt</b> ( <i>x</i> )    | Return the square root of <i>x</i> .                              |  |

6361 **int(x)** Return the argument truncated to an integer. Truncation shall be toward 0 when  
6362  $x > 0$ .

6363 **rand()** Return a random number  $n$ , such that  $0 \leq n < 1$ .

6364 **srand([expr])** Set the seed value for *rand* to *expr* or use the time of day if *expr* is omitted. The  
6365 previous seed value shall be returned.

### 6366 **String Functions**

6367 The string functions in the following list shall be supported. Although the grammar (see  
6368 **Grammar** (on page 2372)) permits built-in functions to appear with no arguments or  
6369 parentheses, unless the argument or parentheses are indicated as optional in the following list  
6370 (by displaying them within the "[ ]" brackets), such use is undefined.

6371 **gsub(ere, repl[, in])**

6372 Behave like **sub** (see below), except that it shall replace all occurrences of the  
6373 regular expression (like the *ed* utility global substitute) in  $\$0$  or in the *in* argument,  
6374 when specified.

6375 **index(s, t)** Return the position, in characters, numbering from 1, in string *s* where string *t* first  
6376 occurs, or zero if it does not occur at all.

6377 **length([s])** Return the length, in characters, of its argument taken as a string, or of the whole  
6378 record,  $\$0$ , if there is no argument.

6379 **match(s, ere)** Return the position, in characters, numbering from 1, in string *s* where the  
6380 extended regular expression *ere* occurs, or zero if it does not occur at all. RSTART  
6381 shall be set to the starting position (which is the same as the returned value), zero  
6382 if no match is found; RLENGTH shall be set to the length of the matched string, -1  
6383 if no match is found.

6384 **split(s, a[, fs])**

6385 Split the string *s* into array elements  $a[1]$ ,  $a[2]$ , ...,  $a[n]$ , and return  $n$ . All elements  
6386 of the array shall be deleted before the split is performed. The separation shall be  
6387 done with the ERE *fs* or with the field separator **FS** if *fs* is not given. Each array  
6388 element shall have a string value when created and, if appropriate, the array  
6389 element shall be considered a numeric string (see **Expressions in awk** (on page  
6390 2358)). The effect of a null string as the value of *fs* is unspecified.

6391 **sprintf(fmt, expr, expr, ...)**

6392 Format the expressions according to the **printf** format given by *fmt* and return the  
6393 resulting string.

6394 **sub(ere, repl[, in])**

6395 Substitute the string *repl* in place of the first instance of the extended regular  
6396 expression *ERE* in string *in* and return the number of substitutions. An ampersand  
6397 ('&') appearing in the string *repl* shall be replaced by the string from *in* that  
6398 matches the ERE. An ampersand preceded with a backslash ('\&') shall be  
6399 interpreted as the literal ampersand character. An occurrence of two consecutive  
6400 backslashes shall be interpreted as just a single literal backslash character. Any  
6401 other occurrence of a backslash (for example, preceding any other character) shall  
6402 be treated as a literal backslash character. Note that if *repl* is a string literal (the  
6403 lexical token **STRING**; see **Grammar** (on page 2372)), the handling of the  
6404 ampersand character occurs after any lexical processing, including any lexical  
6405 backslash escape sequence processing. If *in* is specified and it is not an lvalue (see  
6406 **Expressions in awk** (on page 2358)), the behavior is undefined. If *in* is omitted, *awk*

- 6407 shall use the current record (\$0) in its place.
- 6408 **substr**(*s*, *m* [, *n* ] )
- 6409 Return the at most *n*-character substring of *s* that begins at position *m*, numbering  
6410 from 1. If *n* is omitted, or if *n* specifies more characters than are left in the string,  
6411 the length of the substring shall be limited by the length of the string *s*.
- 6412 **tolower**(*s*) Return a string based on the string *s*. Each character in *s* that is an uppercase letter  
6413 specified to have a **tolower** mapping by the *LC\_CTYPE* category of the current  
6414 locale shall be replaced in the returned string by the lowercase letter specified by  
6415 the mapping. Other characters in *s* shall be unchanged in the returned string.
- 6416 **toupper**(*s*) Return a string based on the string *s*. Each character in *s* that is a lowercase letter  
6417 specified to have a **toupper** mapping by the *LC\_CTYPE* category of the current  
6418 locale is replaced in the returned string by the uppercase letter specified by the  
6419 mapping. Other characters in *s* are unchanged in the returned string.
- 6420 All of the preceding functions that take *ERE* as a parameter expect a pattern or a string valued  
6421 expression that is a regular expression as defined in **Regular Expressions** (on page 2363).
- 6422 **Input/Output and General Functions**
- 6423 The input/output and general functions are:
- 6424 **close**(*expression*)
- 6425 Close the file or pipe opened by a **print** or **printf** statement or a call to **getline** with  
6426 the same string-valued *expression*. The limit on the number of open *expression*  
6427 arguments is implementation-defined. If the close was successful, the function  
6428 shall return zero; otherwise, it shall return non-zero.
- 6429 *expression* / **getline** [*var*]
- 6430 Read a record of input from a stream piped from the output of a command. The  
6431 stream shall be created if no stream is currently open with the value of *expression* as  
6432 its command name. The stream created shall be equivalent to one created by a call  
6433 to the *popen*() function with the value of *expression* as the *command* argument and a  
6434 value of *r* as the *mode* argument. As long as the stream remains open, subsequent  
6435 calls in which *expression* evaluates to the same string value shall read subsequent  
6436 records from the stream. The stream shall remain open until the **close** function is  
6437 called with an expression that evaluates to the same string value. At that time, the  
6438 stream shall be closed as if by a call to the *pclose*() function. If *var* is omitted, \$0  
6439 and **NF** shall be set; otherwise, *var* shall be set and, if appropriate, it shall be  
6440 considered a numeric string (see **Expressions in awk** (on page 2358)).
- 6441 The **getline** operator can form ambiguous constructs when there are  
6442 unparenthesized operators (including concatenate) to the left of the ' | ' (to the  
6443 beginning of the expression containing **getline**). In the context of the '\$'  
6444 operator, ' | ' shall behave as if it had a lower precedence than '\$'. The result of  
6445 evaluating other operators is unspecified, and conforming applications shall  
6446 parenthesize properly all such usages.
- 6447 **getline** Set \$0 to the next input record from the current input file. This form of **getline** shall  
6448 set the **NF**, **NR**, and **FNR** variables.
- 6449 **getline var** Set variable *var* to the next input record from the current input file and, if  
6450 appropriate, *var* shall be considered a numeric string (see **Expressions in awk** (on  
6451 page 2358)). This form of **getline** shall set the **FNR** and **NR** variables.

6452 **getline** [*var*] < *expression*

6453 Read the next record of input from a named file. The *expression* shall be evaluated  
 6454 to produce a string that is used as a pathname. If the file of that name is not  
 6455 currently open, it shall be opened. As long as the stream remains open, subsequent  
 6456 calls in which *expression* evaluates to the same string value shall read subsequent  
 6457 records from the file. The file shall remain open until the **close** function is called  
 6458 with an expression that evaluates to the same string value. If *var* is omitted, \$0 and  
 6459 **NF** shall be set; otherwise, *var* shall be set and, if appropriate, it shall be considered  
 6460 a numeric string (see **Expressions in awk** (on page 2358)).

6461 The **getline** operator can form ambiguous constructs when there are  
 6462 unparenthesized binary operators (including concatenate) to the right of the '<'  
 6463 (up to the end of the expression containing the **getline**). The result of evaluating  
 6464 such a construct is unspecified, and conforming applications shall parenthesize  
 6465 properly all such usages.

6466 **system**(*expression*)

6467 Execute the command given by *expression* in a manner equivalent to the *system*()  
 6468 function defined in the System Interfaces volume of IEEE Std 1003.1-200x and  
 6469 return the exit status of the command.

6470 All forms of **getline** shall return 1 for successful input, zero for end-of-file, and -1 for an error.

6471 Where strings are used as the name of a file or pipeline, the application shall ensure that the  
 6472 strings are textually identical. The terminology “same string value” implies that “equivalent  
 6473 strings”, even those that differ only by <space>s, represent different files.

## 6474 **User-Defined Functions**

6475 The *awk* language also provides user-defined functions. Such functions can be defined as:

```
6476 function name([parameter, ...]) { statements }
```

6477 A function can be referred to anywhere in an *awk* program; in particular, its use can precede its  
 6478 definition. The scope of a function is global.

6479 Function parameters, if present, can be either scalars or arrays; the behavior is undefined if an  
 6480 array name is passed as a parameter that the function uses as a scalar, or if a scalar expression is  
 6481 passed as a parameter that the function uses as an array. Function parameters shall be passed by  
 6482 value if scalar and by reference if array name.

6483 The number of parameters in the function definition need not match the number of parameters  
 6484 in the function call. Excess formal parameters can be used as local variables. If fewer arguments  
 6485 are supplied in a function call than are in the function definition, the extra parameters that are  
 6486 used in the function body as scalars shall evaluate to the uninitialized value until they are  
 6487 otherwise initialized, and the extra parameters that are used in the function body as arrays shall  
 6488 be treated as uninitialized arrays where each element evaluates to the uninitialized value until  
 6489 otherwise initialized.

6490 When invoking a function, no white space can be placed between the function name and the  
 6491 opening parenthesis. Function calls can be nested and recursive calls can be made upon  
 6492 functions. Upon return from any nested or recursive function call, the values of all of the calling  
 6493 function's parameters shall be unchanged, except for array parameters passed by reference. The  
 6494 **return** statement can be used to return a value. If a **return** statement appears outside of a  
 6495 function definition, the behavior is undefined.

6496 In the function definition, <newline>s shall be optional before the opening brace and after the  
 6497 closing brace. Function definitions can appear anywhere in the program where a *pattern-action*

6498 pair is allowed.

### 6499 Grammar

6500 The grammar in this section and the lexical conventions in the following section shall together  
 6501 describe the syntax for *awk* programs. The general conventions for this style of grammar are  
 6502 described in Section 1.10 (on page 2220). A valid program can be represented as the non-  
 6503 terminal symbol *program* in the grammar. This formal syntax shall take precedence over the  
 6504 preceding text syntax description.

```

6505 %token NAME NUMBER STRING ERE
6506 %token FUNC_NAME /* Name followed by '(' without white space. */

6507 /* Keywords */
6508 %token Begin End
6509 /* 'BEGIN' 'END' */

6510 %token Break Continue Delete Do Else
6511 /* 'break' 'continue' 'delete' 'do' 'else' */

6512 %token Exit For Function If In
6513 /* 'exit' 'for' 'function' 'if' 'in' */

6514 %token Next Print Printf Return While
6515 /* 'next' 'print' 'printf' 'return' 'while' */

6516 /* Reserved function names */
6517 %token BUILTIN_FUNC_NAME
6518 /* One token for the following:
6519 * atan2 cos sin exp log sqrt int rand srand
6520 * gsub index length match split sprintf sub
6521 * substr tolower toupper close system
6522 */
6523 %token GETLINE
6524 /* Syntactically different from other built-ins. */

6525 /* Two-character tokens. */
6526 %token ADD_ASSIGN SUB_ASSIGN MUL_ASSIGN DIV_ASSIGN MOD_ASSIGN POW_ASSIGN
6527 /* '+' '-' '*' '/' '%' '^' */

6528 %token OR AND NO_MATCH EQ LE GE NE INCR DECR APPEND
6529 /* '|' '&&' '!~' '==' '<=' '>=' '!=' '++' '--' '>>' */

6530 /* One-character tokens. */
6531 %token '{' '}' '(' ')' '[' ']' ',' ';' NEWLINE
6532 %token '+' '-' '*' '%' '^' '!' '>' '<' '|' '?' ':' '~' '$' '='

6533 %start program
6534 %%

6535 program : item_list
6536 | actionless_item_list
6537 ;

6538 item_list : newline_opt
6539 | actionless_item_list item terminator
6540 | item_list item terminator
6541 | item_list action terminator
6542 ;

```

```

6543     actionless_item_list : item_list           pattern terminator
6544         | actionless_item_list pattern terminator
6545         ;

6546     item                   : pattern action
6547         | Function NAME      '(' param_list_opt ')'
6548           newline_opt action
6549         | Function FUNC_NAME '(' param_list_opt ')'
6550           newline_opt action
6551         ;

6552     param_list_opt        : /* empty */
6553         | param_list
6554         ;

6555     param_list            : NAME
6556         | param_list ',' NAME
6557         ;

6558     pattern               : Begin
6559         | End
6560         | expr
6561         | expr ',' newline_opt expr
6562         ;

6563     action                 : '{' newline_opt
6564         | '{' newline_opt terminated_statement_list
6565         | '{' newline_opt unterminated_statement_list
6566         ;

6567     terminator            : terminator ';'
6568         | terminator NEWLINE
6569         |
6570           ';'
6571         |
6572           NEWLINE
6571         ;

6572     terminated_statement_list : terminated_statement
6573         | terminated_statement_list terminated_statement
6574         ;

6575     unterminated_statement_list : unterminated_statement
6576         | terminated_statement_list unterminated_statement
6577         ;

6578     terminated_statement  : action newline_opt
6579         | If '(' expr ')' newline_opt terminated_statement
6580         | If '(' expr ')' newline_opt terminated_statement
6581           Else newline_opt terminated_statement
6582         | While '(' expr ')' newline_opt terminated_statement
6583         | For '(' simple_statement_opt ';'
6584           expr_opt ';' simple_statement_opt ')' newline_opt
6585           terminated_statement
6586         | For '(' NAME In NAME ')' newline_opt
6587           terminated_statement
6588         | ';' newline_opt
6589         | terminatable_statement NEWLINE newline_opt
6590         | terminatable_statement ';'      newline_opt

```

```

6591         ;
6592     unterminated_statement : terminatable_statement
6593         | If '(' expr ')' newline_opt unterminated_statement
6594         | If '(' expr ')' newline_opt terminated_statement
6595         | Else newline_opt unterminated_statement
6596         | While '(' expr ')' newline_opt unterminated_statement
6597         | For '(' simple_statement_opt ';'
6598         |   expr_opt ';' simple_statement_opt ')' newline_opt
6599         |   unterminated_statement
6600         | For '(' NAME In NAME ')' newline_opt
6601         |   unterminated_statement
6602         ;
6603     terminatable_statement : simple_statement
6604         | Break
6605         | Continue
6606         | Next
6607         | Exit expr_opt
6608         | Return expr_opt
6609         | Do newline_opt terminated_statement While '(' expr ')'
6610         ;
6611     simple_statement_opt : /* empty */
6612         | simple_statement
6613         ;
6614     simple_statement : Delete NAME '[' expr_list '['
6615         | expr
6616         | print_statement
6617         ;
6618     print_statement : simple_print_statement
6619         | simple_print_statement output_redirection
6620         ;
6621     simple_print_statement : Print print_expr_list_opt
6622         | Print '(' multiple_expr_list ')'
6623         | Printf print_expr_list
6624         | Printf '(' multiple_expr_list ')'
6625         ;
6626     output_redirection : '>' expr
6627         | APPEND expr
6628         | '|' expr
6629         ;
6630     expr_list_opt : /* empty */
6631         | expr_list
6632         ;
6633     expr_list : expr
6634         | multiple_expr_list
6635         ;
6636     multiple_expr_list : expr ',' newline_opt expr
6637         | multiple_expr_list ',' newline_opt expr

```

```

6638          ;
6639          expr_opt      : /* empty */
6640                        | expr
6641          ;
6642          expr          : unary_expr
6643                        | non_unary_expr
6644          ;
6645          unary_expr    : '+' expr
6646                        | '-' expr
6647                        | unary_expr '^'      expr
6648                        | unary_expr '*'      expr
6649                        | unary_expr '/'      expr
6650                        | unary_expr '%'      expr
6651                        | unary_expr '+'      expr
6652                        | unary_expr '-'      expr
6653                        | unary_expr          non_unary_expr
6654                        | unary_expr '<'      expr
6655                        | unary_expr LE      expr
6656                        | unary_expr NE      expr
6657                        | unary_expr EQ      expr
6658                        | unary_expr '>'      expr
6659                        | unary_expr GE      expr
6660                        | unary_expr '~'      expr
6661                        | unary_expr NO_MATCH expr
6662                        | unary_expr In NAME
6663                        | unary_expr AND newline_opt expr
6664                        | unary_expr OR  newline_opt expr
6665                        | unary_expr '?' expr ':' expr
6666                        | unary_input_function
6667          ;
6668          non_unary_expr : '(' expr ')'
6669                        | '!' expr
6670                        | non_unary_expr '^'      expr
6671                        | non_unary_expr '*'      expr
6672                        | non_unary_expr '/'      expr
6673                        | non_unary_expr '%'      expr
6674                        | non_unary_expr '+'      expr
6675                        | non_unary_expr '-'      expr
6676                        | non_unary_expr          non_unary_expr
6677                        | non_unary_expr '<'      expr
6678                        | non_unary_expr LE      expr
6679                        | non_unary_expr NE      expr
6680                        | non_unary_expr EQ      expr
6681                        | non_unary_expr '>'      expr
6682                        | non_unary_expr GE      expr
6683                        | non_unary_expr '~'      expr
6684                        | non_unary_expr NO_MATCH expr
6685                        | non_unary_expr In NAME
6686                        | '(' multiple_expr_list ')' In NAME
6687                        | non_unary_expr AND newline_opt expr

```

```

6688         | non_unary_expr OR newline_opt expr
6689         | non_unary_expr '?' expr ':' expr
6690         | NUMBER
6691         | STRING
6692         | lvalue
6693         | ERE
6694         | lvalue INCR
6695         | lvalue DECR
6696         | INCR lvalue
6697         | DECR lvalue
6698         | lvalue POW_ASSIGN expr
6699         | lvalue MOD_ASSIGN expr
6700         | lvalue MUL_ASSIGN expr
6701         | lvalue DIV_ASSIGN expr
6702         | lvalue ADD_ASSIGN expr
6703         | lvalue SUB_ASSIGN expr
6704         | lvalue '=' expr
6705         | FUNC_NAME '(' expr_list_opt ')'
6706         | /* no white space allowed before '(' */
6707         | BUILTIN_FUNC_NAME '(' expr_list_opt ')'
6708         | BUILTIN_FUNC_NAME
6709         | non_unary_input_function
6710         ;

6711     print_expr_list_opt : /* empty */
6712         | print_expr_list
6713         ;

6714     print_expr_list : print_expr
6715         | print_expr_list ',' newline_opt print_expr
6716         ;

6717     print_expr : unary_print_expr
6718         | non_unary_print_expr
6719         ;

6720     unary_print_expr : '+' print_expr
6721         | '-' print_expr
6722         | unary_print_expr '^' print_expr
6723         | unary_print_expr '*' print_expr
6724         | unary_print_expr '/' print_expr
6725         | unary_print_expr '%' print_expr
6726         | unary_print_expr '+' print_expr
6727         | unary_print_expr '-' print_expr
6728         | unary_print_expr non_unary_print_expr
6729         | unary_print_expr '~' print_expr
6730         | unary_print_expr NO_MATCH print_expr
6731         | unary_print_expr In NAME
6732         | unary_print_expr AND newline_opt print_expr
6733         | unary_print_expr OR newline_opt print_expr
6734         | unary_print_expr '?' print_expr ':' print_expr
6735         ;

6736     non_unary_print_expr : '(' expr ')'
6737         | '!' print_expr

```

```

6738         | non_unary_print_expr '^'      print_expr
6739         | non_unary_print_expr '*'      print_expr
6740         | non_unary_print_expr '/'      print_expr
6741         | non_unary_print_expr '%'      print_expr
6742         | non_unary_print_expr '+'      print_expr
6743         | non_unary_print_expr '-'      print_expr
6744         | non_unary_print_expr      non_unary_print_expr
6745         | non_unary_print_expr '~'      print_expr
6746         | non_unary_print_expr NO_MATCH print_expr
6747         | non_unary_print_expr In NAME
6748         | '(' multiple_expr_list ')' In NAME
6749         | non_unary_print_expr AND newline_opt print_expr
6750         | non_unary_print_expr OR  newline_opt print_expr
6751         | non_unary_print_expr '?' print_expr ':' print_expr
6752         | NUMBER
6753         | STRING
6754         | lvalue
6755         | ERE
6756         | lvalue INCR
6757         | lvalue DECR
6758         | INCR lvalue
6759         | DECR lvalue
6760         | lvalue POW_ASSIGN print_expr
6761         | lvalue MOD_ASSIGN print_expr
6762         | lvalue MUL_ASSIGN print_expr
6763         | lvalue DIV_ASSIGN print_expr
6764         | lvalue ADD_ASSIGN print_expr
6765         | lvalue SUB_ASSIGN print_expr
6766         | lvalue '=' print_expr
6767         | FUNC_NAME '(' expr_list_opt ')'
6768         | /* no white space allowed before '(' */
6769         | BUILTIN_FUNC_NAME '(' expr_list_opt ')'
6770         | BUILTIN_FUNC_NAME
6771         ;

6772     lvalue      : NAME
6773         | NAME '[' expr_list ']'
6774         | '$' expr
6775         ;

6776     non_unary_input_function : simple_get
6777         | simple_get '<' expr
6778         | non_unary_expr '|' simple_get
6779         ;

6780     unary_input_function : unary_expr '|' simple_get
6781         ;

6782     simple_get      : GETLINE
6783         | GETLINE lvalue
6784         ;

6785     newline_opt    : /* empty */
6786         | newline_opt NEWLINE
6787         ;

```

6788 This grammar has several ambiguities that shall be resolved as follows:

- 6789 • Operator precedence and associativity shall be as described in Table 4-1 (on page 2358).
- 6790 • In case of ambiguity, an **else** shall be associated with the most immediately preceding **if** that
- 6791 would satisfy the grammar.
- 6792 • In some contexts, a slash ( '/' ) that is used to surround an ERE could also be the division
- 6793 operator. This shall be resolved in such a way that wherever the division operator could
- 6794 appear, a slash is assumed to be the division operator. (There is no unary division operator.)

6795 One convention that might not be obvious from the formal grammar is where <newline>s are

6796 acceptable. There are several obvious placements such as terminating a statement, and a

6797 backslash can be used to escape <newline>s between any lexical tokens. In addition, <newline>s

6798 without backslashes can follow a comma, an open brace, logical AND operator ("&&"), logical

6799 OR operator ("||"), the **do** keyword, the **else** keyword, and the closing parenthesis of an **if**, **for**,

6800 or **while** statement. For example:

```
6801 { print $1,
6802         $2 }
```

### 6803 Lexical Conventions

6804 The lexical conventions for *awk* programs, with respect to the preceding grammar, shall be as

6805 follows:

- 6806 1. Except as noted, *awk* shall recognize the longest possible token or delimiter beginning at a
- 6807 given point.
- 6808 2. A comment shall consist of any characters beginning with the number sign character and
- 6809 terminated by, but excluding the next occurrence of, a <newline>. Comments shall have
- 6810 no effect, except to delimit lexical tokens.
- 6811 3. The <newline> shall be recognized as the token **NEWLINE**.
- 6812 4. A backslash character immediately followed by a <newline> shall have no effect.
- 6813 5. The token **STRING** shall represent a string constant. A string constant shall begin with the
- 6814 character ' " '. Within a string constant, a backslash character shall be considered to begin
- 6815 an escape sequence as specified in the table in the Base Definitions volume of
- 6816 IEEE Std 1003.1-200x, Chapter 5, File Format Notation ('\\', '\a', '\b', '\f', '\n',
- 6817 '\r', '\t', '\v'). In addition, the escape sequences in Table 4-2 (on page 2364) shall be
- 6818 recognized. A <newline> shall not occur within a string constant. A string constant shall be
- 6819 terminated by the first unescaped occurrence of the character ' " ' after the one that begins
- 6820 the string constant. The value of the string shall be the sequence of all unescaped
- 6821 characters and values of escape sequences between, but not including, the two delimiting
- 6822 ' " ' characters.
- 6823 6. The token **ERE** represents an extended regular expression constant. An ERE constant shall
- 6824 begin with the slash character. Within an ERE constant, a backslash character shall be
- 6825 considered to begin an escape sequence as specified in the table in the Base Definitions
- 6826 volume of IEEE Std 1003.1-200x, Chapter 5, File Format Notation. In addition, the escape
- 6827 sequences in Table 4-2 (on page 2364) shall be recognized. The application shall ensure that
- 6828 a <newline> does not occur within an ERE constant. An ERE constant shall be terminated
- 6829 by the first unescaped occurrence of the slash character after the one that begins the ERE
- 6830 constant. The extended regular expression represented by the ERE constant shall be the
- 6831 sequence of all unescaped characters and values of escape sequences between, but not
- 6832 including, the two delimiting slash characters.

- 6833 7. A <blank> shall have no effect, except to delimit lexical tokens or within **STRING** or **ERE**  
6834 tokens.
- 6835 8. The token **NUMBER** shall represent a numeric constant. Its form and numeric value shall  
6836 be equivalent to either of the tokens **floating-constant** or **integer-constant** as specified by  
6837 the ISO C standard, with the following exceptions:
- 6838 a. An integer constant cannot begin with 0x or include the hexadecimal digits 'a', 'b',  
6839 'c', 'd', 'e', 'f', 'A', 'B', 'C', 'D', 'E', or 'F'.
  - 6840 b. The value of an integer constant beginning with 0 shall be taken in decimal rather  
6841 than octal.
  - 6842 c. An integer constant cannot include a suffix ('u', 'U', 'l', or 'L').
  - 6843 d. A floating constant cannot include a suffix ('f', 'F', 'l', or 'L').
- 6844 If the value is too large or too small to be representable (see Section 1.7.2 (on page 2207)),  
6845 the behavior is undefined.
- 6846 9. A sequence of underscores, digits, and alphabetic characters from the portable character set (see the  
6847 Base Definitions volume of IEEE Std 1003.1-200x, Section 6.1, Portable Character Set),  
6848 beginning with an underscore or alphabetic, shall be considered a word.
- 6849 10. The following words are keywords that shall be recognized as individual tokens; the name  
6850 of the token is the same as the keyword:

6851	<b>BEGIN</b>	<b>delete</b>	<b>END</b>	<b>function</b>	<b>in</b>	<b>printf</b>	
6852	<b>break</b>	<b>do</b>	<b>exit</b>	<b>getline</b>	<b>next</b>	<b>return</b>	
6853	<b>continue</b>	<b>else</b>	<b>for</b>	<b>if</b>	<b>print</b>	<b>while</b>	

- 6854 11. The following words are names of built-in functions and shall be recognized as the token  
6855 **BUILTIN\_FUNC\_NAME**:

6856	<b>atan2</b>	<b>gsub</b>	<b>log</b>	<b>split</b>	<b>sub</b>	<b>toupper</b>	
6857	<b>close</b>	<b>index</b>	<b>match</b>	<b>sprintf</b>	<b>substr</b>		
6858	<b>cos</b>	<b>int</b>	<b>rand</b>	<b>sqrt</b>	<b>system</b>		
6859	<b>exp</b>	<b>length</b>	<b>sin</b>	<b>srand</b>	<b>tolower</b>		

6860 The above-listed keywords and names of built-in functions are considered reserved words.

- 6861 12. The token **NAME** shall consist of a word that is not a keyword or a name of a built-in  
6862 function and is not followed immediately (without any delimiters) by the ' ( ' character.
- 6863 13. The token **FUNC\_NAME** shall consist of a word that is not a keyword or a name of a  
6864 built-in function, followed immediately (without any delimiters) by the ' ( ' character. The  
6865 ' ( ' character shall not be included as part of the token.
- 6866 14. The following two-character sequences shall be recognized as the named tokens:

Token Name	Sequence	Token Name	Sequence
<b>ADD_ASSIGN</b>	+=	<b>NO_MATCH</b>	!~
<b>SUB_ASSIGN</b>	-=	<b>EQ</b>	==
<b>MUL_ASSIGN</b>	*=	<b>LE</b>	<=
<b>DIV_ASSIGN</b>	/=	<b>GE</b>	>=
<b>MOD_ASSIGN</b>	%=	<b>NE</b>	!=
<b>POW_ASSIGN</b>	^=	<b>INCR</b>	++
<b>OR</b>		<b>DECR</b>	--
<b>AND</b>	&&	<b>APPEND</b>	>>

6876 15. The following single characters shall be recognized as tokens whose names are the  
6877 character:

6878 <newline> { } ( ) [ ] , ; + - \* % ^ ! > < | ? : ~ \$ =

6879 There is a lexical ambiguity between the token **ERE** and the tokens `'/'` and **DIV\_ASSIGN**.  
6880 When an input sequence begins with a slash character in any syntactic context where the token  
6881 `'/'` or **DIV\_ASSIGN** could appear as the next token in a valid program, the longer of those two  
6882 tokens that can be recognized shall be recognized. In any other syntactic context where the token  
6883 **ERE** could appear as the next token in a valid program, the token **ERE** shall be recognized.

#### 6884 **EXIT STATUS**

6885 The following exit values shall be returned:

6886 0 All input files were processed successfully.

6887 >0 An error occurred.

6888 The exit status can be altered within the program by using an **exit** expression.

#### 6889 **CONSEQUENCES OF ERRORS**

6890 If any *file* operand is specified and the named file cannot be accessed, *awk* shall write a  
6891 diagnostic message to standard error and terminate without any further action.

6892 If the program specified by either the *program* operand or a *progfile* operand is not a valid *awk*  
6893 program (as specified in the EXTENDED DESCRIPTION section), the behavior is undefined.

#### 6894 **APPLICATION USAGE**

6895 The **index**, **length**, **match**, and **substr** functions should not be confused with similar functions in  
6896 the ISO C standard; the *awk* versions deal with characters, while the ISO C standard deals with  
6897 bytes.

6898 Because the concatenation operation is represented by adjacent expressions rather than an  
6899 explicit operator, it is often necessary to use parentheses to enforce the proper evaluation  
6900 precedence.

#### 6901 **EXAMPLES**

6902 The *awk* program specified in the command line is most easily specified within single-quotes (for  
6903 example, `'program'`) for applications using *sh*, because *awk* programs commonly contain  
6904 characters that are special to the shell, including double-quotes. In the cases where an *awk*  
6905 program contains single-quote characters, it is usually easiest to specify most of the program as  
6906 strings within single-quotes concatenated by the shell with quoted single-quote characters. For  
6907 example:

```
6908 awk '/\''/ { print "quote:", $0 }'
```

6909 prints all lines from the standard input containing a single-quote character, prefixed with *quote*..

6910 The following are examples of simple *awk* programs:

6911 1. Write to the standard output all input lines for which field 3 is greater than 5:

```
6912 $3 > 5
```

6913 2. Write every tenth line:

```
6914 (NR % 10) == 0
```

6915 3. Write any line with a substring matching the regular expression:

```
6916 /(G|D)(2[0-9][[:alpha:]]*)/
```

- 6917 4. Print any line with a substring containing a 'G' or 'D', followed by a sequence of digits  
6918 and characters. This example uses character classes **digit** and **alpha** to match language-  
6919 independent digit and alphabetic characters respectively:
- ```
6920 /(G|D)([[:digit:][:alpha:]]*)/
```
- 6921 5. Write any line in which the second field matches the regular expression and the fourth  
6922 field does not:
- ```
6923 $2 ~ /xyz/ && $4 !~ /xyz/
```
- 6924 6. Write any line in which the second field contains a backslash:
- ```
6925 $2 ~ /\//
```
- 6926 7. Write any line in which the second field contains a backslash. Note that backslash escapes  
6927 are interpreted twice, once in lexical processing of the string and once in processing the  
6928 regular expression:
- ```
6929 $2 ~ "\\\""
```
- 6930 8. Write the second to the last and the last field in each line. Separate the fields by a colon:
- ```
6931 {OFS=":";print $(NF-1), $NF}
```
- 6932 9. Write the line number and number of fields in each line. The three strings representing the  
6933 line number, the colon, and the number of fields are concatenated and that string is written  
6934 to standard output:
- ```
6935 {print NR ":" NF}
```
- 6936 10. Write lines longer than 72 characters:
- ```
6937 length($0) > 72
```
- 6938 11. Write first two fields in opposite order separated by the **OFS**:
- ```
6939 { print $2, $1 }
```
- 6940 12. Same, with input fields separated by comma or <space>s and <tab>s, or both:
- ```
6941 BEGIN { FS = ",[ \t]*|[ \t]+" }
6942 { print $2, $1 }
```
- 6943 13. Add up first column, print sum, and average:
- ```
6944 {s += $1 }
6945 END {print "sum is ", s, " average is", s/NR}
```
- 6946 14. Write fields in reverse order, one per line (many lines out for each line in):
- ```
6947 { for (i = NF; i > 0; --i) print $i }
```
- 6948 15. Write all lines between occurrences of the strings **start** and **stop**:
- ```
6949 /start/, /stop/
```
- 6950 16. Write all lines whose first field is different from the previous one:
- ```
6951 $1 != prev { print; prev = $1 }
```
- 6952 17. Simulate *echo*:
- ```
6953 BEGIN {
6954     for (i = 1; i < ARGV; ++i)
6955         printf("%s%s", ARGV[i], i==ARGV-1?"\n":" ")
```

```

6956     }
6957 18. Write the path prefixes contained in the PATH environment variable, one per line:
6958     BEGIN {
6959         n = split (ENVIRON["PATH"], path, ":")
6960         for (i = 1; i <= n; ++i)
6961             print path[i]
6962     }

```

6963 19. If there is a file named **input** containing page headers of the form:

```
6964     Page #
```

6965 and a file named **program** that contains:

```
6966     /Page/    { $2 = n++; }
6967             { print }

```

6968 then the command line:

```
6969     awk -f program n=5 input
```

6970 prints the file **input**, filling in page numbers starting at 5.

#### 6971 RATIONALE

6972 The ISO POSIX-2 standard description is based on the new *awk*, “*nawk*”, (see the referenced *The*  
6973 *AWK Programming Language*), which introduced a number of new features to the historical *awk*:

- 6974 1. New keywords: **delete**, **do**, **functin**, **return**
- 6975 2. New built-in functions: **atan2**, **close**, **cos**, **gsub**, **match**, **rand**, **sin**, **srand**, **sub**, **system**
- 6976 3. New predefined variables: **FNR**, **ARGC**, **ARGV**, **RSTART**, **RLENGTH**, **SUBSEP**
- 6977 4. New expression operators: **?**, **:**, **..**, **^**
- 6978 5. The **FS** variable and the third argument to **split**, now treated as extended regular  
6979 expressions.
- 6980 6. The operator precedence, changed to more closely match the C language. Two examples  
6981 of code that operate differently are:

```
6982     while ( n /= 10 > 1) ...
6983     if (!"wk" ~ /bwk/) ...

```

6984 Several features have been added based on newer implementations of *awk*:

- 6985 • Multiple instances of **-f *profilename*** are permitted
- 6986 • The new option **-v *assignment***
- 6987 • The new predefined variable **ENVIRON**
- 6988 • New built-in functions **toupper**, and **tolower**
- 6989 • More formatting capabilities are added to **printf** to match the ISO C standard

6990 The overall *awk* syntax has always been based on the C language, with a few features from the  
6991 shell command language and other sources. Because of this, it is not completely compatible with  
6992 any other language, which has caused confusion for some users. It is not the intent of the  
6993 standard developers to address such issues. IEEE Std 1003.1-200x has made a few relatively  
6994 minor changes toward making the language more compatible with the C language as specified  
6995 by the ISO C standard; most of these changes are based on similar changes in recent

6996 implementations, as described above. There remain several C-language conventions that are not  
6997 in *awk*. One of the notable ones is the comma operator, which is commonly used to specify  
6998 multiple expressions in the C language **for** statement. Also, there are various places where *awk*  
6999 is more restrictive than the C language regarding the type of expression that can be used in a given  
7000 context. These limitations are due to the different features that the *awk* language does provide.

7001 Regular expressions in *awk* have been extended somewhat from historical implementations to  
7002 make them a pure superset of extended regular expressions, as defined by IEEE Std 1003.1-200x  
7003 (see the Base Definitions volume of IEEE Std 1003.1-200x, Section 9.4, Extended Regular  
7004 Expressions). The main extensions are internationalization features and interval expressions.  
7005 Historical implementations of *awk* have long supported backslash escape sequences as an  
7006 extension to extended regular expressions, and this extension has been retained despite  
7007 inconsistency with other utilities. The number of escape sequences recognized in both extended  
7008 regular expressions and strings has varied (generally increasing with time) among  
7009 implementations. The set specified by IEEE Std 1003.1-200x includes most sequences known to  
7010 be supported by popular implementations and by the ISO C standard. One sequence that is not  
7011 supported is hexadecimal value escapes beginning with `'\x'`. This would allow values  
7012 expressed in more than 9 bits to be used within *awk* as in the ISO C standard. However, because  
7013 this syntax has a non-deterministic length, it does not permit the subsequent character to be a  
7014 hexadecimal digit. This limitation can be dealt with in the C language by the use of lexical string  
7015 concatenation. In the *awk* language, concatenation could also be a solution for strings, but not for  
7016 extended regular expressions (either lexical ERE tokens or strings used dynamically as regular  
7017 expressions). Because of this limitation, the feature has not been added to IEEE Std 1003.1-200x.

7018 When a string variable is used in a context where an extended regular expression normally  
7019 appears (where the lexical token ERE is used in the grammar) the string does not contain the  
7020 literal slashes.

7021 Some versions of *awk* allow the form:

```
7022 func name(args, ... ) { statements }
```

7023 This has been deprecated by the authors of the language, who asked that it not be included in  
7024 IEEE Std 1003.1-200x.

7025 Historical implementations of *awk* produce an error if a **next** statement is executed in a **BEGIN**  
7026 action, and cause *awk* to terminate if a **next** statement is executed in an **END** action. This  
7027 behavior has not been documented, and it was not believed that it was necessary to standardize  
7028 it.

7029 The specification of conversions between string and numeric values is much more detailed than  
7030 in the documentation of historical implementations or in the referenced *The AWK Programming*  
7031 *Language*. Although most of the behavior is designed to be intuitive, the details are necessary to  
7032 ensure compatible behavior from different implementations. This is especially important in  
7033 relational expressions since the types of the operands determine whether a string or numeric  
7034 comparison is performed. From the perspective of an application writer, it is usually sufficient to  
7035 expect intuitive behavior and to force conversions (by adding zero or concatenating a null  
7036 string) when the type of an expression does not obviously match what is needed. The intent has  
7037 been to specify historical practice in almost all cases. The one exception is that, in historical  
7038 implementations, variables and constants maintain both string and numeric values after their  
7039 original value is converted by any use. This means that referencing a variable or constant can  
7040 have unexpected side effects. For example, with historical implementations the following  
7041 program:

```
7042 {  
7043     a = "+2"
```

```

7044         b = 2
7045         if (NR % 2)
7046             c = a + b
7047         if (a == b)
7048             print "numeric comparison"
7049         else
7050             print "string comparison"
7051     }

```

7052 would perform a numeric comparison (and output numeric comparison) for each odd-  
7053 numbered line, but perform a string comparison (and output string comparison) for each even-  
7054 numbered line. IEEE Std 1003.1-200x ensures that comparisons will be numeric if necessary.  
7055 With historical implementations, the following program:

```

7056 BEGIN {
7057     OFMT = "%e"
7058     print 3.14
7059     OFMT = "%f"
7060     print 3.14
7061 }

```

7062 would output "3.140000e+00" twice, because in the second **print** statement the constant  
7063 "3.14" would have a string value from the previous conversion. IEEE Std 1003.1-200x requires  
7064 that the output of the second **print** statement be "3.140000". The behavior of historical  
7065 implementations was seen as too unintuitive and unpredictable.

7066 It was pointed out that with the rules contained in early drafts, the following script would print  
7067 nothing:

```

7068 BEGIN {
7069     y[1.5] = 1
7070     OFMT = "%e"
7071     print y[1.5]
7072 }

```

7073 Therefore, a new variable, **CONVFMT**, was introduced. The **OFMT** variable is now restricted to  
7074 affecting output conversions of numbers to strings and **CONVFMT** is used for internal  
7075 conversions, such as comparisons or array indexing. The default value is the same as that for  
7076 **OFMT**, so unless a program changes **CONVFMT** (which no historical program would do), it  
7077 will receive the historical behavior associated with internal string conversions.

7078 The POSIX *awk* lexical and syntactic conventions are specified more formally than in other  
7079 sources. Again the intent has been to specify historical practice. One convention that may not be  
7080 obvious from the formal grammar as in other verbal descriptions is where <newline>s are  
7081 acceptable. There are several obvious placements such as terminating a statement, and a  
7082 backslash can be used to escape <newline>s between any lexical tokens. In addition, <newline>s  
7083 without backslashes can follow a comma, an open brace, a logical AND operator ("&&"), a  
7084 logical OR operator ("||"), the **do** keyword, the **else** keyword, and the closing parenthesis of an  
7085 **if**, **for**, or **while** statement. For example:

```

7086 { print $1,
7087     $2 }

```

7088 The requirement that *awk* add a trailing <newline> to the program argument text is to simplify  
7089 the grammar, making it match a text file in form. There is no way for an application or test suite  
7090 to determine whether a literal <newline> is added or whether *awk* simply acts as if it did.

7091 IEEE Std 1003.1-200x requires several changes from historical implementations in order to  
 7092 support internationalization. Probably the most subtle of these is the use of the decimal-point  
 7093 character, defined by the *LC\_NUMERIC* category of the locale, in representations of floating-  
 7094 point numbers. This locale-specific character is used in recognizing numeric input, in converting  
 7095 between strings and numeric values, and in formatting output. However, regardless of locale,  
 7096 the period character (the decimal-point character of the POSIX locale) is the decimal-point  
 7097 character recognized in processing *awk* programs (including assignments in command line  
 7098 arguments). This is essentially the same convention as the one used in the ISO C standard. The  
 7099 difference is that the C language includes the *setlocale()* function, which permits an application  
 7100 to modify its locale. Because of this capability, a C application begins executing with its locale  
 7101 set to the C locale, and only executes in the environment-specified locale after an explicit call to  
 7102 *setlocale()*. However, adding such an elaborate new feature to the *awk* language was seen as  
 7103 inappropriate for IEEE Std 1003.1-200x. It is possible to execute an *awk* program explicitly in any  
 7104 desired locale by setting the environment in the shell.

7105 The undefined behavior resulting from NULs in extended regular expressions allows future  
 7106 extensions for the GNU *gawk* program to process binary data.

7107 The behavior in the case of invalid *awk* programs (including lexical, syntactic, and semantic  
 7108 errors) is undefined because it was considered overly limiting on implementations to specify. In  
 7109 most cases such errors can be expected to produce a diagnostic and a non-zero exit status.  
 7110 However, some implementations may choose to extend the language in ways that make use of  
 7111 certain invalid constructs. Other invalid constructs might be deemed worthy of a warning, but  
 7112 otherwise cause some reasonable behavior. Still other constructs may be very difficult to detect  
 7113 in some implementations. Also, different implementations might detect a given error during an  
 7114 initial parsing of the program (before reading any input files) while others might detect it when  
 7115 executing the program after reading some input. Implementors should be aware that diagnosing  
 7116 errors as early as possible and producing useful diagnostics can ease debugging of applications,  
 7117 and thus make an implementation more usable.

7118 The unspecified behavior from using multi-character **RS** values is to allow possible future  
 7119 extensions based on extended regular expressions used for record separators. Historical  
 7120 implementations take the first character of the string and ignore the others.

7121 Unspecified behavior when *split(string,array,<null>)* is used is to allow a proposed future  
 7122 extension that would split up a string into an array of individual characters.

7123 In the context of the **getline** function, equally good arguments for different precedences of the |  
 7124 and < operators can be made. Historical practice has been that:

```
7125 getline < "a" "b"
```

7126 is parsed as:

```
7127 ( getline < "a" ) "b"
```

7128 although many would argue that the intent was that the file **ab** should be read. However:

```
7129 getline < "x" + 1
```

7130 parses as:

```
7131 getline < ( "x" + 1 )
```

7132 Similar problems occur with the | version of **getline**, particularly in combination with \$. For  
 7133 example:

```
7134 $"echo hi" | getline
```

7135 (This situation is particularly problematic when used in a **print** statement, where the `|getline`  
7136 part might be a redirection of the **print**.)

7137 Since in most cases such constructs are not (or at least should not) be used (because they have a  
7138 natural ambiguity for which there is no conventional parsing), the meaning of these constructs  
7139 has been made explicitly unspecified. (The effect is that a conforming application that runs into  
7140 the problem must parenthesize to resolve the ambiguity.) There appeared to be few if any actual  
7141 uses of such constructs.

7142 Grammars can be written that would cause an error under these circumstances. Where  
7143 backwards compatibility is not a large consideration, implementors may wish to use such  
7144 grammars.

7145 Some historical implementations have allowed some built-in functions to be called without an  
7146 argument list, the result being a default argument list chosen in some “reasonable” way. Use of  
7147 **length** as a synonym for **length(\$0)** is the only one of these forms that is thought to be widely  
7148 known or widely used; this particular form is documented in various places (for example, most  
7149 historical *awk* reference pages, although not in the referenced *The AWK Programming Language*)  
7150 as legitimate practice. With this exception, default argument lists have always been  
7151 undocumented and vaguely defined, and it is not at all clear how (or if) they should be  
7152 generalized to user-defined functions. They add no useful functionality and preclude possible  
7153 future extensions that might need to name functions without calling them. Not standardizing  
7154 them seems the simplest course. The standard developers considered that **length** merited special  
7155 treatment, however, since it has been documented in the past and sees possibly substantial use  
7156 in historical programs. Accordingly, this usage has been made legitimate, but Issue 5 removed  
7157 the obsolescent marking for XSI-conforming implementations and many otherwise conforming  
7158 applications depend on this feature.

7159 In **sub** and **gsub**, if *repl* is a string literal (the lexical token **STRING**), then two consecutive  
7160 backslash characters should be used in the string to ensure a single backslash will precede the  
7161 ampersand when the resultant string is passed to the function. (For example, to specify one  
7162 literal ampersand in the replacement string, use **gsub(ERE, "\\&")**.)

7163 Historically the only special character in the *repl* argument of **sub** and **gsub** string functions was  
7164 the ampersand ('&') character and preceding it with the backslash character was used to turn  
7165 off its special meaning.

7166 The description in the ISO POSIX-2:1993 standard introduced behavior such that the backslash  
7167 character was another special character and it was unspecified whether there were any other  
7168 special characters. This description introduced several portability problems, some of which are  
7169 described below, and so it has been replaced with the more historical description. Some of the  
7170 problems include:

- 7171 • Historically, to create the replacement string, a script could use **gsub(ERE, "\\&")**, but with  
7172 the ISO POSIX-2:1993 standard wording, it was necessary to use **gsub(ERE, "\\&")**.  
7173 Backslash characters are doubled here because all string literals are subject to lexical analysis,  
7174 which would reduce each pair of backslash characters to a single backslash before being  
7175 passed to **gsub**.
- 7176 • Since it was unspecified what the special characters were, for portable scripts to guarantee  
7177 that characters are printed literally, each character had to be preceded with a backslash. (For  
7178 example, a portable script had to use **gsub(ERE, "\\h\\i")** to produce a replacement string  
7179 of "hi".)

7180 The description for comparisons in the ISO POSIX-2:1993 standard did not properly describe  
7181 historical practice because of the way numeric strings are compared as numbers. The current  
7182 rules cause the following code:

```

7183     if (0 == "000")
7184         print "strange, but true"
7185     else
7186         print "not true"

```

7187 to do a numeric comparison, causing the **if** to succeed. It should be intuitively obvious that this  
 7188 is incorrect behavior, and indeed, no historical implementation of *awk* actually behaves this way.

7189 To fix this problem, the definition of *numeric string* was enhanced to include only those values  
 7190 obtained from specific circumstances (mostly external sources) where it is not possible to  
 7191 determine unambiguously whether the value is intended to be a string or a numeric.

7192 Variables that are assigned to a numeric string shall also be treated as a numeric string. (For  
 7193 example, the notion of a numeric string can be propagated across assignments.) In comparisons,  
 7194 all variables having the uninitialized value are to be treated as a numeric operand evaluating to  
 7195 the numeric value zero.

7196 Uninitialized variables include all types of variables including scalars, array elements, and fields.  
 7197 The definition of an uninitialized value in **Variables and Special Variables** (on page 2361) is  
 7198 necessary to describe the value placed on uninitialized variables and on fields that are valid (for  
 7199 example, < **\$NF**) but have no characters in them and to describe how these variables are to be  
 7200 used in comparisons. A valid field, such as **\$1**, that has no characters in it can be obtained by  
 7201 from an input line of "\t\t" when **FS**=' \t '. Historically, the comparison (**\$1**<10) was done  
 7202 numerically after evaluating **\$1** to the value zero.

7203 The phrase "... also shall have the numeric value of the numeric string" was removed from  
 7204 several sections of the ISO POSIX-2:1993 standard because it specifies an unnecessary  
 7205 implementation detail. It is not necessary for IEEE Std 1003.1-200x to specify that these objects  
 7206 be assigned two different values. It is only necessary to specify that these objects may evaluate  
 7207 to two different values depending on context.

7208 The description of numeric string processing is based on the behavior of the *atof()* function in  
 7209 the ISO C standard. While it is not a requirement for an implementation to use this function,  
 7210 many historical implementations of *awk* do. In the ISO C standard, floating-point constants use a  
 7211 period as a decimal point character for the language itself, independent of the current locale, but  
 7212 the *atof()* function and the associated *strtod()* function use the decimal point character of the  
 7213 current locale when converting strings to numeric values. Similarly in *awk*, floating-point  
 7214 constants in an *awk* script use a period independent of the locale, but input strings use the  
 7215 decimal point character of the locale.

#### 7216 **FUTURE DIRECTIONS**

7217 None.

#### 7218 **SEE ALSO**

7219 *grep*, *lex*, *sed*, the System Interfaces volume of IEEE Std 1003.1-200x, *atof()*, *setlocale()*, *strtod()*

#### 7220 **CHANGE HISTORY**

7221 First released in Issue 2.

#### 7222 **Issue 5**

7223 FUTURE DIRECTIONS section added.

#### 7224 **Issue 6**

7225 The *awk* utility is aligned with the IEEE P1003.2b draft standard.

7226 The normative text is reworded to avoid use of the term "must" for application requirements. |

7227  
7228  
7229

IEEE PASC Interpretation 1003.2 #211 is applied, adding the sentence “An occurrence of two consecutive backslashes shall be interpreted as just a single literal backslash character.” into the description of the **sub** string function.

7230 **NAME**7231           **basename** — return non-directory portion of a pathname7232 **SYNOPSIS**7233           **basename** *string* [*suffix*]7234 **DESCRIPTION**

7235           The *string* operand shall be treated as a pathname, as defined in the Base Definitions volume of  
 7236           IEEE Std 1003.1-200x, Section 3.266, Pathname. The string *string* shall be converted to the  
 7237           filename corresponding to the last pathname component in *string* and then the suffix string  
 7238           *suffix*, if present, shall be removed. This shall be done by performing actions equivalent to the  
 7239           following steps in order:

- 7240           1. If *string* is a null string, it is unspecified whether the resulting string is ' . ' or a null string.  
 7241           In either case, skip steps 2 through 6.
- 7242           2. If *string* is "/", it is implementation-defined whether steps 3 to 6 are skipped or  
 7243           processed.
- 7244           3. If *string* consists entirely of slash characters, *string* shall be set to a single slash character. In  
 7245           this case, skip steps 4 to 6.
- 7246           4. If there are any trailing slash characters in *string*, they shall be removed.
- 7247           5. If there are any slash characters remaining in *string*, the prefix of *string* up to and including  
 7248           the last slash character in *string* shall be removed.
- 7249           6. If the *suffix* operand is present, is not identical to the characters remaining in *string*, and is  
 7250           identical to a suffix of the characters remaining in *string*, the suffix *suffix* shall be removed  
 7251           from *string*. Otherwise, *string* is not modified by this step. It shall not be considered an  
 7252           error if *suffix* is not found in *string*.

7253           The resulting string shall be written to standard output.

7254 **OPTIONS**

7255           None.

7256 **OPERANDS**

7257           The following operands shall be supported:

7258           *string*           A string.7259           *suffix*           A string.7260 **STDIN**

7261           Not used.

7262 **INPUT FILES**

7263           None.

7264 **ENVIRONMENT VARIABLES**7265           The following environment variables shall affect the execution of *basename*:

7266           *LANG*           Provide a default value for the internationalization variables that are unset or null.  
 7267           (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
 7268           Internationalization Variables for the precedence of internationalization variables  
 7269           used to determine the values of locale categories.)

7270           *LC\_ALL*          If set to a non-empty string value, override the values of all the other  
 7271           internationalization variables.

7272            *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 7273 characters (for example, single-byte as opposed to multi-byte characters in  
 7274 arguments).

7275            *LC\_MESSAGES*  
 7276                      Determine the locale that should be used to affect the format and contents of  
 7277 diagnostic messages written to standard error.

7278 XSI        *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

7279 **ASYNCHRONOUS EVENTS**  
 7280            Default.

7281 **STDOUT**  
 7282            The *basename* utility shall write a line to the standard output in the following format:  
 7283            "%s\n", <resulting string>

7284 **STDERR**  
 7285            The standard error shall be used only for diagnostic messages.

7286 **OUTPUT FILES**  
 7287            None.

7288 **EXTENDED DESCRIPTION**  
 7289            None.

7290 **EXIT STATUS**  
 7291            The following exit values shall be returned:  
 7292            0 Successful completion.  
 7293            >0 An error occurred.

7294 **CONSEQUENCES OF ERRORS**  
 7295            Default.

7296 **APPLICATION USAGE**  
 7297            The definition of *pathname* specifies implementation-defined behavior for pathnames starting  
 7298 with two slash characters. Therefore, applications shall not arbitrarily add slashes to the  
 7299 beginning of a pathname unless they can ensure that there are more or less than two or are  
 7300 prepared to deal with the implementation-defined consequences.

7301 **EXAMPLES**  
 7302            If the string *string* is a valid pathname:  
 7303            \$(basename "*string*")  
 7304            produces a filename that could be used to open the file named by *string* in the directory returned  
 7305 by:  
 7306            \$(dirname "*string*")  
 7307            If the string *string* is not a valid pathname, the same algorithm is used, but the result need not be  
 7308 a valid filename. The *basename* utility is not expected to make any judgements about the validity  
 7309 of *string* as a pathname; it just follows the specified algorithm to produce a result string.

7310            The following shell script compiles */usr/src/cmd/cat.c* and moves the output to a file named *cat*  
 7311 in the current directory when invoked with the argument */usr/src/cmd/cat* or with the argument  
 7312 */usr/src/cmd/cat.c*:

```
7313      c99 $(dirname "$1")/$(basename "$1" .c).c
7314      mv a.out $(basename "$1" .c)
```

**7315 RATIONALE**

7316 The behaviors of *basename* and *dirname* have been coordinated so that when *string* is a valid  
7317 pathname:

```
7318      $(basename "string")
```

7319 would be a valid filename for the file in the directory:

```
7320      $(dirname "string")
```

7321 This would not work for the early proposal versions of these utilities due to the way it specified  
7322 handling of trailing slashes.

7323 Since the definition of *pathname* specifies implementation-defined behavior for pathnames  
7324 starting with two slash characters, this volume of IEEE Std 1003.1-200x specifies similar  
7325 implementation-defined behavior for the *basename* and *dirname* utilities.

**7326 FUTURE DIRECTIONS**

7327 None.

**7328 SEE ALSO**

7329 *dirname*, Section 2.5 (on page 2235)

**7330 CHANGE HISTORY**

7331 First released in Issue 2.

**7332 Issue 6**

7333 IEEE PASC Interpretation 1003.2 #164 is applied.

7334 The normative text is reworded to avoid use of the term “must” for application requirements.

7335 **NAME**

7336 batch — schedule commands to be executed in a batch queue

7337 **SYNOPSIS**7338 UP *batch*

7339

7340 **DESCRIPTION**7341 The *batch* utility shall read commands from standard input and schedule them for execution in a  
7342 batch queue. It shall be the equivalent of the command:

7343 at -q b -m now

7344 where queue *b* is a special *at* queue, specifically for batch jobs. Batch jobs shall be submitted to  
7345 the batch queue with no time constraints and shall be run by the system using algorithms, based  
7346 on unspecified factors, that may vary with each invocation of *batch*.7347 XSI Users shall be permitted to use *batch* if their name appears in the file */usr/lib/cron/at.allow*. If  
7348 that file does not exist, the file */usr/lib/cron/at.deny* shall be checked to determine whether the  
7349 user shall be denied access to *batch*. If neither file exists, only a process with the appropriate  
7350 privileges shall be allowed to submit a job. If only *at.deny* exists and is empty, global usage shall  
7351 be permitted. The *at.allow* and *at.deny* files shall consist of one user name per line.7352 **OPTIONS**

7353 None.

7354 **OPERANDS**

7355 None.

7356 **STDIN**7357 The standard input shall be a text file consisting of commands acceptable to the shell command  
7358 language described in Chapter 2 (on page 2231).7359 **INPUT FILES**7360 XSI The text files */usr/lib/cron/at.allow* and */usr/lib/cron/at.deny* shall contain zero or more user  
7361 names, one per line, of users who are, respectively, authorized or denied access to the *at* and  
7362 *batch* utilities.7363 **ENVIRONMENT VARIABLES**7364 The following environment variables shall affect the execution of *batch*:7365 *LANG* Provide a default value for the internationalization variables that are unset or null.  
7366 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
7367 Internationalization Variables for the precedence of internationalization variables  
7368 used to determine the values of locale categories.)7369 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
7370 internationalization variables.7371 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
7372 characters (for example, single-byte as opposed to multi-byte characters in  
7373 arguments and input files).7374 *LC\_MESSAGES*7375 Determine the locale that should be used to affect the format and contents of  
7376 diagnostic messages written to standard error and informative messages written to  
7377 standard output.7378 *LC\_TIME* Determine the format and contents for date and time strings written by *batch*.

- 7379 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 7380 **SHELL** Determine the name of a command interpreter to be used to invoke the at-job. If  
7381 the variable is unset or null, *sh* shall be used. If it is set to a value other than a name  
7382 for *sh*, the implementation shall do one of the following: use that shell; use *sh*; use  
7383 the login shell from the user database; any of the preceding accompanied by a  
7384 warning diagnostic about which was chosen.
- 7385 **TZ** Determine the timezone. The job shall be submitted for execution at the time  
7386 specified by *timespec* or *-t time* relative to the timezone specified by the *TZ*  
7387 variable. If *timespec* specifies a timezone, it overrides *TZ*. If *timespec* does not  
7388 specify a timezone and *TZ* is unset or null, an unspecified default timezone shall  
7389 be used.
- 7390 **ASYNCHRONOUS EVENTS**
- 7391 Default.
- 7392 **STDOUT**
- 7393 When standard input is a terminal, prompts of unspecified format for each line of the user input  
7394 described in the STDIN section may be written to standard output.
- 7395 **STDERR**
- 7396 The following shall be written to standard error when a job has been successfully submitted:
- 7397 "job %s at %s\n", *at\_job\_id*, <*date*>
- 7398 where *date* shall be equivalent in format to the output of:
- 7399 date +"%a %b %e %T %Y"
- 7400 The date and time written shall be adjusted so that they appear in the timezone of the user (as  
7401 determined by the *TZ* variable).
- 7402 Neither this, nor warning messages concerning the selection of the command interpreter, are  
7403 considered a diagnostic that changes the exit status.
- 7404 Diagnostic messages, if any, shall be written to standard error.
- 7405 **OUTPUT FILES**
- 7406 None.
- 7407 **EXTENDED DESCRIPTION**
- 7408 None.
- 7409 **EXIT STATUS**
- 7410 The following exit values shall be returned:
- 7411 0 Successful completion.
- 7412 >0 An error occurred.
- 7413 **CONSEQUENCES OF ERRORS**
- 7414 The job shall not be scheduled.

7415 **APPLICATION USAGE**

7416 It may be useful to redirect standard output within the specified commands.

7417 **EXAMPLES**

7418 1. This sequence can be used at a terminal:

```
7419     batch
7420     sort < file >outfile
7421     EOT
```

7422 2. This sequence, which demonstrates redirecting standard error to a pipe, is useful in a  
7423 command procedure (the sequence of output redirection specifications is significant):

```
7424     batch <<! diff file1 file2 2>&1 >outfile | mailx mygroup !
```

7425 **RATIONALE**

7426 Early proposals described *batch* in a manner totally separated from *at*, even though the historical  
7427 model treated it almost as a synonym for *at -qb*. A number of features were added to list and  
7428 control batch work separately from those in *at*. Upon further reflection, it was decided that the  
7429 benefit of this did not merit the change to the historical interface.

7430 The *-m* option was included on the equivalent *at* command because it is historical practice to  
7431 mail results to the submitter, even if all job-produced output is redirected. As explained in the  
7432 RATIONALE for *at*, the **now** keyword submits the job for immediate execution (after scheduling  
7433 delays), despite some historical systems where *at now* would have been considered an error.

7434 **FUTURE DIRECTIONS**

7435 None.

7436 **SEE ALSO**

7437 *at*

7438 **CHANGE HISTORY**

7439 First released in Issue 2.

7440 **Issue 6**

7441 This utility is now marked as part of the User Portability Utilities option.

7442 The NAME is changed to align with the IEEE P1003.2b draft standard.

7443 The normative text is reworded to avoid use of the term “must” for application requirements.

7444 **NAME**

7445       bc — arbitrary-precision arithmetic language

7446 **SYNOPSIS**7447       bc [-l] [*file* ...]7448 **DESCRIPTION**

7449       The *bc* utility shall implement an arbitrary precision calculator. It shall take input from any files  
 7450       given, then read from the standard input. If the standard input and standard output to *bc* are  
 7451       attached to a terminal, the invocation of *bc* shall be considered to be *interactive*, causing  
 7452       behavioral constraints described in the following sections.

7453 **OPTIONS**

7454       The *bc* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2,  
 7455       Utility Syntax Guidelines.

7456       The following option shall be supported:

7457       -l           (The letter ell.) Define the math functions and initialize *scale* to 20, instead of the  
 7458       default zero; see the EXTENDED DESCRIPTION section.

7459 **OPERANDS**

7460       The following operand shall be supported:

7461       *file*        A pathname of a text file containing *bc* program statements. After all *files* have  
 7462       been read, *bc* shall read the standard input.

7463 **STDIN**

7464       See the INPUT FILES section.

7465 **INPUT FILES**

7466       Input files shall be text files containing a sequence of comments, statements, and function  
 7467       definitions that shall be executed as they are read.

7468 **ENVIRONMENT VARIABLES**7469       The following environment variables shall affect the execution of *bc*:

7470       *LANG*        Provide a default value for the internationalization variables that are unset or null.  
 7471       (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
 7472       Internationalization Variables for the precedence of internationalization variables  
 7473       used to determine the values of locale categories.)

7474       *LC\_ALL*      If set to a non-empty string value, override the values of all the other  
 7475       internationalization variables.

7476       *LC\_CTYPE*   Determine the locale for the interpretation of sequences of bytes of text data as  
 7477       characters (for example, single-byte as opposed to multi-byte characters in  
 7478       arguments and input files).

7479       *LC\_MESSAGES*

7480       Determine the locale that should be used to affect the format and contents of  
 7481       diagnostic messages written to standard error.

7482       XSI        *NLS\_PATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

7483 **ASYNCHRONOUS EVENTS**

7484       Default.

7485 **STDOUT**

7486 The output of the *bc* utility shall be controlled by the program read, and consist of zero or more  
 7487 lines containing the value of all executed expressions without assignments. The radix and  
 7488 precision of the output shall be controlled by the values of the **obase** and **scale** variables; see the  
 7489 EXTENDED DESCRIPTION section.

7490 **STDERR**

7491 The standard error shall be used only for diagnostic messages. |

7492 **OUTPUT FILES**

7493 None.

7494 **EXTENDED DESCRIPTION**7495 **Grammar**

7496 The grammar in this section and the lexical conventions in the following section shall together  
 7497 describe the syntax for *bc* programs. The general conventions for this style of grammar are  
 7498 described in Section 1.10 (on page 2220). A valid program can be represented as the non-  
 7499 terminal symbol **program** in the grammar. This formal syntax shall take precedence over the text  
 7500 syntax description.

```

7501 %token    EOF NEWLINE STRING LETTER NUMBER
7502 %token    MUL_OP
7503 /*      '*' , '/' , '%'          */
7504 %token    ASSIGN_OP
7505 /*      '=' , '+=' , '-=' , '*=' , '/=' , '%=' , '^=' */
7506 %token    REL_OP
7507 /*      '==' , '<=' , '>=' , '!=' , '<' , '>'          */
7508 %token    INCR_DECR
7509 /*      '++' , '--'              */
7510 %token    Define    Break    Quit    Length
7511 /*      'define' , 'break' , 'quit' , 'length'      */
7512 %token    Return    For    If    While    Sqrt
7513 /*      'return' , 'for' , 'if' , 'while' , 'sqrt'  */
7514 %token    Scale    Ibase    Obase    Auto
7515 /*      'scale' , 'ibase' , 'obase' , 'auto'      */
7516 %start    program
7517 %%
7518 program    : EOF
7519             | input_item program
7520             ;
7521 input_item : semicolon_list NEWLINE
7522             | function
7523             ;
7524 semicolon_list : /* empty */
7525                 | statement
7526                 | semicolon_list ';' statement
7527                 | semicolon_list ';'

```

```

7528                                     ;
7529     statement_list                   : /* empty */
7530                                     | statement
7531                                     | statement_list NEWLINE
7532                                     | statement_list NEWLINE statement
7533                                     | statement_list ';'
7534                                     | statement_list ';' statement
7535                                     ;
7536     statement                         : expression
7537                                     | STRING
7538                                     | Break
7539                                     | Quit
7540                                     | Return
7541                                     | Return '(' return_expression ')'
7542                                     | For '(' expression ';'
7543                                         relational_expression ';'
7544                                         expression ')' statement
7545                                     | If '(' relational_expression ')' statement
7546                                     | While '(' relational_expression ')' statement
7547                                     | '{' statement_list '}'
7548                                     ;
7549     function                           : Define LETTER '(' opt_parameter_list ')'
7550                                     | '{' NEWLINE opt_auto_define_list
7551                                         statement_list '}'
7552                                     ;
7553     opt_parameter_list                 : /* empty */
7554                                     | parameter_list
7555                                     ;
7556     parameter_list                   : LETTER
7557                                     | define_list ',' LETTER
7558                                     ;
7559     opt_auto_define_list              : /* empty */
7560                                     | Auto define_list NEWLINE
7561                                     | Auto define_list ';'
7562                                     ;
7563     define_list                       : LETTER
7564                                     | LETTER '[' ']'
7565                                     | define_list ',' LETTER
7566                                     | define_list ',' LETTER '[' ']'
7567                                     ;
7568     opt_argument_list                 : /* empty */
7569                                     | argument_list
7570                                     ;
7571     argument_list                     : expression
7572                                     | LETTER '[' ']' ',' argument_list"
7573                                     ;

```

```

7574 relational_expression : expression
7575                        | expression REL_OP expression
7576                        ;
7577 return_expression     : /* empty */
7578                        | expression
7579                        ;
7580 expression            : named_expression
7581                        | NUMBER
7582                        | '(' expression ')'
7583                        | LETTER '(' opt_argument_list ')'
7584                        | '-' expression
7585                        | expression '+' expression
7586                        | expression '-' expression
7587                        | expression MUL_OP expression
7588                        | expression '^' expression
7589                        | INCR_DECR named_expression
7590                        | named_expression INCR_DECR
7591                        | named_expression ASSIGN_OP expression
7592                        | Length '(' expression ')'
7593                        | Sqrt '(' expression ')'
7594                        | Scale '(' expression ')'
7595                        ;
7596 named_expression     : LETTER
7597                        | LETTER '[' expression ']'
7598                        | Scale
7599                        | Ibase
7600                        | Obase
7601                        ;

```

## 7602 Lexical Conventions in bc

7603 The lexical conventions for *bc* programs, with respect to the preceding grammar, shall be as  
7604 follows:

- 7605 1. Except as noted, *bc* shall recognize the longest possible token or delimiter beginning at a  
7606 given point.
- 7607 2. A comment shall consist of any characters beginning with the two adjacent characters  
7608 `"/**"` and terminated by the next occurrence of the two adjacent characters `"*/"`.  
7609 Comments shall have no effect except to delimit lexical tokens.
- 7610 3. The `<newline>` shall be recognized as the token **NEWLINE**.
- 7611 4. The token **STRING** shall represent a string constant; it shall consist of any characters  
7612 beginning with the double-quote character (`'"`) and terminated by another occurrence of  
7613 the double-quote character. The value of the string is the sequence of all characters  
7614 between, but not including, the two double-quote characters. All characters shall be taken  
7615 literally from the input, and there is no way to specify a string containing a double-quote  
7616 character. The length of the value of each string shall be limited to `{BC_STRING_MAX}`  
7617 bytes.
- 7618 5. A `<blank>` shall have no effect except as an ordinary character if it appears within a  
7619 **STRING** token, or to delimit a lexical token other than **STRING**.

- 7620 6. The combination of a backslash character immediately followed by a <newline> shall have  
7621 no effect other than to delimit lexical tokens with the following exceptions:
- 7622 • It shall be interpreted as the character sequence "\<newline>" in **STRING** tokens.
  - 7623 • It shall be ignored as part of a multi-line **NUMBER** token.
- 7624 7. The token **NUMBER** shall represent a numeric constant. It shall be recognized by the  
7625 following grammar:
- ```
7626 NUMBER : integer
7627         | '.' integer
7628         | integer '.'
7629         | integer '.' integer
7630         ;

7631 integer : digit
7632         | integer digit
7633         ;

7634 digit   : 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7
7635         | 8 | 9 | A | B | C | D | E | F
7636         ;
```
- 7637 8. The value of a **NUMBER** token shall be interpreted as a numeral in the base specified by  
7638 the value of the internal register **ibase** (described below). Each of the **digit** characters shall  
7639 have the value from 0 to 15 in the order listed here, and the period character shall represent  
7640 the radix point. The behavior is undefined if digits greater than or equal to the value of  
7641 **ibase** appear in the token. However, note the exception for single-digit values being  
7642 assigned to **ibase** and **obase** themselves, in **Operations in bc** (on page 2400).
- 7643 9. The following keywords shall be recognized as tokens:
- |      |               |              |               |               |              |  |
|------|---------------|--------------|---------------|---------------|--------------|--|
| 7644 | <b>auto</b>   | <b>ibase</b> | <b>length</b> | <b>return</b> | <b>while</b> |  |
| 7645 | <b>break</b>  | <b>if</b>    | <b>obase</b>  | <b>scale</b>  |              |  |
| 7646 | <b>define</b> | <b>for</b>   | <b>quit</b>   | <b>sqrt</b>   |              |  |
- 7647 10. Any of the following characters occurring anywhere except within a keyword shall be  
7648 recognized as the token **LETTER**:
- ```
7649 a b c d e f g h i j k l m n o p q r s t u v w x y z
```
- 7650 11. The following single-character and two-character sequences shall be recognized as the  
7651 token **ASSIGN\_OP**:
- ```
7652 = += -= *= /= %= ^=
```
- 7653 12. If an '=' character, as the beginning of a token, is followed by a '-' character with no  
7654 intervening delimiter, the behavior is undefined.
- 7655 13. The following single-characters shall be recognized as the token **MUL\_OP**:
- ```
7656 * / %
```
- 7657 14. The following single-character and two-character sequences shall be recognized as the  
7658 token **REL\_OP**:
- ```
7659 == <= >= != < >
```
- 7660 15. The following two-character sequences shall be recognized as the token **INCR\_DECR**:

- 7661            ++    --
- 7662            16. The following single characters shall be recognized as tokens whose names are the  
7663            character:
- 7664            <newline> ( ) , + - ; [ ] ^ { }
- 7665            17. The token **EOF** is returned when the end of input is reached.

### 7666            **Operations in bc**

7667            There are three kinds of identifiers: ordinary identifiers, array identifiers, and function  
7668            identifiers. All three types consist of single lowercase letters. Array identifiers shall be followed  
7669            by square brackets ("[]"). An array subscript is required except in an argument or auto list.  
7670            Arrays are singly dimensioned and can contain up to {BC\_DIM\_MAX} elements. Indexing shall  
7671            begin at zero so an array is indexed from 0 to {BC\_DIM\_MAX}-1. Subscripts shall be truncated  
7672            to integers. The application shall ensure that function identifiers are followed by parentheses,  
7673            possibly enclosing arguments. The three types of identifiers do not conflict.

7674            The following table summarizes the rules for precedence and associativity of all operators.  
7675            Operators on the same line shall have the same precedence; rows are in order of decreasing  
7676            precedence.

7677            **Table 4-3 Operators in bc**

| Operator                  | Associativity |
|---------------------------|---------------|
| ++, --                    | N/A           |
| unary -                   | N/A           |
| ^                         | Right to left |
| *, /, %                   | Left to right |
| +, binary -               | Left to right |
| =, +=, -=, *=, /=, %=, ^= | Right to left |
| ==, <=, >=, !=, <, >      | None          |

7686            Each expression or named expression has a *scale*, which is the number of decimal digits that  
7687            shall be maintained as the fractional portion of the expression.

7688            *Named expressions* are places where values are stored. Named expressions shall be valid on the  
7689            left side of an assignment. The value of a named expression shall be the value stored in the place  
7690            named. Simple identifiers and array elements are named expressions; they have an initial value  
7691            of zero and an initial scale of zero.

7692            The internal registers **scale**, **ibase**, and **obase** are all named expressions. The scale of an  
7693            expression consisting of the name of one of these registers shall be zero; values assigned to any  
7694            of these registers are truncated to integers. The **scale** register shall contain a global value used in  
7695            computing the scale of expressions (as described below). The value of the register **scale** is  
7696            limited to  $0 \leq \text{scale} \leq \{\text{BC\_SCALE\_MAX}\}$  and shall have a default value of zero. The **ibase** and  
7697            **obase** registers are the input and output number radix, respectively. The value of **ibase** shall be  
7698            limited to:

7699             $2 \leq \text{ibase} \leq 16$

7700            The value of **obase** shall be limited to:

7701             $2 \leq \text{obase} \leq \{\text{BC\_BASE\_MAX}\}$

7702            When either **ibase** or **obase** is assigned a single **digit** value from the list in **Lexical Conventions**  
7703            **in bc** (on page 2398), the value shall be assumed in hexadecimal. (For example, **ibase=A** sets to

7704 base ten, regardless of the current **ibase** value.) Otherwise, the behavior is undefined when  
 7705 digits greater than or equal to the value of **ibase** appear in the input. Both **ibase** and **obase** shall  
 7706 have initial values of 10.

7707 Internal computations shall be conducted as if in decimal, regardless of the input and output  
 7708 bases, to the specified number of decimal digits. When an exact result is not achieved, (for  
 7709 example, **scale=0**; 3.2/1) the result shall be truncated.

7710 For all values of **obase** specified by this volume of IEEE Std 1003.1-200x, *bc* shall output numeric  
 7711 values by performing each of the following steps in order:

- 7712 1. If the value is less than zero, a hyphen ('-') character shall be output.
- 7713 2. One of the following is output, depending on the numerical value:
  - 7714 • If the absolute value of the numerical value is greater than or equal to one, the integer  
 7715 portion of the value shall be output as a series of digits appropriate to **obase** (as  
 7716 described below) most significant digit first. The most significant non-zero digit shall  
 7717 be output next, followed by each successively less significant digit.
  - 7718 • If the absolute value of the numerical value is less than one but greater than zero and  
 7719 the scale of the numerical value is greater than zero, it is unspecified whether the  
 7720 character 0 is output.
  - 7721 • If the numerical value is zero, the character 0 shall be output.
- 7722 3. If the scale of the value is greater than zero and the numeric value is not zero, a period  
 7723 character shall be output, followed by a series of digits appropriate to **obase** (as described  
 7724 below) representing the most significant portion of the fractional part of the value. If *s*  
 7725 represents the scale of the value being output, the number of digits output shall be *s* if  
 7726 **obase** is 10, less than or equal to *s* if **obase** is greater than 10, or greater than or equal to *s* if  
 7727 **obase** is less than 10. For **obase** values other than 10, this should be the number of digits  
 7728 needed to represent a precision of  $10^s$ .

7729 For **obase** values from 2 to 16, valid digits are the first **obase** of the single characters:

7730 0 1 2 3 4 5 6 7 8 9 A B C D E F

7731 which represent the values zero to 15, inclusive, respectively.

7732 For bases greater than 16, each digit shall be written as a separate multi-digit decimal number.  
 7733 Each digit except the most significant fractional digit shall be preceded by a single <space>. For  
 7734 bases from 17 to 100, *bc* shall write two-digit decimal numbers; for bases from 101 to 1 000,  
 7735 three-digit decimal strings, and so on. For example, the decimal number 1 024 in base 25 would  
 7736 be written as:

7737 Δ01Δ15Δ24

7738 in base 125, as:

7739 Δ008Δ024

7740 Very large numbers shall be split across lines with 70 characters per line in the POSIX locale;  
 7741 other locales may split at different character boundaries. Lines that are continued shall end with  
 7742 a backslash ('\').

7743 A function call shall consist of a function name followed by parentheses containing a comma-  
 7744 separated list of expressions, which are the function arguments. A whole array passed as an  
 7745 argument shall be specified by the array name followed by empty square brackets. All function  
 7746 arguments shall be passed by value. As a result, changes made to the formal parameters shall  
 7747 have no effect on the actual arguments. If the function terminates by executing a **return**

7748 statement, the value of the function shall be the value of the expression in the parentheses of the  
7749 **return** statement or shall be zero if no expression is provided or if there is no **return** statement.

7750 The result of **sqrt**(*expression*) shall be the square root of the expression. The result shall be  
7751 truncated in the least significant decimal place. The scale of the result shall be the scale of the  
7752 expression or the value of **scale**, whichever is larger.

7753 The result of **length**(*expression*) shall be the total number of significant decimal digits in the  
7754 expression. The scale of the result shall be zero.

7755 The result of **scale**(*expression*) shall be the scale of the expression. The scale of the result shall be  
7756 zero.

7757 A numeric constant shall be an expression. The scale shall be the number of digits that follow the  
7758 radix point in the input representing the constant, or zero if no radix point appears.

7759 The sequence ( *expression* ) shall be an expression with the same value and scale as *expression*.  
7760 The parentheses can be used to alter the normal precedence.

7761 The semantics of the unary and binary operators are as follows:

7762 *-expression*  
7763 The result shall be the negative of the *expression*. The scale of the result shall be the scale of  
7764 *expression*.

7765 The unary increment and decrement operators shall not modify the scale of the named  
7766 expression upon which they operate. The scale of the result shall be the scale of that named  
7767 expression.

7768 *++named-expression*  
7769 The named expression shall be incremented by one. The result shall be the value of the  
7770 named expression after incrementing.

7771 *--named-expression*  
7772 The named expression shall be decremented by one. The result shall be the value of the  
7773 named expression after decrementing.

7774 *named-expression++*  
7775 The named expression shall be incremented by one. The result shall be the value of the  
7776 named expression before incrementing.

7777 *named-expression--*  
7778 The named expression shall be decremented by one. The result shall be the value of the  
7779 named expression before decrementing.

7780 The exponentiation operator, circumflex ( '^ ' ), shall bind right to left.

7781 *expression^expression*  
7782 The result shall be the first *expression* raised to the power of the second *expression*. If the  
7783 second expression is not an integer, the behavior is undefined. If *a* is the scale of the left  
7784 expression and *b* is the absolute value of the right expression, the scale of the result shall be:  
7785 if  $b \geq 0$   $\min(a * b, \max(\text{scale}, a))$  if  $b < 0$  *scale*

7786 The multiplicative operators ( ' \* ' , ' / ' , ' % ' ) shall bind left to right.

7787 *expression\*expression*  
7788 The result shall be the product of the two expressions. If *a* and *b* are the scales of the two  
7789 expressions, then the scale of the result shall be:

7790  $\min(a+b, \max(\text{scale}, a, b))$

7791 *expression*/*expression*

7792 The result shall be the quotient of the two expressions. The scale of the result shall be the  
7793 value of **scale**.

7794 *expression*%*expression*

7795 For expressions *a* and *b*, *a*%*b* shall be evaluated equivalent to the steps:

7796 1. Compute *a*/*b* to current scale.

7797 2. Use the result to compute:

7798  $a - (a / b) * b$

7799 to scale:

7800  $\max(\text{scale} + \text{scale}(b), \text{scale}(a))$

7801 The scale of the result shall be:

7802  $\max(\text{scale} + \text{scale}(b), \text{scale}(a))$

7803 When **scale** is zero, the '*%*' operator is the mathematical remainder operator.

7804 The additive operators ('+', '-') shall bind left to right.

7805 *expression*+*expression*

7806 The result shall be the sum of the two expressions. The scale of the result shall be the  
7807 maximum of the scales of the expressions.

7808 *expression*-*expression*

7809 The result shall be the difference of the two expressions. The scale of the result shall be the  
7810 maximum of the scales of the expressions.

7811 The assignment operators ('=', '+=', '-=', '\*=', '/=', '%=', '^=') shall bind right to left.

7812 *named-expression*=*expression*

7813 This expression shall result in assigning the value of the expression on the right to the  
7814 named expression on the left. The scale of both the named expression and the result shall be  
7815 the scale of *expression*.

7816 The compound assignment forms:

7817 *named-expression* <operator>= *expression*

7818 shall be equivalent to:

7819 *named-expression*=*named-expression* <operator> *expression*

7820 except that the *named-expression* shall be evaluated only once.

7821 Unlike all other operators, the relational operators ('<', '>', '<=', '>=', '==', '!=') shall be  
7822 only valid as the object of an **if**, **while**, or inside a **for** statement.

7823 *expression1*<*expression2*

7824 The relation shall be true if the value of *expression1* is strictly less than the value of  
7825 *expression2*.

7826 *expression1*>*expression2*

7827 The relation shall be true if the value of *expression1* is strictly greater than the value of  
7828 *expression2*.

7829 *expression1* <= *expression2*  
 7830 The relation shall be true if the value of *expression1* is less than or equal to the value of  
 7831 *expression2*.

7832 *expression1* >= *expression2*  
 7833 The relation shall be true if the value of *expression1* is greater than or equal to the value of  
 7834 *expression2*.

7835 *expression1* = *expression2*  
 7836 The relation shall be true if the values of *expression1* and *expression2* are equal.

7837 *expression1* != *expression2*  
 7838 The relation shall be true if the values of *expression1* and *expression2* are unequal.

7839 There are only two storage classes in *bc*, global and automatic (local). Only identifiers that are  
 7840 local to a function need be declared with the **auto** command. The arguments to a function shall  
 7841 be local to the function. All other identifiers are assumed to be global and available to all  
 7842 functions. All identifiers, global and local, have initial values of zero. Identifiers declared as auto  
 7843 shall be allocated on entry to the function and released on returning from the function. They  
 7844 therefore do not retain values between function calls. Auto arrays shall be specified by the array  
 7845 name followed by empty square brackets. On entry to a function, the old values of the names  
 7846 that appear as parameters and as automatic variables shall be pushed onto a stack. Until the  
 7847 function returns, reference to these names shall refer only to the new values.

7848 References to any of these names from other functions that are called from this function also  
 7849 refer to the new value until one of those functions uses the same name for a local variable.

7850 When a statement is an expression, unless the main operator is an assignment, execution of the  
 7851 statement shall write the value of the expression followed by a <newline>.

7852 When a statement is a string, execution of the statement shall write the value of the string.

7853 Statements separated by semicolons or <newline>s shall be executed sequentially. In an  
 7854 interactive invocation of *bc*, each time a <newline> is read that satisfies the grammatical  
 7855 production:

7856 `input_item : semicolon_list NEWLINE`

7857 the sequential list of statements making up the **semicolon\_list** shall be executed immediately  
 7858 and any output produced by that execution shall be written without any delay due to buffering.

7859 In an **if** statement (**if**(*relation*) *statement*), the *statement* shall be executed if the relation is true.

7860 The **while** statement (**while**(*relation*) *statement*) implements a loop in which the *relation* is tested;  
 7861 each time the *relation* is true, the *statement* shall be executed and the *relation* retested. When the  
 7862 *relation* is false, execution shall resume after *statement*.

7863 A **for** statement (**for**(*expression*; *relation*; *expression*) *statement*) shall be the same as:

7864 `first-expression`  
 7865 `while (relation) {`  
 7866 `statement`  
 7867 `last-expression`  
 7868 `}`

7869 The application shall ensure that all three expressions are present.

7870 The **break** statement shall cause termination of a **for** or **while** statement.

7871 The **auto** statement (**auto** *identifier* [*identifier*] ...) shall cause the values of the identifiers to be  
 7872 pushed down. The identifiers can be ordinary identifiers or array identifiers. Array identifiers

7873 shall be specified by following the array name by empty square brackets. The application shall  
7874 ensure that the **auto** statement is the first statement in a function definition.

7875 A **define** statement:

```
7876 define LETTER ( opt_parameter_list ) {
7877     opt_auto_define_list
7878     statement_list
7879 }
```

7880 defines a function named **LETTER**. If a function named **LETTER** was previously defined, the  
7881 **define** statement shall replace the previous definition. The expression:

```
7882 LETTER ( opt_argument_list )
```

7883 shall invoke the function named **LETTER**. The behavior is undefined if the number of  
7884 arguments in the invocation does not match the number of parameters in the definition.  
7885 Functions shall be defined before they are invoked. A function shall be considered to be defined  
7886 within its own body, so recursive calls are valid. The values of numeric constants within a  
7887 function shall be interpreted in the base specified by the value of the **ibase** register when the  
7888 function is invoked.

7889 The **return** statements (**return** and **return(expression)**) shall cause termination of a function,  
7890 popping of its auto variables, and specification of the result of the function. The first form shall  
7891 be equivalent to **return(0)**. The value and scale of the result returned by the function shall be the  
7892 value and scale of the expression returned.

7893 The **quit** statement (**quit**) shall stop execution of a *bc* program at the point where the statement  
7894 occurs in the input, even if it occurs in a function definition, or in an **if**, **for**, or **while** statement.

7895 The following functions shall be defined when the **-l** option is specified:

```
7896 s( expression )
7897     Sine of argument in radians.
```

```
7898 c( expression )
7899     Cosine of argument in radians.
```

```
7900 a( expression )
7901     Arctangent of argument.
```

```
7902 l( expression )
7903     Natural logarithm of argument.
```

```
7904 e( expression )
7905     Exponential function of argument.
```

```
7906 j( expression, expression )
7907     Bessel function of integer order.
```

7908 The scale of the result returned by these functions shall be the value of the **scale** register at the  
7909 time the function is invoked. The value of the **scale** register after these functions have completed  
7910 their execution shall be the same value it had upon invocation. The behavior is undefined if any  
7911 of these functions is invoked with an argument outside the domain of the mathematical  
7912 function.

## 7913 EXIT STATUS

7914 The following exit values shall be returned:

7915 0 All input files were processed successfully.

7916 *unspecified* An error occurred.

### 7917 CONSEQUENCES OF ERRORS

7918 If any *file* operand is specified and the named file cannot be accessed, *bc* shall write a diagnostic message to standard error and terminate without any further action.

7920 In an interactive invocation of *bc*, the utility should print an error message and recover following any error in the input. In a non-interactive invocation of *bc*, invalid input causes undefined behavior.

### 7923 APPLICATION USAGE

7924 Automatic variables in *bc* do not work in exactly the same way as in either C or PL/1.

7925 For historical reasons, the exit status from *bc* cannot be relied upon to indicate that an error has occurred. Returning zero after an error is possible. Therefore, *bc* should be used primarily by interactive users (who can react to error messages) or by application programs that can somehow validate the answers returned as not including error messages.

7929 The *bc* utility always uses the period ( `'.'` ) character to represent a radix point, regardless of any decimal-point character specified as part of the current locale. In languages like C or *awk*, the period character is used in program source, so it can be portable and unambiguous, while the locale-specific character is used in input and output. Because there is no distinction between source and input in *bc*, this arrangement would not be possible. Using the locale-specific character in *bc*'s input would introduce ambiguities into the language; consider the following example in a locale with a comma as the decimal-point character:

```
7936 define f(a,b) {
7937     ...
7938 }
7939 ...
7940 f(1,2,3)
```

7941 Because of such ambiguities, the period character is used in input. Having input follow different conventions from output would be confusing in either pipeline usage or interactive usage, so the period is also used in output.

### 7944 EXAMPLES

7945 In the shell, the following assigns an approximation of the first ten digits of ' $\pi$ ' to the variable *x*:

```
7947 x=$(printf "%s\n" 'scale = 10; 104348/33215' | bc)
```

7948 The following *bc* program prints the same approximation of ' $\pi$ ', with a label, to standard output:

```
7950 scale = 10
7951 "pi equals "
7952 104348 / 33215
```

7953 The following defines a function to compute an approximate value of the exponential function (note that such a function is predefined if the `-l` option is specified):

```
7955 scale = 20
7956 define e(x){
7957     auto a, b, c, i, s
7958     a = 1
7959     b = 1
7960     s = 1
```

```

7961         for (i = 1; i == 1; i++){
7962             a = a*x
7963             b = b*i
7964             c = a/b
7965             if (c == 0) {
7966                 return(s)
7967             }
7968             s = s+c
7969         }
7970     }

```

7971 The following prints approximate values of the exponential function of the first ten integers:

```

7972     for (i = 1; i <= 10; ++i) {
7973         e(i)
7974     }

```

#### 7975 RATIONALE

7976 The *bc* utility is implemented historically as a front-end processor for *dc*; *dc* was not selected to  
 7977 be part of this volume of IEEE Std 1003.1-200x because *bc* was thought to have a more intuitive  
 7978 programmatic interface. Current implementations that implement *bc* using *dc* are expected to be  
 7979 compliant.

7980 The exit status for error conditions has been left unspecified for several reasons:

- 7981 • The *bc* utility is used in both interactive and non-interactive situations. Different exit codes  
 7982 may be appropriate for the two uses.
- 7983 • It is unclear when a non-zero exit should be given; divide-by-zero, undefined functions, and  
 7984 syntax errors are all possibilities.
- 7985 • It is not clear what utility the exit status has.
- 7986 • In the 4.3 BSD, System V, and Ninth Edition implementations, *bc* works in conjunction with  
 7987 *dc*. The *dc* utility is the parent, *bc* is the child. This was done to cleanly terminate *bc* if *dc*  
 7988 aborted.

7989 The decision to have *bc* exit upon encountering an inaccessible input file is based on the belief  
 7990 that *bc file1 file2* is used most often when at least *file1* contains data/function  
 7991 declarations/initializations. Having *bc* continue with prerequisite files missing is probably not  
 7992 useful. There is no implication in the CONSEQUENCES OF ERRORS section that *bc* must check  
 7993 all its files for accessibility before opening any of them.

7994 There was considerable debate on the appropriateness of the language accepted by *bc*. Several  
 7995 reviewers preferred to see either a pure subset of the C language or some changes to make the  
 7996 language more compatible with C. While the *bc* language has some obvious similarities to C, it  
 7997 has never claimed to be compatible with any version of C. An interpreter for a subset of C might  
 7998 be a very worthwhile utility, and it could potentially make *bc* obsolete. However, no such utility  
 7999 is known in historical practice, and it was not within the scope of this volume of  
 8000 IEEE Std 1003.1-200x to define such a language and utility. If and when they are defined, it may  
 8001 be appropriate to include them in a future version of this volume of IEEE Std 1003.1-200x. This  
 8002 left the following alternatives:

- 8003 1. Exclude any calculator language from this volume of IEEE Std 1003.1-200x.

8004 The consensus of the standard developers was that a simple programmatic calculator  
 8005 language is very useful for both applications and interactive users. The only arguments for  
 8006 excluding any calculator were that it would become obsolete if and when a C-compatible

8007 one emerged, or that the absence would encourage the development of such a C-  
 8008 compatible one. These arguments did not sufficiently address the needs of current  
 8009 application writers.

8010 2. Standardize the historical *dc*, possibly with minor modifications.

8011 The consensus of the standard developers was that *dc* is a fundamentally less usable  
 8012 language and that that would be far too severe a penalty for avoiding the issue of being  
 8013 similar to but incompatible with C.

8014 3. Standardize the historical *bc*, possibly with minor modifications.

8015 This was the approach taken. Most of the proponents of changing the language would not  
 8016 have been satisfied until most or all of the incompatibilities with C were resolved. Since  
 8017 most of the changes considered most desirable would break historical applications and  
 8018 require significant modification to historical implementations, almost no modifications  
 8019 were made. The one significant modification that was made was the replacement of the  
 8020 historical *bc* assignment operators "*=+*", and so on, with the more modern "*+=*", and so  
 8021 on. The older versions are considered to be fundamentally flawed because of the lexical  
 8022 ambiguity in uses like *a=-1*.

8023 In order to permit implementations to deal with backwards compatibility as they see fit,  
 8024 the behavior of this one ambiguous construct was made undefined. (At least three  
 8025 implementations have been known to support this change already, so the degree of change  
 8026 involved should not be great.)

8027 The '*%*' operator is the mathematical remainder operator when **scale** is zero. The behavior of  
 8028 this operator for other values of **scale** is from historical implementations of *bc*, and has been  
 8029 maintained for the sake of historical applications despite its non-intuitive nature.

8030 Historical implementations permit setting **ibase** and **obase** to a broader range of values. This  
 8031 includes values less than 2, which were not seen as sufficiently useful to standardize. These  
 8032 implementations do not interpret input properly for values of **ibase** that are greater than 16. This  
 8033 is because numeric constants are recognized syntactically, rather than lexically, as described in  
 8034 this volume of IEEE Std 1003.1-200x. They are built from lexical tokens of single hexadecimal  
 8035 digits and periods. Since <blank>s between tokens are not visible at the syntactic level, it is not  
 8036 possible to recognize the multi-digit "digits" used in the higher bases properly. The ability to  
 8037 recognize input in these bases was not considered useful enough to require modifying these  
 8038 implementations. Note that the recognition of numeric constants at the syntactic level is not a  
 8039 problem with conformance to this volume of IEEE Std 1003.1-200x, as it does not impact the  
 8040 behavior of conforming applications (and correct *bc* programs). Historical implementations also  
 8041 accept input with all of the digits '0'-'9' and 'A'-'F' regardless of the value of **ibase**; since  
 8042 digits with value greater than or equal to **ibase** are not really appropriate, the behavior when  
 8043 they appear is undefined, except for the common case of:

```
8044 ibase=8;
8045     /* Process in octal base. */
8046     ...
8047 ibase=A
8048     /* Restore decimal base. */
```

8049 In some historical implementations, if the expression to be written is an uninitialized array  
 8050 element, a leading <space> and/or up to four leading 0 characters may be output before the  
 8051 character zero. This behavior is considered a bug; it is unlikely that any currently conforming  
 8052 application relies on:

- 8053 `echo 'b[3]' | bc`
- 8054 returning 0000 rather than 0.
- 8055 Exact calculation of the number of fractional digits to output for a given value in a base other  
8056 than 10 can be computationally expensive. Historical implementations use a faster  
8057 approximation, and this is permitted. Note that the requirements apply only to values of **obase**  
8058 that this volume of IEEE Std 1003.1-200x requires implementations to support (in particular, not  
8059 to 1, 0, or negative bases, if an implementation supports them as an extension).
- 8060 Historical implementations of *bc* did not allow array parameters to be passed as the last  
8061 parameter to a function. New implementations are encouraged to remove this restriction even  
8062 though it is not required by the grammar.
- 8063 **FUTURE DIRECTIONS**
- 8064 None.
- 8065 **SEE ALSO**
- 8066 *awk*
- 8067 **CHANGE HISTORY**
- 8068 First released in Issue 4.
- 8069 **Issue 5**
- 8070 FUTURE DIRECTIONS section added.
- 8071 **Issue 6**
- 8072 Updated to align with the IEEE P1003.2b draft standard, which included resolution of several  
8073 interpretations of the ISO POSIX-2: 1993 standard.
- 8074 The normative text is reworded to avoid use of the term “must” for application requirements.

8075 **NAME**  
8076       bg — run jobs in the background

8077 **SYNOPSIS**  
8078 UP       bg [*job\_id* ...]  
8079

8080 **DESCRIPTION**  
8081       If job control is enabled (see the description of *set -m*), the *bg* utility shall resume suspended jobs  
8082       from the current environment (see Section 2.12 (on page 2263)) by running them as background  
8083       jobs. If the job specified by *job\_id* is already a running background job, the *bg* utility shall have no  
8084       effect and shall exit successfully.

8085       Using *bg* to place a job into the background shall cause its process ID to become “known in the  
8086       current shell execution environment”, as if it had been started as an asynchronous list; see  
8087       Section 2.9.3.1 (on page 2252).

8088 **OPTIONS**  
8089       None.

8090 **OPERANDS**  
8091       The following operand shall be supported:

8092       *job\_id*       Specify the job to be resumed as a background job. If no *job\_id* operand is given,  
8093       the most recently suspended job shall be used. The format of *job\_id* is described in  
8094       the Base Definitions volume of IEEE Std 1003.1-200x, Section 3.203, Job Control Job  
8095       ID.

8096 **STDIN**  
8097       Not used.

8098 **INPUT FILES**  
8099       None.

8100 **ENVIRONMENT VARIABLES**  
8101       The following environment variables shall affect the execution of *bg*:

8102       *LANG*       Provide a default value for the internationalization variables that are unset or null.  
8103       (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
8104       Internationalization Variables for the precedence of internationalization variables  
8105       used to determine the values of locale categories.)

8106       *LC\_ALL*       If set to a non-empty string value, override the values of all the other  
8107       internationalization variables.

8108       *LC\_CTYPE*     Determine the locale for the interpretation of sequences of bytes of text data as  
8109       characters (for example, single-byte as opposed to multi-byte characters in  
8110       arguments).

8111       *LC\_MESSAGES*  
8112       Determine the locale that should be used to affect the format and contents of  
8113       diagnostic messages written to standard error.

8114 XSI       *NLSPATH*     Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

8115 **ASYNCHRONOUS EVENTS**  
8116       Default.

**8117 STDOUT**

8118 The output of *bg* shall consist of a line in the format:

8119 "[%d] %s\n", <*job-number*>, <*command*>

8120 where the fields are as follows:

8121 <*job-number*> A number that can be used to identify the job to the *wait*, *fg*, and *kill* utilities. Using  
8122 these utilities, the job can be identified by prefixing the job number with '% '.

8123 <*command*> The associated command that was given to the shell.

**8124 STDERR**

8125 The standard error shall be used only for diagnostic messages.

**8126 OUTPUT FILES**

8127 None.

**8128 EXTENDED DESCRIPTION**

8129 None.

**8130 EXIT STATUS**

8131 The following exit values shall be returned:

8132 0 Successful completion.

8133 >0 An error occurred.

**8134 CONSEQUENCES OF ERRORS**

8135 If job control is disabled, the *bg* utility shall exit with an error and no job shall be placed in the  
8136 background.

**8137 APPLICATION USAGE**

8138 A job is generally suspended by typing the SUSP character (<control>-Z on most systems); see  
8139 the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface. At  
8140 that point, *bg* can put the job into the background. This is most effective when the job is  
8141 expecting no terminal input and its output has been redirected to non-terminal files. A  
8142 background job can be forced to stop when it has terminal output by issuing the command:

8143 `stty tostop`

8144 A background job can be stopped with the command:

8145 `kill -s stop job ID`

8146 The *bg* utility does not work as expected when it is operating in its own utility execution  
8147 environment because that environment has no suspended jobs. In the following examples:

8148 ... | xargs bg

8149 (bg)

8150 each *bg* operates in a different environment and does not share its parent shell's understanding  
8151 of jobs. For this reason, *bg* is generally implemented as a shell regular built-in.

**8152 EXAMPLES**

8153 None.

**8154 RATIONALE**

8155 The extensions to the shell specified in this volume of IEEE Std 1003.1-200x have mostly been  
8156 based on features provided by the KornShell. The job control features provided by *bg*, *fg*, and *jobs*  
8157 are also based on the KornShell. The standard developers examined the characteristics of the C  
8158 shell versions of these utilities and found that differences exist. Despite widespread use of the C

8159 shell, the KornShell versions were selected for this volume of IEEE Std 1003.1-200x to maintain a  
8160 degree of uniformity with the rest of the KornShell features selected (such as the very popular  
8161 command line editing features).

8162 The *bg* utility is expected to wrap its output if the output exceeds the number of display  
8163 columns.

8164 **FUTURE DIRECTIONS**

8165 None.

8166 **SEE ALSO**

8167 *fg, kill, jobs, wait*

8168 **CHANGE HISTORY**

8169 First released in Issue 4.

8170 **Issue 6**

8171 This utility is now marked as part of the User Portability Utilities option.

8172 The JC margin marker on the SYNOPSIS is removed since support for Job Control is mandatory  
8173 in this issue. This is a FIPS requirement.

8174 **NAME**8175 `c99` — compile standard C programs8176 **SYNOPSIS**

```
8177 CD c99 [-c][-D name[=value]]...[-E][-g][-I directory] ... [-L directory]
8178 ... [-o outfile][-Ooptlevel][-s][-U name]... operand ...
8179
```

8180 **DESCRIPTION**

8181 The `c99` utility is an interface to the standard C compilation system; it shall accept source code  
 8182 conforming to the ISO C standard. The system conceptually consists of a compiler and link  
 8183 editor. The files referenced by *operands* shall be compiled and linked to produce an executable  
 8184 file. (It is unspecified whether the linking occurs entirely within the operation of `c99`; some  
 8185 implementations may produce objects that are not fully resolved until the file is executed.)

8186 If the `-c` option is specified, for all pathname operands of the form *file.c*, the files:

8187 `$(basename pathname .c).o`

8188 shall be created as the result of successful compilation. If the `-c` option is not specified, it is  
 8189 unspecified whether such `.o` files are created or deleted for the *file.c* operands.

8190 If there are no options that prevent link editing (such as `-c` or `-E`), and all operands compile and  
 8191 link without error, the resulting executable file shall be written according to the `-o outfile` option  
 8192 (if present) or to the file `a.out`.

8193 The executable file shall be created as specified in Section 1.7.1.4 (on page 2204), except that the  
 8194 file permission bits shall be set to:

8195 `S_IRWXO | S_IRWXG | S_IRWXU`

8196 and the bits specified by the *umask* of the process shall be cleared.

8197 **OPTIONS**

8198 The `c99` utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section  
 8199 12.2, Utility Syntax Guidelines, except that:

- 8200 • The `-I library` operands have the format of options, but their position within a list of  
 8201 operands affects the order in which libraries are searched.
- 8202 • The order of specifying the `-I` and `-L` options is significant.
- 8203 • Conforming applications shall specify each option separately; that is, grouping option letters  
 8204 (for example, `-cO`) need not be recognized by all implementations.

8205 The following options shall be supported:

8206 `-c` Suppress the link-edit phase of the compilation, and do not remove any object files  
 8207 that are produced.

8208 `-g` Produce symbolic information in the object or executable files; the nature of this  
 8209 information is unspecified, and may be modified by implementation-defined  
 8210 interactions with other options.

8211 `-s` Produce object or executable files, or both, from which symbolic and other  
 8212 information not required for proper execution using the *exec* family defined in the  
 8213 System Interfaces volume of IEEE Std 1003.1-200x, has been removed (stripped). If  
 8214 both `-g` and `-s` options are present, the action taken is unspecified.

8215 `-o outfile` Use the pathname *outfile*, instead of the default `a.out`, for the executable file  
 8216 produced. If the `-o` option is present with `-c` or `-E`, the result is unspecified.

- 8217        **-D** *name*[=*value*]  
8218                Define *name* as if by a C-language **#define** directive. If no *=value* is given, a value of  
8219                1 shall be used. The **-D** option has lower precedence than the **-U** option. That is, if  
8220                *name* is used in both a **-U** and a **-D** option, *name* shall be undefined regardless of  
8221                the order of the options. Additional implementation-defined *names* may be  
8222                provided by the compiler. Implementations shall support at least 2 048 bytes of **-D**  
8223                definitions and 256 *names*.
- 8224        **-E**                Copy C-language source files to standard output, expanding all preprocessor  
8225                directives; no compilation shall be performed. If any operand is not a text file, the  
8226                effects are unspecified.
- 8227        **-I** *directory*    Change the algorithm for searching for headers whose names are not absolute  
8228                pathnames to look in the *directory* named by the *directory* pathname before  
8229                looking in the usual places. Thus, headers whose names are enclosed in double-  
8230                quotes (" ") shall be searched for first in the *directory* of the file with the **#include**  
8231                line, then in *directories* named in **-I** options, and last in the usual places. For  
8232                headers whose names are enclosed in angle brackets ("**<**""), the header shall be  
8233                searched for only in *directories* named in **-I** options and then in the usual places.  
8234                *Directories* named in **-I** options shall be searched in the order specified.  
8235                Implementations shall support at least ten instances of this option in a single *c99*  
8236                command invocation.
- 8237        **-L** *directory*    Change the algorithm of searching for the libraries named in the **-I** objects to look  
8238                in the *directory* named by the *directory* pathname before looking in the usual  
8239                places. *Directories* named in **-L** options shall be searched in the order specified.  
8240                Implementations shall support at least ten instances of this option in a single *c99*  
8241                command invocation. If a *directory* specified by a **-L** option contains files named  
8242                **libc.a**, **libm.a**, **libl.a**, or **liby.a**, the results are unspecified.
- 8243        **-O** *optlevel*    Specify the level of code optimization. If the *optlevel* option-argument is the digit  
8244                '0', all special code optimizations shall be disabled. If it is the digit '1', the  
8245                nature of the optimization is unspecified. If the **-O** option is omitted, the nature of  
8246                the system's default optimization is unspecified. It is unspecified whether code  
8247                generated in the presence of the **-O 0** option is the same as that generated when  
8248                **-O** is omitted. Other *optlevel* values may be supported.
- 8249        **-U** *name*        Remove any initial definition of *name*.
- 8250        Multiple instances of the **-D**, **-I**, **-U**, and **-L** options can be specified.

## 8251 OPERANDS

- 8252        An *operand* is either in the form of a pathname or the form **-I** *library*. The application shall  
8253        ensure that at least one operand of the pathname form is specified. The following operands shall  
8254        be supported:
- 8255        *file.c*            A C-language source file to be compiled and optionally linked. The application  
8256                shall ensure that the operand is of this form if the **-c** option is used.
- 8257        *file.a*            A library of object files typically produced by the *ar* utility, and passed directly to  
8258                the link editor. Implementations may recognize implementation-defined suffixes  
8259                other than **.a** as denoting object file libraries.
- 8260        *file.o*            An object file produced by *c99 -c* and passed directly to the link editor.  
8261                Implementations may recognize implementation-defined suffixes other than **.o** as  
8262                denoting object files.

- 8263 The processing of other files is implementation-defined.
- 8264 *-l library* (The letter ell.) Search the library named:
- 8265 *liblibrary.a*
- 8266 A library shall be searched when its name is encountered, so the placement of a *-l*
- 8267 operand is significant. Several standard libraries can be specified in this manner, as
- 8268 described in the EXTENDED DESCRIPTION section. Implementations may
- 8269 recognize implementation-defined suffixes other than *.a* as denoting libraries.
- 8270 **STDIN**
- 8271 Not used.
- 8272 **INPUT FILES**
- 8273 The input file shall be one of the following: a text file containing a C-language source program,
- 8274 an object file in the format produced by *c99 -c*, or a library of object files, in the format produced
- 8275 by archiving zero or more object files, using *ar*. Implementations may supply additional utilities
- 8276 that produce files in these formats. Additional input file formats are implementation-defined.
- 8277 **ENVIRONMENT VARIABLES**
- 8278 The following environment variables shall affect the execution of *c99*:
- 8279 *LANG* Provide a default value for the internationalization variables that are unset or null.
- 8280 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
- 8281 Internationalization Variables for the precedence of internationalization variables
- 8282 used to determine the values of locale categories.)
- 8283 *LC\_ALL* If set to a non-empty string value, override the values of all the other
- 8284 internationalization variables.
- 8285 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
- 8286 characters (for example, single-byte as opposed to multi-byte characters in
- 8287 arguments and input files).
- 8288 *LC\_MESSAGES*
- 8289 Determine the locale that should be used to affect the format and contents of
- 8290 diagnostic messages written to standard error.
- 8291 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 8292 *TMPDIR* Provide a pathname that should override the default directory for temporary files,
- 8293 XSI if any. On XSI-conforming systems, provide a pathname that shall override the
- 8294 default directory for temporary files, if any.
- 8295 **ASYNCHRONOUS EVENTS**
- 8296 Default.
- 8297 **STDOUT**
- 8298 If more than one *file* operand ending in *.c* (or possibly other unspecified suffixes) is given, for
- 8299 each such file:
- 8300 "%s:\n", *<file>*
- 8301 may be written. These messages, if written, shall precede the processing of each input file; they
- 8302 shall not be written to the standard output if they are written to the standard error, as described
- 8303 in the STDERR section.
- 8304 If the *-E* option is specified, the standard output shall be a text file that represents the results of
- 8305 the preprocessing stage of the language; it may contain extra information appropriate for
- 8306 subsequent compilation passes.

8307 **STDERR**

8308 The standard error shall be used only for diagnostic messages. If more than one *file* operand  
 8309 ending in *.c* (or possibly other unspecified suffixes) is given, for each such file:

8310 "%s:\n", <*file*>

8311 may be written to allow identification of the diagnostic and warning messages with the  
 8312 appropriate input file. These messages, if written, shall precede the processing of each input file;  
 8313 they shall not be written to the standard error if they are written to the standard output, as  
 8314 described in the STDOUT section.

8315 This utility may produce warning messages about certain conditions that do not warrant  
 8316 returning an error (non-zero) exit value.

8317 **OUTPUT FILES**

8318 Object files or executable files or both are produced in unspecified formats.

8319 **EXTENDED DESCRIPTION**8320 **Standard Libraries**

8321 The *c99* utility shall recognize the following **-I** operands for standard libraries:

8322 **-I c** This operand shall make visible all functions referenced in the System Interfaces  
 8323 volume of IEEE Std 1003.1-200x, with the possible exception of those functions  
 8324 listed as residing in <**aio.h**>, <**arpa/inet.h**>, <**math.h**>, <**mqueue.h**>, <**netdb.h**>,  
 8325 <**netinet/in.h**>, <**pthread.h**>, <**sched.h**>, <**semaphore.h**>, <**spawn.h**>,  
 8326 <**sys/socket.h**>, *pthread\_kill()*, and *pthread\_sigmask()* in <**signal.h**>, functions  
 8327 marked as extensions other than as part of the MF or MPR extensions in  
 8328 <**sys/mman.h**>, functions marked as ADV in <**fcntl.h**>, and functions marked as  
 8329 CS, CPT, and TMR in <**time.h**>. This operand shall not be required to be present to  
 8330 cause a search of this library.

8331 **-I l** This operand shall make visible all functions required by the C-language output of  
 8332 *lex* that are not made available through the **-I c** operand.

8333 **-I pthread** This operand shall make visible all functions referenced in <**pthread.h**> and  
 8334 *pthread\_kill()* and *pthread\_sigmask()* referenced in <**signal.h**>. An implementation  
 8335 may search this library in the absence of this operand.

8336 **-I m** This operand shall make visible all functions referenced in <**math.h**>. An  
 8337 implementation may search this library in the absence of this operand.

8338 **-I rt** This operand shall make visible all functions referenced in <**aio.h**>, <**mqueue.h**>,  
 8339 <**sched.h**>, <**semaphore.h**>, and <**spawn.h**>, functions marked as extensions other  
 8340 than as part of the MF or MPR extensions in <**sys/mman.h**>, functions marked as  
 8341 ADV in <**fcntl.h**>, and functions marked as CS, CPT, and TMR in <**time.h**>. An  
 8342 implementation may search this library in the absence of this operand.

8343 **-I trace** This operand shall make visible all functions referenced in <**trace.h**>. An  
 8344 implementation may search this library in the absence of this operand.

8345 **-I xnet** This operand makes visible all functions referenced in <**arpa/inet.h**>, <**netdb.h**>,  
 8346 <**netinet/in.h**>, and <**sys/socket.h**>. An implementation may search this library in  
 8347 the absence of this operand.

8348 **-I y** This operand shall make visible all functions required by the C-language output of  
 8349 *yacc* that are not made available through the **-I c** operand.

8350 In the absence of options that inhibit invocation of the link editor, such as `-c` or `-E`, the *c99* utility  
 8351 shall cause the equivalent of a `-l c` operand to be passed to the link editor as the last `-l` operand,  
 8352 causing it to be searched after all other object files and libraries are loaded.

8353 It is unspecified whether the libraries `libc.a`, `libm.a`, `librt.a`, `libpthread.a`, `libl.a`, `liby.a`, or `libxnet`  
 8354 exist as regular files. The implementation may accept as `-l` operands names of objects that do  
 8355 not exist as regular files.

### 8356 External Symbols

8357 The C compiler and link editor shall support the significance of external symbols up to a length  
 8358 of at least 31 bytes; the action taken upon encountering symbols exceeding the implementation-  
 8359 defined maximum symbol length is unspecified.

8360 The compiler and link editor shall support a minimum of 511 external symbols per source or  
 8361 object file, and a minimum of 4095 external symbols in total. A diagnostic message shall be  
 8362 written to the standard output if the implementation-defined limit is exceeded; other actions are  
 8363 unspecified.

### 8364 Programming Environments

8365 All implementations shall support one of the following programming environments as a default.  
 8366 Implementations may support more than one of the following programming environments.  
 8367 Applications can use *sysconf()* or *getconf* to determine which programming environments are  
 8368 supported.

8369 **Table 4-4** Programming Environments: Type Sizes

| 8370 | Programming Environment             | Bits in | Bits in | Bits in | Bits in |
|------|-------------------------------------|---------|---------|---------|---------|
| 8371 | <i>getconf</i> Name                 | int     | long    | pointer | off_t   |
| 8372 | <code>_POSIX_V6_ILP32_OFF32</code>  | 32      | 32      | 32      | 32      |
| 8373 | <code>_POSIX_V6_ILP32_OFFBIG</code> | 32      | 32      | 32      | ≥64     |
| 8374 | <code>_POSIX_V6_LP64_OFF64</code>   | 32      | 64      | 64      | 64      |
| 8375 | <code>_POSIX_V6_LP64_OFFBIG</code>  | ≥32     | ≥64     | ≥64     | ≥64     |

8376 All implementations shall support one or more environments where the widths of the following  
 8377 types are no greater than the width of type `long`:

8378 `blksize_t`, `cc_t`, `mode_t`, `nfds_t`, `pid_t`, `ptrdiff_t`, `size_t`, `speed_t`, `ssize_t`, `suseconds_t`,  
 8379 `tcflag_t`, `useconds_t`, `wchar_t`, `wint_t`

8380 The executable files created when these environments are selected shall be in a proper format for  
 8381 execution by the *exec* family of functions. Each environment may be one of the ones in Table 4-4,  
 8382 or it may be another environment. The names for the environments that meet this requirement  
 8383 shall be output by a *getconf* command using the `_POSIX_V6_WIDTH_RESTRICTED_ENVS`  
 8384 argument. If more than one environment meets the requirement, the names of all such  
 8385 environments shall be output on separate lines. Any of these names can then be used in a  
 8386 subsequent *getconf* command to obtain the flags specific to that environment with the following  
 8387 suffixes added as appropriate:

8388 `_CFLAGS` To get the C compiler flags.

8389 `_LDFLAGS` To get the linker/loader flags.

8390 `_LIBS` To get the libraries.

8391 This requirement may be removed in a future version of IEEE Std 1003.1.

8392 When this utility processes a file containing a function called *main()*, it shall be defined with a  
 8393 return type equivalent to **int**. Using return from *main()* shall be equivalent (other than with  
 8394 respect to language scope issues) to calling *exit()* with the returned value. Reaching the end of  
 8395 *main()* shall be equivalent to calling *exit(0)*. The implementation shall not declare a prototype  
 8396 for this function.

8397 Implementations provide configuration strings for C compiler flags, linker/loader flags, and  
 8398 libraries for each supported environment. When an application needs to use a specific  
 8399 programming environment rather than the implementation default programming environment  
 8400 while compiling, the application shall first verify that the implementation supports the desired  
 8401 environment. If the desired programming environment is supported, the application shall then  
 8402 invoke *c99* with the appropriate C compiler flags as the first options for the compile, the  
 8403 appropriate linker/loader flags after any other options but before any operands, and the  
 8404 appropriate libraries at the end of the operands.

8405 Conforming applications shall not attempt to link together object files compiled for different  
 8406 programming models. Applications shall also be aware that binary data placed in shared  
 8407 memory or in files might not be recognized by applications built for other programming models.

8408 **Table 4-5** Programming Environments: *c99* and *cc* Arguments

| 8409<br>8410         | <b>Programming Environment</b><br><i>getconf</i> Name | <b>Use</b>                                           | <b><i>c99</i> and <i>cc</i> Arguments</b><br><i>getconf</i> Name                                                                   |
|----------------------|-------------------------------------------------------|------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------|
| 8411<br>8412<br>8413 | <code>_POSIX_V6_ILP32_OFF32</code>                    | C Compiler Flags<br>Linker/Loader Flags<br>Libraries | <code>POSIX_V6_ILP32_OFF32_CFLAGS</code><br><code>POSIX_V6_ILP32_OFF32_LDFLAGS</code><br><code>POSIX_V6_ILP32_OFF32_LIBS</code>    |
| 8414<br>8415<br>8416 | <code>_POSIX_V6_ILP32_OFFBIG</code>                   | C Compiler Flags<br>Linker/Loader Flags<br>Libraries | <code>POSIX_V6_ILP32_OFFBIG_CFLAGS</code><br><code>POSIX_V6_ILP32_OFFBIG_LDFLAGS</code><br><code>POSIX_V6_ILP32_OFFBIG_LIBS</code> |
| 8417<br>8418<br>8419 | <code>_POSIX_V6_LP64_OFF64</code>                     | C Compiler Flags<br>Linker/Loader Flags<br>Libraries | <code>POSIX_V6_LP64_OFF64_CFLAGS</code><br><code>POSIX_V6_LP64_OFF64_LDFLAGS</code><br><code>POSIX_V6_LP64_OFF64_LIBS</code>       |
| 8420<br>8421<br>8422 | <code>_POSIX_V6_LPBIG_OFFBIG</code>                   | C Compiler Flags<br>Linker/Loader Flags<br>Libraries | <code>POSIX_V6_LPBIG_OFFBIG_CFLAGS</code><br><code>POSIX_V6_LPBIG_OFFBIG_LDFLAGS</code><br><code>POSIX_V6_LPBIG_OFFBIG_LIBS</code> |

#### 8423 **EXIT STATUS**

8424 The following exit values shall be returned:

8425     **0** Successful compilation or link edit.

8426     **>0** An error occurred.

#### 8427 **CONSEQUENCES OF ERRORS**

8428 When *c99* encounters a compilation error that causes an object file not to be created, it shall write  
 8429 a diagnostic to standard error and continue to compile other source code operands, but it shall  
 8430 not perform the link phase and return a non-zero exit status. If the link edit is unsuccessful, a  
 8431 diagnostic message shall be written to standard error and *c99* exits with a non-zero status. A  
 8432 conforming application shall rely on the exit status of *c99*, rather than on the existence or mode  
 8433 of the executable file.

8434 **APPLICATION USAGE**

8435 Since the *c99* utility usually creates files in the current directory during the compilation process,  
8436 it is typically necessary to run the *c99* utility in a directory in which a file can be created.

8437 On systems providing POSIX Conformance (see the Base Definitions volume of  
8438 IEEE Std 1003.1-200x, Chapter 2, Conformance), *c99* is required only with the C-Language  
8439 Development option; XSI-conformant systems always provide *c99*.

8440 Some historical implementations have created *.o* files when *-c* is not specified and more than  
8441 one source file is given. Since this area is left unspecified, the application cannot rely on *.o* files  
8442 being created, but it also must be prepared for any related *.o* files that already exist being deleted  
8443 at the completion of the link edit.

8444 Some historical implementations have permitted *-L* options to be interspersed with *-I* operands  
8445 on the command line. For an application to compile consistently on systems that do not behave  
8446 like this, it is necessary for a conforming application to supply all *-L* options before any of the *-I*  
8447 options.

8448 There is the possible implication that if a user supplies versions of the standard functions (before  
8449 they would be encountered by an implicit *-I c* or explicit *-I m*), that those versions would be  
8450 used in place of the standard versions. There are various reasons this might not be true  
8451 (functions defined as macros, manipulations for clean name space, and so on), so the existence of  
8452 files named in the same manner as the standard libraries within the *-L* directories is explicitly  
8453 stated to produce unspecified behavior.

8454 All of the functions specified in the System Interfaces volume of IEEE Std 1003.1-200x may be  
8455 made visible by implementations when the Standard C Library is searched. Conforming  
8456 applications must explicitly request searching the other standard libraries when functions made  
8457 visible by those libraries are used.

8458 **EXAMPLES**

8459 1. The following usage example compiles **foo.c** and creates the executable file **foo**:

```
8460 c99 -o foo foo.c
```

8461 The following usage example compiles **foo.c** and creates the object file **foo.o**:

```
8462 c99 -c foo.c
```

8463 The following usage example compiles **foo.c** and creates the executable file **a.out**:

```
8464 c99 foo.c
```

8465 The following usage example compiles **foo.c**, links it with **bar.o**, and creates the executable  
8466 file **a.out**. It also creates and leaves **foo.o**:

```
8467 c99 foo.c bar.o
```

8468 2. The following example shows how an application using threads interfaces can test for  
8469 support of and use a programming environment supporting 32-bit **int**, **long**, and **pointer**  
8470 types and an **off\_t** type using at least 64 bits:

```
8471 if [ $(getconf _POSIX_V6_ILP32_OFFBIG) != "-1" ]
8472 then
8473     c99 $(getconf POSIX_V6_ILP32_OFFBIG_CFLAGS) -D_XOPEN_SOURCE=600 \
8474         $(getconf POSIX_V6_ILP32_OFFBIG_LDFLAGS) foo.c -o foo \
8475         $(getconf POSIX_V6_ILP32_OFFBIG_LIBS) -l pthread
8476 else
8477     echo ILP32_OFFBIG programming environment not supported
```

```
8478         exit 1
8479     fi
```

8480 3. The following examples clarify the use and interactions of `-L` options and `-I` operands. |

8481 Consider the case in which module `a.c` calls function `f()` in library `libQ.a`, and module `b.c`  
8482 calls function `g()` in library `libp.a`. Assume that both libraries reside in `/a/b/c`. The  
8483 command line to compile and link in the desired way is:

```
8484 c99 -L /a/b/c main.o a.c -l Q b.c -l p
```

8485 In this case the `-I Q` operand need only precede the first `-I p` operand, since both `libQ.a`  
8486 and `libp.a` reside in the same directory.

8487 Multiple `-L` operands can be used when library name collisions occur. Building on the  
8488 previous example, suppose that the user wants to use a new `libp.a`, in `/a/a/a`, but still wants  
8489 `f()` from `/a/b/c/libQ.a`:

```
8490 c99 -L /a/a/a -L /a/b/c main.o a.c -l Q b.c -l p
```

8491 In this example, the linker searches the `-L` options in the order specified, and finds  
8492 `/a/a/a/libp.a` before `/a/b/c/libp.a` when resolving references for `b.c`. The order of the `-I`  
8493 operands is still important, however. |

8494 4. The following example shows how an application can use a programming environment |  
8495 where the widths of the following types: |

```
8496     blksize_t, cc_t, mode_t, nfd_t, pid_t, ptrdiff_t, size_t, speed_t, ssize_t, suseconds_t, |  
8497     tcflag_t, useconds_t, wchar_t, wint_t |
```

8498 are no greater than the width of type `long`: |

```
8499 # First choose one of the listed environments ... |
```

```
8500 # ... if there are no additional constraints, the first one will do: |  
8501 CENV=$(getconf _POSIX_V6_WIDTH_RESTRICTED_ENVS | head -n 1) |
```

```
8502 # ... or, if an environment that supports large files is preferred, |  
8503 # look for names that contain "OFF64" or "OFFBIG". (This chooses |  
8504 # the last one in the list if none match): |
```

```
8505 for CENV in $(getconf _POSIX_V6_WIDTH_RESTRICTED_ENVS) |  
8506 do |
```

```
8507     case $CENV in |  
8508         *OFF64*|*OFFBIG*) break ;; |  
8509     esac |
```

```
8510 done |
```

```
8511 # The chosen environment name can now be used like this: |
```

```
8512 c99 $(getconf ${CENV}_CFLAGS) -D _POSIX_C_SOURCE=200xxxL \ |  
8513 $(getconf ${CENV}_LD_FLAGS) foo.c -o foo \ |  
8514 $(getconf ${CENV}_LIBS) |
```

## 8515 RATIONALE |

8516 The `c99` utility is based on the `c89` utility originally introduced in the ISO POSIX-2: 1993 standard. |

8517 Some of the changes from `c89` include the modification to the contents of the Standard Libraries |  
8518 section to account for new headers and options; for example, `<spawn.h>` added to the `-I rt` |  
8519 operand, and the `-l trace` operand added for the Tracing functions. |

8520 **FUTURE DIRECTIONS**

8521 None.

8522 **SEE ALSO**8523 *ar*, *getconf*, *make*, *nm*, *strip*, *umask*, the System Interfaces volume of IEEE Std 1003.1-200x, |8524 *sysconf()*, the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 13, Headers |8525 **CHANGE HISTORY**

8526 First released in Issue 6. Included for alignment with the ISO/IEC 9899:1999 standard.

8527 **NAME**

8528 cal — print a calendar

8529 **SYNOPSIS**8530 xSI cal [ [*month*] *year* ]

8531

8532 **DESCRIPTION**

8533 The *cal* utility shall write a calendar to standard output using the Julian calendar for dates from  
8534 January 1, 1 through September 2, 1752 and the Gregorian calendar for dates from September 14,  
8535 1752 through December 31, 9999 as though the Gregorian calendar had been adopted on  
8536 September 14, 1752.

8537 **OPTIONS**

8538 None.

8539 **OPERANDS**

8540 The following operands shall be supported:

8541 *month* Specify the month to be displayed, represented as a decimal integer from 1  
8542 (January) to 12 (December). The default shall be the current month.

8543 *year* Specify the year for which the calendar is displayed, represented as a decimal  
8544 integer from 1 to 9999. The default shall be the current year.

8545 **STDIN**

8546 Not used.

8547 **INPUT FILES**

8548 None.

8549 **ENVIRONMENT VARIABLES**8550 The following environment variables shall affect the execution of *cal*:

8551 *LANG* Provide a default value for the internationalization variables that are unset or null.  
8552 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
8553 Internationalization Variables for the precedence of internationalization variables  
8554 used to determine the values of locale categories.)

8555 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
8556 internationalization variables.

8557 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
8558 characters (for example, single-byte as opposed to multi-byte characters in  
8559 arguments).

8560 *LC\_MESSAGES*

8561 Determine the locale that should be used to affect the format and contents of  
8562 diagnostic messages written to standard error, and informative messages written  
8563 to standard output.

8564 *LC\_TIME* Determine the format and contents of the calendar.

8565 *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

8566 *TZ* Determine the timezone used to calculate the value of the current month.

8567 **ASYNCHRONOUS EVENTS**

8568 Default.

8569 **STDOUT**

8570 The standard output shall be used to display the calendar, in an unspecified format.

8571 **STDERR**

8572 The standard error shall be used only for diagnostic messages.

8573 **OUTPUT FILES**

8574 None.

8575 **EXTENDED DESCRIPTION**

8576 None.

8577 **EXIT STATUS**

8578 The following exit values shall be returned:

8579 0 Successful completion.

8580 &gt;0 An error occurred.

8581 **CONSEQUENCES OF ERRORS**

8582 Default.

8583 **APPLICATION USAGE**

8584 Note that:

8585 cal 83

8586 refers to A.D. 83, not 1983.

8587 **EXAMPLES**

8588 None.

8589 **RATIONALE**

8590 None.

8591 **FUTURE DIRECTIONS**8592 A future version of IEEE Std 1003.1-200x may support locale-specific recognition of the date of  
8593 adoption of the Gregorian calendar.8594 **SEE ALSO**

8595 None.

8596 **CHANGE HISTORY**

8597 First released in Issue 2.

8598 **Issue 6**8599 The DESCRIPTION is updated to allow for traditional behavior for years before the adoption of  
8600 the Gregorian calendar.

8601 **NAME**

8602           cat — concatenate and print files

8603 **SYNOPSIS**8604           cat [-u][*file ...*]8605 **DESCRIPTION**8606           The *cat* utility shall read files in sequence and shall write their contents to the standard output in  
8607           the same sequence. |8608 **OPTIONS**8609           The *cat* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2,  
8610           Utility Syntax Guidelines.

8611           The following option shall be supported:

8612           **-u**           Write bytes from the input file to the standard output without delay as each is  
8613           read.8614 **OPERANDS**

8615           The following operand shall be supported:

8616           *file*           A pathname of an input file. If no *file* operands are specified, the standard input |  
8617           shall be used. If a *file* is '-', the *cat* utility shall read from the standard input at |  
8618           that point in the sequence. The *cat* utility shall not close and reopen standard input  
8619           when it is referenced in this way, but shall accept multiple occurrences of '-' as a  
8620           *file* operand.8621 **STDIN**8622           The standard input shall be used only if no *file* operands are specified, or if a *file* operand is '-'. |  
8623           See the INPUT FILES section.8624 **INPUT FILES**

8625           The input files can be any file type.

8626 **ENVIRONMENT VARIABLES**8627           The following environment variables shall affect the execution of *cat*:8628           **LANG**           Provide a default value for the internationalization variables that are unset or null.  
8629           (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
8630           Internationalization Variables for the precedence of internationalization variables  
8631           used to determine the values of locale categories.)8632           **LC\_ALL**          If set to a non-empty string value, override the values of all the other  
8633           internationalization variables.8634           **LC\_CTYPE**       Determine the locale for the interpretation of sequences of bytes of text data as  
8635           characters (for example, single-byte as opposed to multi-byte characters in  
8636           arguments).8637           **LC\_MESSAGES**8638           Determine the locale that should be used to affect the format and contents of  
8639           diagnostic messages written to standard error.8640 **XSI**           **NLSPATH**       Determine the location of message catalogs for the processing of *LC\_MESSAGES*.8641 **ASYNCHRONOUS EVENTS**

8642           Default.

8643 **STDOUT**

8644 The standard output shall contain the sequence of bytes read from the input files. Nothing else  
8645 shall be written to the standard output.

8646 **STDERR**

8647 The standard error shall be used only for diagnostic messages.

8648 **OUTPUT FILES**

8649 None.

8650 **EXTENDED DESCRIPTION**

8651 None.

8652 **EXIT STATUS**

8653 The following exit values shall be returned:

8654 0 All input files were output successfully.

8655 >0 An error occurred.

8656 **CONSEQUENCES OF ERRORS**

8657 Default.

8658 **APPLICATION USAGE**

8659 The **-u** option has value in prototyping non-blocking reads from FIFOs. The intent is to support  
8660 the following sequence:

```
8661 mkfifo foo  
8662 cat -u foo > /dev/tty13 &  
8663 cat -u > foo
```

8664 It is unspecified whether standard output is or is not buffered in the default case. This is  
8665 sometimes of interest when standard output is associated with a terminal, since buffering may  
8666 delay the output. The presence of the **-u** option guarantees that unbuffered I/O is available. It is  
8667 implementation-defined whether the *cat* utility buffers output if the **-u** option is not specified.  
8668 Traditionally, the **-u** option is implemented using the equivalent of the *setvbuf()* function  
8669 defined in the System Interfaces volume of IEEE Std 1003.1-200x.

8670 **EXAMPLES**

8671 The following command:

```
8672 cat myfile
```

8673 writes the contents of the file **myfile** to standard output.

8674 The following command:

```
8675 cat doc1 doc2 > doc.all
```

8676 concatenates the files **doc1** and **doc2** and writes the result to **doc.all**.

8677 Because of the shell language mechanism used to perform output redirection, a command such  
8678 as this:

```
8679 cat doc doc.end > doc
```

8680 causes the original data in **doc** to be lost.

8681 The command:

```
8682 cat start - middle - end > file
```

8683 when standard input is a terminal, gets two arbitrary pieces of input from the terminal with a  
 8684 single invocation of *cat*. Note, however, that if standard input is a regular file, this would be  
 8685 equivalent to the command:

```
8686 cat start - middle /dev/null end > file
```

8687 because the entire contents of the file would be consumed by *cat* the first time '-' was used as a  
 8688 *file* operand and an end-of-file condition would be detected immediately when '-' was  
 8689 referenced the second time.

#### 8690 RATIONALE

8691 Historical versions of the *cat* utility include the options *-e*, *-t*, and *-v*, which permit the ends of  
 8692 lines, <tab>s, and invisible characters, respectively, to be rendered visible in the output. The  
 8693 standard developers omitted these options because they provide too fine a degree of control  
 8694 over what is made visible, and similar output can be obtained using a command such as:

```
8695 sed -n -e 's/$/$/' -e l pathname
```

8696 The *-s* option was omitted because it corresponds to different functions in BSD and System V-  
 8697 based systems. The BSD *-s* option to squeeze blank lines can be accomplished by the shell script  
 8698 shown in following example:

```
8699 sed -n '  

  8700 # Write non-empty lines.  

  8701 ./ {  

  8702     p  

  8703     d  

  8704 }  

  8705 # Write a single empty line, then look for more empty lines.  

  8706 /^$/ p  

  8707 # Get next line, discard the held <newline> (empty line),  

  8708 # and look for more empty lines.  

  8709 :Empty  

  8710 /^$/ {  

  8711     N  

  8712     s/./.  

  8713     b Empty  

  8714 }  

  8715 # Write the non-empty line before going back to search  

  8716 # for the first in a set of empty lines.  

  8717 p  

  8718 '
```

8719 The System V *-s* option to silence error messages can be accomplished by redirecting the  
 8720 standard error. Note that the BSD documentation for *cat* uses the term "blank line" to mean the  
 8721 same as the POSIX "empty line": a line consisting only of a <newline>.

8722 The BSD *-n* option was omitted because similar functionality can be obtained from the *-n*  
 8723 option of the *pr* utility.

#### 8724 FUTURE DIRECTIONS

8725 None.

#### 8726 SEE ALSO

8727 *more*

8728 **CHANGE HISTORY**

8729 First released in Issue 2.

## 8730 NAME

8731 cd — change the working directory

## 8732 SYNOPSIS

8733 cd [-L] [-P] [*directory*]

8734 cd -

## 8735 DESCRIPTION

8736 The *cd* utility shall change the working directory of the current shell execution environment (see  
8737 Section 2.12 (on page 2263)) by executing the following steps in sequence. (In the following  
8738 steps, the symbol **curpath** represents an intermediate value used to simplify the description of  
8739 the algorithm used by *cd*. There is no requirement that **curpath** be made visible to the  
8740 application.)

- 8741 1. If no *directory* operand is given and the *HOME* environment variable is empty or  
8742 undefined, the default behavior is implementation-defined and no further steps shall be  
8743 taken.
- 8744 2. If no *directory* operand is given and the *HOME* environment variable is set to a non-empty  
8745 value, the *cd* utility shall behave as if the directory named in the *HOME* environment  
8746 variable was specified as the *directory* operand.
- 8747 3. If the *directory* operand begins with a slash character, set **curpath** to the operand and  
8748 proceed to step 7.
- 8749 4. If the first component of the *directory* operand is dot or dot-dot, proceed to step 6.
- 8750 5. Starting with the first pathname in the colon-separated pathnames of *CDPATH* (see the  
8751 ENVIRONMENT VARIABLES section) if the pathname is non-null, test if the  
8752 concatenation of that pathname, a slash character, and the *directory* operand names a  
8753 directory. If the pathname is null, test if the concatenation of dot, a slash character, and the  
8754 operand names a directory. In either case, if the resulting string names an existing  
8755 directory, set **curpath** to that string and proceed to step 7. Otherwise, repeat this step with  
8756 the next pathname in *CDPATH* until all pathnames have been tested.
- 8757 6. Set **curpath** to the string formed by the concatenation of the value of *PWD* a slash  
8758 character, and the operand.
- 8759 7. If the *-P* option is in effect, the *cd* utility shall perform actions equivalent to the *chdir()*  
8760 function, called with **curpath** as the *path* argument. If these actions succeed, the *PWD*  
8761 environment variable shall be set to an absolute pathname for the current working  
8762 directory and shall not contain filename components that, in the context of pathname  
8763 resolution, refer to a file of type symbolic link. If there is insufficient permission on the new  
8764 directory, or on any parent of that directory, to determine the current working directory,  
8765 the value of the *PWD* environment variable is unspecified. If the actions equivalent to  
8766 *chdir()* fail for any reason, the *cd* utility shall display an appropriate error message and not  
8767 alter the *PWD* environment variable. Whether the actions equivalent to *chdir()* succeed or  
8768 fail, no further steps shall be taken.
- 8769 8. The **curpath** value shall then be converted to canonical form as follows, considering each  
8770 component from beginning to end, in sequence:
  - 8771 a. Dot components and any slashes that separate them from the next component shall  
8772 be deleted.
  - 8773 b. For each dot-dot component, if there is a preceding component and it is neither root  
8774 nor dot-dot, the preceding component, all slashes separating the preceding  
8775 component from dot-dot, dot-dot, and all slashes separating dot-dot from the

8776 following component shall be deleted.

8777 c. An implementation may further simplify **curpath** by removing any trailing slash  
8778 characters that are not also leading slashes, replacing multiple non-leading  
8779 consecutive slashes with a single slash, and replacing three or more leading  
8780 slashes with a single slash. If, as a result of this canonicalization, the **curpath** variable  
8781 is null, no further steps shall be taken.

8782 9. The *cd* utility shall then perform actions equivalent to the *chdir()* function called with  
8783 **curpath** as the *path* argument. If these actions failed for any reason, the *cd* utility shall  
8784 display an appropriate error message and no further steps shall be taken. The *PWD*  
8785 environment variable shall be set to **curpath**.

8786 If, during the execution of the above steps, the *PWD* environment variable is changed, the  
8787 *OLDPWD* environment variable shall also be changed to the value of the old working directory  
8788 (that is the current working directory immediately prior to the call to *cd*).

#### 8789 OPTIONS

8790 The *cd* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2,  
8791 Utility Syntax Guidelines.

8792 The following options shall be supported by the implementation:

8793 **-L** Handle the operand dot-dot logically; symbolic link components shall not be  
8794 resolved before dot-dot components are processed (see steps 5. and 6. in the  
8795 DESCRIPTION).

8796 **-P** Handle the operand dot-dot physically; symbolic link components shall be  
8797 resolved before dot-dot components are processed (see step 4. in the  
8798 DESCRIPTION).

8799 If both **-L** and **-P** options are specified, the last of these options shall be used and all others  
8800 ignored. If neither **-L** nor **-P** is specified, the operand shall be handled dot-dot logically; see the  
8801 DESCRIPTION.

#### 8802 OPERANDS

8803 The following operands shall be supported:

8804 *directory* An absolute or relative pathname of the directory that shall become the new  
8805 working directory. The interpretation of a relative pathname by *cd* depends on the  
8806 **-L** option and the *CDPATH* and *PWD* environment variables. If *directory* is an  
8807 empty string, the results are unspecified.

8808 **-** When a hyphen is used as the operand, this shall be equivalent to the command:

```
8809 cd "$OLDPWD" && pwd
```

8810 which changes to the previous working directory and then writes its name.

#### 8811 STDIN

8812 Not used.

#### 8813 INPUT FILES

8814 None.

#### 8815 ENVIRONMENT VARIABLES

8816 The following environment variables shall affect the execution of *cd*:

8817 *CDPATH* A colon-separated list of pathnames that refer to directories. The *cd* utility shall use  
8818 this list in its attempt to change the directory, as described in the DESCRIPTION.  
8819 An empty string in place of a directory pathname represents the current directory.

- 8820 If *CDPATH* is not set, it shall be treated as if it were an empty string.
- 8821 *HOME* The name of the directory, used when no *directory* operand is specified.
- 8822 *LANG* Provide a default value for the internationalization variables that are unset or null.  
8823 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
8824 Internationalization Variables for the precedence of internationalization variables  
8825 used to determine the values of locale categories.)
- 8826 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
8827 internationalization variables.
- 8828 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
8829 characters (for example, single-byte as opposed to multi-byte characters in  
8830 arguments).
- 8831 *LC\_MESSAGES*  
8832 Determine the locale that should be used to affect the format and contents of  
8833 diagnostic messages written to standard error.
- 8834 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 8835 *OLDPWD* A pathname of the previous working directory, used by *cd -*.
- 8836 *PWD* This variable shall be set as specified in the DESCRIPTION. If an application sets  
8837 or unsets the value of *PWD*, the behavior of *cd* is unspecified.
- 8838 **ASYNCHRONOUS EVENTS**
- 8839 Default.
- 8840 **STDOUT**
- 8841 If a non-empty directory name from *CDPATH* is used, or if *cd -* is used, an absolute pathname of  
8842 the new working directory shall be written to the standard output as follows:
- 8843 "%s\n", <*new directory*>
- 8844 Otherwise, there shall be no output.
- 8845 **STDERR**
- 8846 The standard error shall be used only for diagnostic messages. |
- 8847 **OUTPUT FILES**
- 8848 None.
- 8849 **EXTENDED DESCRIPTION**
- 8850 None.
- 8851 **EXIT STATUS**
- 8852 The following exit values shall be returned:
- 8853 0 The directory was successfully changed.
- 8854 >0 An error occurred.
- 8855 **CONSEQUENCES OF ERRORS**
- 8856 The working directory shall remain unchanged.

**8857 APPLICATION USAGE**

8858 Since *cd* affects the current shell execution environment, it is always provided as a shell regular  
8859 built-in. If it is called in a subshell or separate utility execution environment, such as one of the  
8860 following:

```
8861 (cd /tmp)
8862 nohup cd
8863 find . -exec cd {} \;
```

8864 it does not affect the working directory of the caller's environment.

8865 The user must have execute (search) permission in *directory* in order to change to it.

**8866 EXAMPLES**

8867 None.

**8868 RATIONALE**

8869 The use of the *CDPATH* was introduced in the System V shell. Its use is analogous to the use of  
8870 the *PATH* variable in the shell. The BSD C shell used a shell parameter *cdpath* for this purpose.

8871 A common extension when *HOME* is undefined is to get the login directory from the user  
8872 database for the invoking user. This does not occur on System V implementations.

8873 Some historical shells, such as the KornShell, took special actions when the directory name  
8874 contained a dot-dot component, selecting the logical parent of the directory, rather than the  
8875 actual parent directory; that is, it moved up one level toward the '/' in the pathname,  
8876 remembering what the user typed, rather than performing the equivalent of:

```
8877 chdir("../");
```

8878 In such a shell, the following commands would not necessarily produce equivalent output for all  
8879 directories:

```
8880 cd .. && ls      ls ..
```

8881 This behavior is not permitted by default because it is not consistent with the definition of dot-  
8882 dot in most historical practice; that is, while this behavior has been optionally available in the  
8883 KornShell, other shells have historically not supported this functionality. The logical pathname  
8884 is stored in the *PWD* environment variable when the *cd* utility completes and this value is used  
8885 to construct the next directory name if *cd* is invoked with the *-L* option.

**8886 FUTURE DIRECTIONS**

8887 None.

**8888 SEE ALSO**

8889 *pwd*, the System Interfaces volume of IEEE Std 1003.1-200x, *chdir()*

**8890 CHANGE HISTORY**

8891 First released in Issue 2.

**8892 Issue 6**

8893 The following new requirements on POSIX implementations derive from alignment with the  
8894 Single UNIX Specification:

- 8895 • The *cd-*, *PWD*, and *OLDPWD* are added.

8896 The *-L* and *-P* options are added to align with the IEEE P1003.2b draft standard. This also  
8897 includes the introduction of a new description to include the effect of these options.

## 8898 NAME

8899 cflow — generate a C-language flowgraph (DEVELOPMENT)

## 8900 SYNOPSIS

```
8901 xSI cflow [-r][-d num][-D name[=def]] ... [-i incl][-I dir] ...
8902 [-U dir] ... file ...
```

8903

## 8904 DESCRIPTION

8905 The *cflow* utility shall analyze a collection of object files or assembler, C-language, *lex* or *yacc*  
 8906 source files, and attempt to build a graph, written to standard output, charting the external  
 8907 references.

## 8908 OPTIONS

8909 The *cflow* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section  
 8910 12.2, Utility Syntax Guidelines, except that the order of the **-D**, **-I**, and **-U** options (which are  
 8911 identical to their interpretation by *c99*) is significant.

8912 The following options shall be supported:

8913 **-d num** Indicate the depth at which the flowgraph is cut off. The application shall ensure  
 8914 that the argument *num* is a decimal integer. By default this is a very large number  
 8915 (typically greater than 32 000). Attempts to set the cut-off depth to a non-positive  
 8916 integer shall be ignored.

8917 **-i incl** Increase the number of included symbols. The *incl* option-argument is one of the  
 8918 following characters:

8919 *x* Include external and static data symbols. The default shall be to include only  
 8920 functions in the flowgraph.

8921 *\_* (Underscore) Include names that begin with an underscore. The default shall  
 8922 be to exclude these functions (and data if **-i x** is used).

8923 **-r** Reverse the caller: callee relationship, producing an inverted listing showing the  
 8924 callers of each function. The listing shall also be sorted in lexicographical order by  
 8925 callee.

## 8926 OPERANDS

8927 The following operand is supported:

8928 *file* The pathname of a file for which a graph is to be generated. Filenames suffixed by  
 8929 *.I* shall be taken to be *lex* input, *.y* as *yacc* input, *.c* as *c99* input, and *.i* as the  
 8930 output of *c99 -E*. Such files shall be processed as appropriate, determined by their  
 8931 suffix.

8932 Files suffixed in *.s* (conventionally assembler source) may have more limited  
 8933 information extracted from them.

## 8934 STDIN

8935 Not used.

## 8936 INPUT FILES

8937 The input files shall be object files or assembler, C-language, *lex* or *yacc* source files.

## 8938 ENVIRONMENT VARIABLES

8939 The following environment variables shall affect the execution of *cflow*:

8940 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 8941 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,

- 8942 Internationalization Variables for the precedence of internationalization variables  
8943 used to determine the values of locale categories.)
- 8944 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
8945 internationalization variables.
- 8946 *LC\_COLLATE*  
8947 Determine the locale for the ordering of the output when the *-r* option is used.
- 8948 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
8949 characters (for example, single-byte as opposed to multi-byte characters in  
8950 arguments and input files).
- 8951 *LC\_MESSAGES*  
8952 Determine the locale that should be used to affect the format and contents of  
8953 diagnostic messages written to standard error.
- 8954 *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 8955 **ASYNCHRONOUS EVENTS**  
8956 Default.
- 8957 **STDOUT**  
8958 The flowgraph written to standard output shall be formatted as follows:  
8959 `"%d %s:%s\n", <reference number>, <global>, <definition>`
- 8960 Each line of output begins with a reference (that is, line) number, followed by indentation of at  
8961 least one column position per level. This is followed by the name of the global, a colon, and its  
8962 definition. Normally globals are only functions not defined as an external or beginning with an  
8963 underscore; see the *OPTIONS* section for the *-i* inclusion option. For information extracted from  
8964 C-language source, the definition consists of an abstract type declaration (for example, *char \**)  
8965 and, delimited by angle brackets, the name of the source file and the line number where the  
8966 definition was found. Definitions extracted from object files indicate the filename and location  
8967 counter under which the symbol appeared (for example, *text*).
- 8968 Once a definition of a name has been written, subsequent references to that name contain only  
8969 the reference number of the line where the definition can be found. For undefined references,  
8970 only "*< >*" shall be written.
- 8971 **STDERR**  
8972 The standard error shall be used only for diagnostic messages.
- 8973 **OUTPUT FILES**  
8974 None.
- 8975 **EXTENDED DESCRIPTION**  
8976 None.
- 8977 **EXIT STATUS**  
8978 The following exit values shall be returned:  
8979 *0* Successful completion.  
8980 *>0* An error occurred.
- 8981 **CONSEQUENCES OF ERRORS**  
8982 Default.

8983 **APPLICATION USAGE**

8984 Files produced by *lex* and *yacc* cause the reordering of line number declarations, and this can  
8985 confuse *cflow*. To obtain proper results, the input of *yacc* or *lex* must be directed to *cflow*.

8986 **EXAMPLES**

8987 Given the following in **file.c**:

```
8988     int i;  
8989     int f();  
8990     int g();  
8991     int h();  
8992     int  
8993     main()  
8994     {  
8995         f();  
8996         g();  
8997         f();  
8998     }  
8999     int  
9000     f()  
9001     {  
9002         i = h();  
9003     }
```

9004 The command:

```
9005 cflow -i x file.c
```

9006 produces the output:

```
9007     1 main: int(), <file.c 6>  
9008     2     f: int(), <file.c 13>  
9009     3         h: <>  
9010     4         i: int, <file.c 1>  
9011     5     g: <>
```

9012 **RATIONALE**

9013 None.

9014 **FUTURE DIRECTIONS**

9015 None.

9016 **SEE ALSO**

9017 *c99*, *lex*, *yacc*

9018 **CHANGE HISTORY**

9019 First released in Issue 2.

9020 **Issue 6**

9021 The normative text is reworded to avoid use of the term “must” for application requirements.

9022 **NAME**

9023 chgrp — change the file group ownership

9024 **SYNOPSIS**9025 chgrp -hR *group file ...*9026 chgrp -R [-H | -L | -P ] *group file ...*9027 **DESCRIPTION**9028 The *chgrp* utility shall set the group ID of the file named by each *file* operand to the group ID  
9029 specified by the *group* operand.9030 For each *file* operand, or, if the **-R** option is used, each file encountered while walking the  
9031 directory trees specified by the *file* operands, the *chgrp* utility shall perform actions equivalent to  
9032 the *chown()* function defined in the System Interfaces volume of IEEE Std 1003.1-200x, called  
9033 with the following arguments:

- 9034 • The *file* operand shall be used as the *path* argument.
- 9035 • The user ID of the file shall be used as the *owner* argument.
- 9036 • The specified group ID shall be used as the *group* argument.

9037 Unless *chgrp* is invoked by a process with appropriate privileges, the set-user-ID and set-group-  
9038 ID bits of a regular file shall be cleared upon successful completion; the set-user-ID and set-  
9039 group-ID bits of other file types may be cleared.9040 **OPTIONS**9041 The *chgrp* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section  
9042 12.2, Utility Syntax Guidelines.

9043 The following options shall be supported by the implementation:

9044 **-h** If the system supports group IDs for symbolic links, for each *file* operand that  
9045 names a file of type symbolic link, *chgrp* shall attempt to set the group ID of the  
9046 symbolic link instead of the file referenced by the symbolic link. If the system does  
9047 not support group IDs for symbolic links, for each *file* operand that names a file of  
9048 type symbolic link, *chgrp* shall do nothing more with the current file and shall go  
9049 on to any remaining files.

9050 **-H** If the **-R** option is specified and a symbolic link referencing a file of type directory  
9051 is specified on the command line, *chgrp* shall change the group of the directory  
9052 referenced by the symbolic link and all files in the file hierarchy below it.

9053 **-L** If the **-R** option is specified and a symbolic link referencing a file of type directory  
9054 is specified on the command line or encountered during the traversal of a file  
9055 hierarchy, *chgrp* shall change the group of the directory referenced by the symbolic  
9056 link and all files in the file hierarchy below it.

9057 **-P** If the **-R** option is specified and a symbolic link is specified on the command line  
9058 or encountered during the traversal of a file hierarchy, *chgrp* shall change the  
9059 group ID of the symbolic link if the system supports this operation. The *chgrp*  
9060 utility shall not follow the symbolic link to any other part of the file hierarchy.

9061 **-R** Recursively change file group IDs. For each *file* operand that names a directory,  
9062 *chgrp* shall change the group of the directory and all files in the file hierarchy below  
9063 it. Unless a **-H**, **-L**, or **-P** option is specified, it is unspecified which of these  
9064 options will be used as the default.

9065 Specifying more than one of the mutually-exclusive options **-H**, **-L**, and **-P** shall not be  
9066 considered an error. The last option specified shall determine the behavior of the utility.

#### 9067 OPERANDS

9068 The following operands shall be supported:

9069 *group* A group name from the group database or a numeric group ID. Either specifies a  
9070 group ID to be given to each file named by one of the *file* operands. If a numeric  
9071 *group* operand exists in the group database as a group name, the group ID number  
9072 associated with that group name is used as the group ID.

9073 *file* A pathname of a file whose group ID is to be modified.

#### 9074 STDIN

9075 Not used.

#### 9076 INPUT FILES

9077 None.

#### 9078 ENVIRONMENT VARIABLES

9079 The following environment variables shall affect the execution of *chgrp*:

9080 *LANG* Provide a default value for the internationalization variables that are unset or null.  
9081 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
9082 Internationalization Variables for the precedence of internationalization variables  
9083 used to determine the values of locale categories.)

9084 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
9085 internationalization variables.

9086 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
9087 characters (for example, single-byte as opposed to multi-byte characters in  
9088 arguments).

#### 9089 LC\_MESSAGES

9090 Determine the locale that should be used to affect the format and contents of  
9091 diagnostic messages written to standard error.

9092 *XSI* *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

#### 9093 ASYNCHRONOUS EVENTS

9094 Default.

#### 9095 STDOUT

9096 Not used.

#### 9097 STDERR

9098 The standard error shall be used only for diagnostic messages.

#### 9099 OUTPUT FILES

9100 None.

#### 9101 EXTENDED DESCRIPTION

9102 None.

#### 9103 EXIT STATUS

9104 The following exit values shall be returned:

9105 0 The utility executed successfully and all requested changes were made.

9106 >0 An error occurred.

9107 **CONSEQUENCES OF ERRORS**

9108 Default.

9109 **APPLICATION USAGE**9110 Only the owner of a file or the user with appropriate privileges may change the owner or group  
9111 of a file.9112 Some implementations restrict the use of *chgrp* to a user with appropriate privileges when the  
9113 *group* specified is not the effective group ID or one of the supplementary group IDs of the calling  
9114 process.9115 **EXAMPLES**

9116 None.

9117 **RATIONALE**9118 The System V and BSD versions use different exit status codes. Some implementations used the  
9119 exit status as a count of the number of errors that occurred; this practice is unworkable since it  
9120 can overflow the range of valid exit status values. The standard developers chose to mask these  
9121 by specifying only 0 and >0 as exit values.9122 The functionality of *chgrp* is described substantially through references to *chown()*. In this way,  
9123 there is no duplication of effort required for describing the interactions of permissions, multiple  
9124 groups, and so on.9125 **FUTURE DIRECTIONS**

9126 None.

9127 **SEE ALSO**9128 *chmod*, *chown*, the System Interfaces volume of IEEE Std 1003.1-200x, *chown()*9129 **CHANGE HISTORY**

9130 First released in Issue 2.

9131 **Issue 6**9132 New options **-H**, **-L**, and **-P** are added to align with the IEEE P1003.2b draft standard. These  
9133 options affect the processing of symbolic links.9134 IEEE PASC Interpretation 1003.2 #172 is applied, changing the CONSEQUENCES OF ERRORS  
9135 section to "Default."

9136 **NAME**

9137           chmod — change the file modes

9138 **SYNOPSIS**9139           chmod [-R] *mode file ...*9140 **DESCRIPTION**9141           The *chmod* utility shall change any or all of the file mode bits of the file named by each *file*  
9142           operand in the way specified by the *mode* operand.9143           It is implementation-defined whether and how the *chmod* utility affects any alternate or  
9144           additional file access control mechanism (see the Base Definitions volume of  
9145           IEEE Std 1003.1-200x, Section 4.4, File Access Permissions) being used for the specified file.9146           Only a process whose effective user ID matches the user ID of the file, or a process with the  
9147           appropriate privileges, shall be permitted to change the file mode bits of a file.9148 **OPTIONS**9149           The *chmod* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section  
9150           12.2, Utility Syntax Guidelines.

9151           The following option shall be supported:

9152           **-R**           Recursively change file mode bits. For each *file* operand that names a directory,  
9153           *chmod* shall change the file mode bits of the directory and all files in the file  
9154           hierarchy below it.9155 **OPERANDS**

9156           The following operands shall be supported:

9157           *mode*           Represents the change to be made to the file mode bits of each file named by one of  
9158           the *file* operands; see the EXTENDED DESCRIPTION section.9159           *file*           A pathname of a file whose file mode bits shall be modified.9160 **STDIN**

9161           Not used.

9162 **INPUT FILES**

9163           None.

9164 **ENVIRONMENT VARIABLES**9165           The following environment variables shall affect the execution of *chmod*:9166           **LANG**           Provide a default value for the internationalization variables that are unset or null.  
9167           (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
9168           Internationalization Variables for the precedence of internationalization variables  
9169           used to determine the values of locale categories.)9170           **LC\_ALL**        If set to a non-empty string value, override the values of all the other  
9171           internationalization variables.9172           **LC\_CTYPE**     Determine the locale for the interpretation of sequences of bytes of text data as  
9173           characters (for example, single-byte as opposed to multi-byte characters in  
9174           arguments).9175           **LC\_MESSAGES**9176           Determine the locale that should be used to affect the format and contents of  
9177           diagnostic messages written to standard error.

- 9178 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 9179 **ASYNCHRONOUS EVENTS**
- 9180 Default.
- 9181 **STDOUT**
- 9182 Not used.
- 9183 **STDERR**
- 9184 The standard error shall be used only for diagnostic messages.
- 9185 **OUTPUT FILES**
- 9186 None.
- 9187 **EXTENDED DESCRIPTION**
- 9188 The *mode* operand shall be either a *symbolic\_mode* expression or a non-negative octal integer. The
- 9189 *symbolic\_mode* form is described by the grammar later in this section.
- 9190 Each **clause** shall specify an operation to be performed on the current file mode bits of each *file*.
- 9191 The operations shall be performed on each *file* in the order in which the **clauses** are specified.
- 9192 The **who** symbols **u**, **g**, and **o** shall specify the *user*, *group*, and *other* parts of the file mode bits,
- 9193 respectively. A **who** consisting of the symbol **a** shall be equivalent to **ugo**.
- 9194 The **perm** symbols **r**, **w**, and **x** represent the *read*, *write*, and *execute/search* portions of file mode
- 9195 bits, respectively. The **perm** symbol **s** shall represent the *set-user-ID-on-execution* (when **who**
- 9196 contains or implies **u**) and *set-group-ID-on-execution* (when **who** contains or implies **g**) bits.
- 9197 The **perm** symbol **X** shall represent the execute/search portion of the file mode bits if the file is a
- 9198 directory or if the current (unmodified) file mode bits have at least one of the execute bits
- 9199 (S\_IXUSR, S\_IXGRP, or S\_IXOTH) set. It shall be ignored if the file is not a directory and none of
- 9200 the execute bits are set in the current file mode bits.
- 9201 The **permcop** symbols **u**, **g**, and **o** shall represent the current permissions associated with the
- 9202 user, group, and other parts of the file mode bits, respectively. For the remainder of this section,
- 9203 **perm** refers to the non-terminals **perm** and **permcop** in the grammar.
- 9204 If multiple **actionlists** are grouped with a single **wholist** in the grammar, each **actionlist** shall be
- 9205 applied in the order specified with that **wholist**. The *op* symbols shall represent the operation
- 9206 performed, as follows:
- 9207 + If **perm** is not specified, the '+' operation shall not change the file mode bits.
- 9208 If **who** is not specified, the file mode bits represented by **perm** for the owner, group, and
- 9209 other permissions, except for those with corresponding bits in the file mode creation mask
- 9210 of the invoking process, shall be set.
- 9211 Otherwise, the file mode bits represented by the specified **who** and **perm** values shall be set.
- 9212 - If **perm** is not specified, the '-' operation shall not change the file mode bits.
- 9213 If **who** is not specified, the file mode bits represented by **perm** for the owner, group, and
- 9214 other permissions, except for those with corresponding bits in the file mode creation mask
- 9215 of the invoking process, shall be cleared.
- 9216 Otherwise, the file mode bits represented by the specified **who** and **perm** values shall be
- 9217 cleared.
- 9218 = Clear the file mode bits specified by the **who** value, or, if no **who** value is specified, all of the
- 9219 file mode bits specified in this volume of IEEE Std 1003.1-200x.

9220 If **perm** is not specified, the '=' operation shall make no further modifications to the file  
 9221 mode bits.

9222 If **who** is not specified, the file mode bits represented by **perm** for the owner, group, and  
 9223 other permissions, except for those with corresponding bits in the file mode creation mask  
 9224 of the invoking process, shall be set.

9225 Otherwise, the file mode bits represented by the specified **who** and **perm** values shall be set.

9226 When using the symbolic mode form on a regular file, it is implementation-defined whether or  
 9227 not:

- 9228 • Requests to set the set-user-ID-on-execution or set-group-ID-on-execution bit when all  
 9229 execute bits are currently clear and none are being set are ignored.
- 9230 • Requests to clear all execute bits also clear the set-user-ID-on-execution and set-group-ID-  
 9231 on-execution bits.
- 9232 • Requests to clear the set-user-ID-on-execution or set-group-ID-on-execution bits when all  
 9233 execute bits are currently clear are ignored. However, if the command *ls -l file* writes an *s* in  
 9234 the position indicating that the set-user-ID-on-execution or set-group-ID-on-execution is set,  
 9235 the commands *chmod u-s file* or *chmod g-s file*, respectively, shall not be ignored.

9236 When using the symbolic mode form on other file types, it is implementation-defined whether  
 9237 or not requests to set or clear the set-user-ID-on-execution or set-group-ID-on-execution bits are  
 9238 honored.

9239 If the **who** symbol **o** is used in conjunction with the **perm** symbol **s** with no other **who** symbols  
 9240 being specified, the set-user-ID-on-execution and set-group-ID-on-execution bits shall not be  
 9241 modified. It shall not be an error to specify the **who** symbol **o** in conjunction with the **perm**  
 9242 symbol **s**.

9243 For an octal integer *mode* operand, the file mode bits shall be set absolutely.

9244 For each bit set in the octal number, the corresponding file permission bit shown in the following  
 9245 table shall be set; all other file permission bits shall be cleared. For regular files, for each bit set in  
 9246 the octal number corresponding to the set-user-ID-on-execution or the set-group-ID-on-  
 9247 execution, bits shown in the following table shall be set; if these bits are not set in the octal  
 9248 number, they are cleared. For other file types, it is implementation-defined whether or not  
 9249 requests to set or clear the set-user-ID-on-execution or set-group-ID-on-execution bits are  
 9250 honored.

| Octal | Mode Bit | | | | | | |
|---|---|---|---|---|---|---|---|
| 4000  | S_ISUID  | 0400  | S_IRUSR  | 0040  | S_IRGRP  | 0004  | S_IROTH  |
| 2000  | S_ISGID  | 0200  | S_IWUSR  | 0020  | S_IWGRP  | 0002  | S_IWOTH  |
|       |          | 0100  | S_IXUSR  | 0010  | S_IXGRP  | 0001  | S_IXOTH  |

9255 When bits are set in the octal number other than those listed in the table above, the behavior is  
 9256 unspecified.

9257 **Grammar for chmod**

9258 The grammar and lexical conventions in this section describe the syntax for the *symbolic\_mode*  
 9259 operand. The general conventions for this style of grammar are described in Section 1.10 (on  
 9260 page 2220). A valid *symbolic\_mode* can be represented as the non-terminal symbol *symbolic\_mode*  
 9261 in the grammar. This formal syntax shall take precedence over the preceding text syntax  
 9262 description.

9263 The lexical processing is based entirely on single characters. Implementations need not allow  
 9264 blank characters within the single argument being processed.

```

9265 %start    symbolic_mode
9266 %%

9267 symbolic_mode    : section
9268                  | symbolic_mode ',' clause
9269                  ;

9270 clause          : actionlist
9271                  | wholist actionlist
9272                  ;

9273 wholist         : who
9274                  | wholist who
9275                  ;

9276 who            : 'u' | 'g' | 'o' | 'a'
9277                  ;

9278 actionlist     : action
9279                  | actionlist action
9280                  ;

9281 action         : op
9282                  | op permlist
9283                  | op permcopy
9284                  ;

9285 permcopy       : 'u' | 'g' | 'o'
9286                  ;

9287 op            : '+' | '-' | '='
9288                  ;

9289 permlist       : perm
9290                  | perm permlist
9291                  ;

9292 perm          : 'r' | 'w' | 'x' | 'X' | 's'
9293                  ;

```

9294 **EXIT STATUS**

9295 The following exit values shall be returned:

9296 **0** The utility executed successfully and all requested changes were made.

9297 **>0** An error occurred.

9298 **CONSEQUENCES OF ERRORS**

9299 Default.

9300 **APPLICATION USAGE**

9301 Some implementations of the *chmod* utility change the mode of a directory before the files in the  
 9302 directory when performing a recursive (**-R** option) change; others change the directory mode  
 9303 after the files in the directory. If an application tries to remove read or search permission for a  
 9304 file hierarchy, the removal attempt fails if the directory is changed first; on the other hand, trying  
 9305 to re-enable permissions to a restricted hierarchy fails if directories are changed last. Users  
 9306 should not try to make a hierarchy inaccessible to themselves.

9307 Some implementations of *chmod* never used the process' *umask* when changing modes; systems  
 9308 conformant with this volume of IEEE Std 1003.1-200x do so when **who** is not specified. Note the  
 9309 difference between:

9310 `chmod a-w file`

9311 which removes all write permissions, and:

9312 `chmod -- -w file`

9313 which removes write permissions that would be allowed if **file** was created with the same  
 9314 *umask*.

9315 Conforming applications should never assume that they know how the set-user-ID and set-  
 9316 group-ID bits on directories are interpreted.

9317 **EXAMPLES**

9318

9319

9320

9321

9322

9323

9324

9325

9326

9327

| Mode               | Results                                                                                                  |
|--------------------|----------------------------------------------------------------------------------------------------------|
| <code>a+=</code>   | Equivalent to <code>a+,a=</code> ; clears all file mode bits.                                            |
| <code>go+-w</code> | Equivalent to <code>go+,go-w</code> ; clears group and other write bits.                                 |
| <code>g=o-w</code> | Equivalent to <code>g=o,g-w</code> ; sets group bit to match other bits and then clears group write bit. |
| <code>g-r+w</code> | Equivalent to <code>g-r,g+w</code> ; clears group read bit and sets group write bit.                     |
| <code>=g</code>    | Sets owner bits to match group bits and sets other bits to match group bits.                             |

9328 **RATIONALE**

9329 The functionality of *chmod* is described substantially through references to concepts defined in  
 9330 the System Interfaces volume of IEEE Std 1003.1-200x. In this way, there is less duplication of  
 9331 effort required for describing the interactions of permissions. However, the behavior of this  
 9332 utility is not described in terms of the *chmod()* function from the System Interfaces volume of  
 9333 IEEE Std 1003.1-200x because that specification requires certain side effects upon alternate file  
 9334 access control mechanisms that might not be appropriate, depending on the implementation.

9335 Implementations that support mandatory file and record locking as specified by the 1984  
 9336 /usr/group standard historically used the combination of set-group-ID bit set and group  
 9337 execute bit clear to indicate mandatory locking. This condition is usually set or cleared with the  
 9338 symbolic mode **perm** symbol **l** instead of the **perm** symbols **s** and **x** so that the mandatory  
 9339 locking mode is not changed without explicit indication that that was what the user intended.  
 9340 Therefore, the details on how the implementation treats these conditions must be defined in the  
 9341 documentation. This volume of IEEE Std 1003.1-200x does not require mandatory locking (nor  
 9342 does the System Interfaces volume of IEEE Std 1003.1-200x), but does allow it as an extension.  
 9343 However, this volume of IEEE Std 1003.1-200x does require that the *ls* and *chmod* utilities work

9344 consistently in this area. If *ls -l file* indicates that the set-group-ID bit is set, *chmod g-s file* must  
 9345 clear it (assuming appropriate privileges exist to change modes).

9346 The System V and BSD versions use different exit status codes. Some implementations used the  
 9347 exit status as a count of the number of errors that occurred; this practice is unworkable since it  
 9348 can overflow the range of valid exit status values. This problem is avoided here by specifying  
 9349 only 0 and >0 as exit values.

9350 The System Interfaces volume of IEEE Std 1003.1-200x indicates that implementation-defined  
 9351 restrictions may cause the S\_ISUID and S\_ISGID bits to be ignored. This volume of  
 9352 IEEE Std 1003.1-200x allows the *chmod* utility to choose to modify these bits before calling  
 9353 *chmod()* (or some function providing equivalent capabilities) for non-regular files. Among other  
 9354 things, this allows implementations that use the set-user-ID and set-group-ID bits on directories  
 9355 to enable extended features to handle these extensions in an intelligent manner.

9356 The **X perm** symbol was adopted from BSD-based systems because it provides commonly  
 9357 desired functionality when doing recursive (**-R** option) modifications. Similar functionality is  
 9358 not provided by the *find* utility. Historical BSD versions of *chmod*, however, only supported **X**  
 9359 with *op+*; it has been extended in this volume of IEEE Std 1003.1-200x because it is also useful  
 9360 with *op=*. (It has also been added for *op-* even though it duplicates **x**, in this case, because it is  
 9361 intuitive and easier to explain.)

9362 The grammar was extended with the *permcop* non-terminal to allow historical-practice forms of  
 9363 symbolic modes like **o=u -g** (that is, set the “other” permissions to the permissions of “owner”  
 9364 minus the permissions of “group”).

#### 9365 **FUTURE DIRECTIONS**

9366 None.

#### 9367 **SEE ALSO**

9368 *ls*, *umask*, the System Interfaces volume of IEEE Std 1003.1-200x, *chmod()*

#### 9369 **CHANGE HISTORY**

9370 First released in Issue 2.

#### 9371 **Issue 6**

9372 The following new requirements on POSIX implementations derive from alignment with the  
 9373 Single UNIX Specification:

- 9374 • Octal modes have been kept and made mandatory despite being marked obsolescent in the  
 9375 previous version of this volume of IEEE Std 1003.1-200x.

9376 IEEE PASC Interpretation 1003.2 #172 is applied, changing the CONSEQUENCES OF ERRORS  
 9377 section to “Default.”.

9378 **NAME**

9379 chown — change the file ownership

9380 **SYNOPSIS**

9381 chown -hR owner[:group] file ...

9382 chown -R [-H | -L | -P ] owner[:group] file ...

9383 **DESCRIPTION**9384 The *chown* utility shall set the user ID of the file named by each *file* operand to the user ID  
9385 specified by the *owner* operand.9386 For each *file* operand, or, if the **-R** option is used, each file encountered while walking the  
9387 directory trees specified by the *file* operands, the *chown* utility shall perform actions equivalent to  
9388 the *chown()* function defined in the System Interfaces volume of IEEE Std 1003.1-200x, called  
9389 with the following arguments:

- 9390 1. The
- file*
- operand shall be used as the
- path*
- argument.
- 
- 9391 2. The user ID indicated by the
- owner*
- portion of the first operand shall be used as the
- owner*
- 
- 9392 argument.
- 
- 9393 3. If the
- group*
- portion of the first operand is given, the group ID indicated by it shall be used
- 
- 9394 as the
- group*
- argument; otherwise, the group ID of the file shall be used as the
- group*
- 
- 9395 argument.

9396 Unless *chown* is invoked by a process with appropriate privileges, the set-user-ID and set-  
9397 group-ID bits of a regular file shall be cleared upon successful completion; the set-user-ID and  
9398 set-group-ID bits of other file types may be cleared.9399 **OPTIONS**9400 The *chown* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section  
9401 12.2, Utility Syntax Guidelines.

9402 The following options shall be supported by the implementation:

9403 **-h** If the system supports user IDs for symbolic links, for each *file* operand that names  
9404 a file of type symbolic link, *chown* shall attempt to set the user ID of the symbolic  
9405 link. If the system supports group IDs for symbolic links, and a group ID was  
9406 specified, for each *file* operand that names a file of type symbolic link, *chown* shall  
9407 attempt to set the group ID of the symbolic link. If the system does not support  
9408 user or group IDs for symbolic links, for each *file* operand that names a file of type  
9409 symbolic link, *chown* shall do nothing more with the current file and shall go on to  
9410 any remaining files.9411 **-H** If the **-R** option is specified and a symbolic link referencing a file of type directory  
9412 is specified on the command line, *chown* shall change the user ID (and group ID, if  
9413 specified) of the directory referenced by the symbolic link and all files in the file  
9414 hierarchy below it.9415 **-L** If the **-R** option is specified and a symbolic link referencing a file of type directory  
9416 is specified on the command line or encountered during the traversal of a file  
9417 hierarchy, *chown* shall change the user ID (and group ID, if specified) of the  
9418 directory referenced by the symbolic link and all files in the file hierarchy below it.9419 **-P** If the **-R** option is specified and a symbolic link is specified on the command line  
9420 or encountered during the traversal of a file hierarchy, *chown* shall change the  
9421 owner ID (and group ID, if specified) of the symbolic link if the system supports  
9422 this operation. The *chown* utility shall not follow the symbolic link to any other

- 9423 part of the file hierarchy.
- 9424 **-R** Recursively change file user and group IDs. For each *file* operand that names a  
9425 directory, *chown* shall change the user ID (and group ID, if specified) of the  
9426 directory and all files in the file hierarchy below it. Unless a **-H**, **-L**, or **-P** option is  
9427 specified, it is unspecified which of these options will be used as the default.
- 9428 Specifying more than one of the mutually-exclusive options **-H**, **-L**, and **-P** shall not be  
9429 considered an error. The last option specified shall determine the behavior of the utility.
- 9430 **OPERANDS**
- 9431 The following operands shall be supported:
- 9432 *owner[:group]* A user ID and optional group ID to be assigned to *file*. The *owner* portion of this |  
9433 operand shall be a user name from the user database or a numeric user ID. Either |  
9434 specifies a user ID which shall be given to each file named by one of the *file* |  
9435 operands. If a numeric *owner* operand exists in the user database as a user name, |  
9436 the user ID number associated with that user name shall be used as the user ID. |  
9437 Similarly, if the *group* portion of this operand is present, it shall be a group name |  
9438 from the group database or a numeric group ID. Either specifies a group ID which |  
9439 shall be given to each file. If a numeric group operand exists in the group database |  
9440 as a group name, the group ID number associated with that group name shall be |  
9441 used as the group ID. |
- 9442 *file* A pathname of a file whose user ID is to be modified.
- 9443 **STDIN**
- 9444 Not used.
- 9445 **INPUT FILES**
- 9446 None.
- 9447 **ENVIRONMENT VARIABLES**
- 9448 The following environment variables shall affect the execution of *chown*:
- 9449 *LANG* Provide a default value for the internationalization variables that are unset or null.  
9450 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
9451 Internationalization Variables for the precedence of internationalization variables  
9452 used to determine the values of locale categories.)
- 9453 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
9454 internationalization variables.
- 9455 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
9456 characters (for example, single-byte as opposed to multi-byte characters in  
9457 arguments).
- 9458 *LC\_MESSAGES*
- 9459 Determine the locale that should be used to affect the format and contents of  
9460 diagnostic messages written to standard error.
- 9461 *XSI* *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 9462 **ASYNCHRONOUS EVENTS**
- 9463 Default.

9464 **STDOUT**

9465 Not used.

9466 **STDERR**

9467 The standard error shall be used only for diagnostic messages. |

9468 **OUTPUT FILES**

9469 None.

9470 **EXTENDED DESCRIPTION**

9471 None.

9472 **EXIT STATUS**

9473 The following exit values shall be returned:

9474 0 The utility executed successfully and all requested changes were made.

9475 &gt;0 An error occurred.

9476 **CONSEQUENCES OF ERRORS**

9477 Default.

9478 **APPLICATION USAGE**9479 Only the owner of a file or the user with appropriate privileges may change the owner or group  
9480 of a file.9481 Some implementations restrict the use of *chown* to a user with appropriate privileges. |9482 **EXAMPLES**

9483 None.

9484 **RATIONALE**9485 The System V and BSD versions use different exit status codes. Some implementations used the  
9486 exit status as a count of the number of errors that occurred; this practice is unworkable since it  
9487 can overflow the range of valid exit status values. These are masked by specifying only 0 and >0  
9488 as exit values.9489 The functionality of *chown* is described substantially through references to functions in the  
9490 System Interfaces volume of IEEE Std 1003.1-200x. In this way, there is no duplication of effort  
9491 required for describing the interactions of permissions, multiple groups, and so on.9492 The 4.3 BSD method of specifying both owner and group was included in this volume of  
9493 IEEE Std 1003.1-200x because:

- 9494 • There are cases where the desired end condition could not be achieved using the
- chgrp*
- and
- 
- 9495
- chown*
- (that only changed the user ID) utilities. (If the current owner is not a member of the
- 
- 9496 desired group and the desired owner is not a member of the current group, the
- chown*
- ()
- 
- 9497 function could fail unless both owner and group are changed at the same time.)
- 
- 9498 • Even if they could be changed independently, in cases where both are being changed, there is
- 
- 9499 a 100% performance penalty caused by being forced to invoke both utilities.

9500 The BSD syntax *user[.group]* was changed to *user[:group]* in this volume of IEEE Std 1003.1-200x  
9501 because the period is a valid character in login names (as specified by the Base Definitions  
9502 volume of IEEE Std 1003.1-200x, login names consist of characters in the portable filename  
9503 character set). The colon character was chosen as the replacement for the period character  
9504 because it would never be allowed as a character in a user name or group name on historical  
9505 implementations.9506 The **-R** option is considered by some observers as an undesirable departure from the historical  
9507 UNIX system tools approach; since a tool, *find*, already exists to recurse over directories, there

9508 seemed to be no good reason to require other tools to have to duplicate that functionality.  
9509 However, the **-R** option was deemed an important user convenience, is far more efficient than  
9510 forking a separate process for each element of the directory hierarchy, and is in widespread  
9511 historical use.

9512 **FUTURE DIRECTIONS**

9513 None.

9514 **SEE ALSO**

9515 *chmod*, *chgrp*, the System Interfaces volume of IEEE Std 1003.1-200x, *chown()*

9516 **CHANGE HISTORY**

9517 First released in Issue 2.

9518 **Issue 6**

9519 New options **-h**, **-H**, **-L**, and **-P** are added to align with the IEEE P1003.2b draft standard. These  
9520 options affect the processing of symbolic links.

9521 The normative text is reworded to avoid use of the term “must” for application requirements.

9522 IEEE PASC Interpretation 1003.2 #172 is applied, changing the CONSEQUENCES OF ERRORS  
9523 section is changed to “Default.”.

9524 **NAME**

9525 cksum — write file checksums and sizes

9526 **SYNOPSIS**9527 cksum [*file* ...]9528 **DESCRIPTION**

9529 The *cksum* utility shall calculate and write to standard output a cyclic redundancy check (CRC)  
 9530 for each input file, and also write to standard output the number of octets in each file. The CRC  
 9531 used is based on the polynomial used for CRC error checking in the ISO/IEC 8802-3:1996  
 9532 standard (Ethernet).

9533 The encoding for the CRC checksum is defined by the generating polynomial:

$$9534 G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$$

9535 Mathematically, the CRC value corresponding to a given file shall be defined by the following  
 9536 procedure:

- 9537 1. The *n* bits to be evaluated are considered to be the coefficients of a mod 2 polynomial *M(x)*  
 9538 of degree *n*−1. These *n* bits are the bits from the file, with the most significant bit being the  
 9539 most significant bit of the first octet of the file and the last bit being the least significant bit  
 9540 of the last octet, padded with zero bits (if necessary) to achieve an integral number of  
 9541 octets, followed by one or more octets representing the length of the file as a binary value,  
 9542 least significant octet first. The smallest number of octets capable of representing this  
 9543 integer shall be used.
- 9544 2. *M(x)* is multiplied by  $x^{32}$  (that is, shifted left 32 bits) and divided by *G(x)* using mod 2  
 9545 division, producing a remainder *R(x)* of degree ≤ 31.
- 9546 3. The coefficients of *R(x)* are considered to be a 32-bit sequence.
- 9547 4. The bit sequence is complemented and the result is the CRC.

9548 **OPTIONS**

9549 None.

9550 **OPERANDS**

9551 The following operand shall be supported:

9552 *file* A pathname of a file to be checked. If no *file* operands are specified, the standard |  
 9553 input shall be used. |

9554 **STDIN**

9555 The standard input shall be used only if no *file* operands are specified. See the INPUT FILES |  
 9556 section. |

9557 **INPUT FILES**

9558 The input files can be any file type.

9559 **ENVIRONMENT VARIABLES**9560 The following environment variables shall affect the execution of *cksum*:

9561 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 9562 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
 9563 Internationalization Variables for the precedence of internationalization variables  
 9564 used to determine the values of locale categories.)

9565 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 9566 internationalization variables.

9567            *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 9568 characters (for example, single-byte as opposed to multi-byte characters in  
 9569 arguments).

9570            *LC\_MESSAGES*  
 9571                    Determine the locale that should be used to affect the format and contents of  
 9572 diagnostic messages written to standard error.

9573 *XSI*            *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

9574 **ASYNCHRONOUS EVENTS**  
 9575            Default.

9576 **STDOUT**  
 9577            For each file processed successfully, the *cksum* utility shall write in the following format:  
 9578            "%u %d %s\n", <checksum>, <# of octets>, <pathname>  
 9579            If no *file* operand was specified, the pathname and its leading <space> shall be omitted.

9580 **STDERR**  
 9581            The standard error shall be used only for diagnostic messages.

9582 **OUTPUT FILES**  
 9583            None.

9584 **EXTENDED DESCRIPTION**  
 9585            None.

9586 **EXIT STATUS**  
 9587            The following exit values shall be returned:  
 9588            0 All files were processed successfully.  
 9589            >0 An error occurred.

9590 **CONSEQUENCES OF ERRORS**  
 9591            Default.

9592 **APPLICATION USAGE**  
 9593            The *cksum* utility is typically used to quickly compare a suspect file against a trusted version of  
 9594 the same, such as to ensure that files transmitted over noisy media arrive intact. However, this  
 9595 comparison cannot be considered cryptographically secure. The chances of a damaged file  
 9596 producing the same CRC as the original are small; deliberate deception is difficult, but probably  
 9597 not impossible.

9598            Although input files to *cksum* can be any type, the results need not be what would be expected  
 9599 on character special device files or on file types not described by the System Interfaces volume of  
 9600 IEEE Std 1003.1-200x. Since this volume of IEEE Std 1003.1-200x does not specify the block size  
 9601 used when doing input, checksums of character special files need not process all of the data in  
 9602 those files.

9603            The algorithm is expressed in terms of a bitstream divided into octets. If a file is transmitted  
 9604 between two systems and undergoes any data transformation (such as changing little-endian  
 9605 byte ordering to big-endian), identical CRC values cannot be expected. Implementations  
 9606 performing such transformations may extend *cksum* to handle such situations.

9607 **EXAMPLES**  
 9608       None.

9609 **RATIONALE**

9610       The following C-language program can be used as a model to describe the algorithm. It assumes  
 9611       that a **char** is one octet. It also assumes that the entire file is available for one pass through the  
 9612       function. This was done for simplicity in demonstrating the algorithm, rather than as an  
 9613       implementation model.

```

9614       static unsigned long crctab[] = {
9615       0x00000000,
9616       0x04c11db7, 0x09823b6e, 0x0d4326d9, 0x130476dc, 0x17c56b6b,
9617       0x1a864db2, 0x1e475005, 0x2608edb8, 0x22c9f00f, 0x2f8ad6d6,
9618       0x2b4bcb61, 0x350c9b64, 0x31cd86d3, 0x3c8ea00a, 0x384fbd8d,
9619       0x4c11db70, 0x48d0c6c7, 0x4593e01e, 0x4152fda9, 0x5f15adac,
9620       0x5bd4b01b, 0x569796c2, 0x52568b75, 0x6a1936c8, 0x6ed82b7f,
9621       0x639b0da6, 0x675a1011, 0x791d4014, 0x7ddc5da3, 0x709f7b7a,
9622       0x745e66cd, 0x9823b6e0, 0x9ce2ab57, 0x91a18d8e, 0x95609039,
9623       0x8b27c03c, 0x8fe6dd8b, 0x82a5fb52, 0x8664e6e5, 0xbe2b5b58,
9624       0xbaea46ef, 0xb7a96036, 0xb3687d81, 0xad2f2d84, 0xa9ee3033,
9625       0xa4ad16ea, 0xa06c0b5d, 0xd4326d90, 0xd0f37027, 0xddb056fe,
9626       0xd9714b49, 0xc7361b4c, 0xc3f706fb, 0xceb42022, 0xca753d95,
9627       0xf23a8028, 0xf6fb9d9f, 0xfb8bb46, 0xff79a6f1, 0xe13ef6f4,
9628       0xe5ffeb43, 0xe8bccd9a, 0xec7dd02d, 0x34867077, 0x30476dc0,
9629       0x3d044b19, 0x39c556ae, 0x278206ab, 0x23431b1c, 0x2e003dc5,
9630       0x2ac12072, 0x128e9dcf, 0x164f8078, 0x1b0ca6a1, 0x1fcd8bb16,
9631       0x018aeb13, 0x054bf6a4, 0x0808d07d, 0x0cc9cdca, 0x7897ab07,
9632       0x7c56b6b0, 0x71159069, 0x75d48dde, 0x6b93ddd8, 0x6f52c06c,
9633       0x6211e6b5, 0x66d0fb02, 0x5e9f46bf, 0x5a5e5b08, 0x571d7dd1,
9634       0x53dc6066, 0x4d9b3063, 0x495a2dd4, 0x44190b0d, 0x40d816ba,
9635       0xaca5c697, 0xa864db20, 0xa527fdf9, 0xa1e6e04e, 0xbf1b04b,
9636       0xbb60adfc, 0xb6238b25, 0xb2e29692, 0x8aad2b2f, 0x8e6c3698,
9637       0x832f1041, 0x87ee0df6, 0x99a95df3, 0x9d684044, 0x902b669d,
9638       0x94ea7b2a, 0xe0b41de7, 0xe4750050, 0xe9362689, 0xedf73b3e,
9639       0xf3b06b3b, 0xf771768c, 0xfa325055, 0xfef34de2, 0xc6bcf05f,
9640       0xc27dede8, 0xcf3ecb31, 0xcbffd686, 0xd5b88683, 0xd1799b34,
9641       0xdc3abded, 0xd8fba05a, 0x690ce0ee, 0x6dcdfd59, 0x608edb80,
9642       0x644fc637, 0x7a089632, 0x7ec98b85, 0x738aad5c, 0x774bb0eb,
9643       0x4f040d56, 0x4bc510e1, 0x46863638, 0x42472b8f, 0x5c007b8a,
9644       0x58c1663d, 0x558240e4, 0x51435d53, 0x251d3b9e, 0x21dc2629,
9645       0x2c9f00f0, 0x285e1d47, 0x36194d42, 0x32d850f5, 0x3f9b762c,
9646       0x3b5a6b9b, 0x0315d626, 0x07d4cb91, 0x0a97ed48, 0x0e56f0ff,
9647       0x1011a0fa, 0x14d0bd4d, 0x19939b94, 0x1d528623, 0xf12f560e,
9648       0xf5ee4bb9, 0xf8ad6d60, 0xfc6c70d7, 0xe22b20d2, 0xe6ea3d65,
9649       0xeba91bbc, 0xef68060b, 0xd727bbb6, 0xd3e6a601, 0xdea580d8,
9650       0xda649d6f, 0xc423cd6a, 0xc0e2d0dd, 0xcda1f604, 0xc960ebb3,
9651       0xbd3e8d7e, 0xb9ff90c9, 0xb4bcb610, 0xb07daba7, 0xae3afba2,
9652       0xaafb615, 0xa7b8c0cc, 0xa379dd7b, 0x9b3660c6, 0x9ff77d71,
9653       0x92b45ba8, 0x9675461f, 0x8832161a, 0x8cf30bad, 0x81b02d74,
9654       0x857130c3, 0x5d8a9099, 0x594b8d2e, 0x5408abf7, 0x50c9b640,
9655       0x4e8ee645, 0x4a4ffb2, 0x470cdd2b, 0x43cdc09c, 0x7b827d21,
9656       0x7f436096, 0x7200464f, 0x76c15bf8, 0x68860bfd, 0x6c47164a,
9657       0x61043093, 0x65c52d24, 0x119b4be9, 0x155a565e, 0x18197087,

```

```

9658     0x1cd86d30, 0x029f3d35, 0x065e2082, 0x0b1d065b, 0x0fdc1bec,
9659     0x3793a651, 0x3352bbe6, 0x3e119d3f, 0x3ad08088, 0x2497d08d,
9660     0x2056cd3a, 0x2d15ebe3, 0x29d4f654, 0xc5a92679, 0xc1683bce,
9661     0xcc2b1d17, 0xc8ea00a0, 0xd6ad50a5, 0xd26c4d12, 0xdf2f6bcb,
9662     0xdbee767c, 0xe3a1cbc1, 0xe760d676, 0xea23f0af, 0xee2ed18,
9663     0xf0a5bd1d, 0xf464a0aa, 0xf9278673, 0xfde69bc4, 0x89b8fd09,
9664     0x8d79e0be, 0x803ac667, 0x84fbdbd0, 0x9abc8bd5, 0x9e7d9662,
9665     0x933eb0bb, 0x97ffad0c, 0xafb010b1, 0xab710d06, 0xa6322bdf,
9666     0xa2f33668, 0xbcb4666d, 0xb8757bda, 0xb5365d03, 0xb1f740b4
9667     };

9668     unsigned long memcrc(const unsigned char *b, size_t n)
9669     {
9670     /* Input arguments:
9671     *  const char*   b == byte sequence to checksum
9672     *  size_t       n == length of sequence
9673     */

9674         register unsigned   i, c, s = 0;

9675         for (i = n; i > 0; --i) {
9676             c = (unsigned)(*b++);
9677             s = (s << 8) ^ crctab[(s >> 24) ^ c];
9678         }

9679         /* Extend with the length of the string. */
9680         while (n != 0) {
9681             c = n & 0377;
9682             n >>= 8;
9683             s = (s << 8) ^ crctab[(s >> 24) ^ c];
9684         }

9685         return ~s;
9686     }

```

9687 The historical practice of writing the number of “blocks” has been changed to writing the  
9688 number of octets, since the latter is not only more useful, but also since historical  
9689 implementations have not been consistent in defining what a “block” meant. Octets are used  
9690 instead of bytes because bytes can differ in size between systems.

9691 The algorithm used was selected to increase the operational robustness of *cksum*. Neither the  
9692 System V nor BSD *sum* algorithm was selected. Since each of these was different and each was  
9693 the default behavior on those systems, no realistic compromise was available if either were  
9694 selected—some set of historical applications would break. Therefore, the name was changed to  
9695 *cksum*. Although the historical *sum* commands will probably continue to be provided for many  
9696 years, programs designed for portability across systems should use the new name.

9697 The algorithm selected is based on that used by the ISO/IEC 8802-3:1996 standard (Ethernet) for  
9698 the frame check sequence field. The algorithm used does not match the technical definition of a  
9699 *checksum*; the term is used for historical reasons. The length of the file is included in the CRC  
9700 calculation because this parallels inclusion of a length field by Ethernet in its CRC, but also  
9701 because it guards against inadvertent collisions between files that begin with different series of  
9702 zero octets. The chance that two different files produce identical CRCs is much greater when  
9703 their lengths are not considered. Keeping the length and the checksum of the file itself separate  
9704 would yield a slightly more robust algorithm, but historical usage has always been that a single  
9705 number (the checksum as printed) represents the signature of the file. It was decided that

9706 historical usage was the more important consideration.

9707 Early proposals contained modifications to the Ethernet algorithm that involved extracting table  
9708 values whenever an intermediate result became zero. This was demonstrated to be less robust  
9709 than the current method and mathematically difficult to describe or justify.

9710 The calculation used is identical to that given in pseudo-code in the referenced Sarwate article.  
9711 The pseudo-code rendition is:

```
9712 X <- 0; Y <- 0;
9713 for i <- m -1 step -1 until 0 do
9714     begin
9715         T <- X(1) ^ A[i];
9716         X(1) <- X(0); X(0) <- Y(1); Y(1) <- Y(0); Y(0) <- 0;
9717         comment: f[T] and f'[T] denote the T-th words in the
9718                 table f and f' ;
9719         X <- X ^ f[T]; Y <- Y ^ f'[T];
9720     end
```

9721 The pseudo-code is reproduced exactly as given; however, note that in the case of *cksum*, **A[i]**  
9722 represents a byte of the file, the words **X** and **Y** are treated as a single 32-bit value, and the tables  
9723 **f** and **f'** are a single table containing 32-bit values.

9724 The referenced Sarwate article also discusses generating the table.

9725 **FUTURE DIRECTIONS**

9726 None.

9727 **SEE ALSO**

9728 None.

9729 **CHANGE HISTORY**

9730 First released in Issue 4.

9731 **NAME**9732           *cmp* — compare two files9733 **SYNOPSIS**9734           *cmp* [ *-l* | *-s* ] *file1 file2*9735 **DESCRIPTION**

9736           The *cmp* utility shall compare two files. The *cmp* utility shall write no output if the files are the  
 9737           same. Under default options, if they differ, it shall write to standard output the byte and line  
 9738           number at which the first difference occurred. Bytes and lines shall be numbered beginning with  
 9739           1.

9740 **OPTIONS**

9741           The *cmp* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section  
 9742           12.2, Utility Syntax Guidelines.

9743           The following options shall be supported:

9744           **-l**           (Lowercase ell.) Write the byte number (decimal) and the differing bytes (octal) for  
 9745           each difference.

9746           **-s**           Write nothing for differing files; return exit status only.

9747 **OPERANDS**

9748           The following operands shall be supported:

9749           *file1*        A pathname of the first file to be compared. If *file1* is '-', the standard input shall  
 9750           be used.

9751           *file2*        A pathname of the second file to be compared. If *file2* is '-', the standard input  
 9752           shall be used.

9753           If both *file1* and *file2* refer to standard input or refer to the same FIFO special, block special, or  
 9754           character special file, the results are undefined.

9755 **STDIN**

9756           The standard input shall be used only if the *file1* or *file2* operand refers to standard input. See the  
 9757           INPUT FILES section.

9758 **INPUT FILES**

9759           The input files can be any file type.

9760 **ENVIRONMENT VARIABLES**

9761           The following environment variables shall affect the execution of *cmp*:

9762           **LANG**        Provide a default value for the internationalization variables that are unset or null.  
 9763           (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
 9764           Internationalization Variables for the precedence of internationalization variables  
 9765           used to determine the values of locale categories.)

9766           **LC\_ALL**     If set to a non-empty string value, override the values of all the other  
 9767           internationalization variables.

9768           **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 9769           characters (for example, single-byte as opposed to multi-byte characters in  
 9770           arguments).

9771           **LC\_MESSAGES**

9772           Determine the locale that should be used to affect the format and contents of  
 9773           diagnostic messages written to standard error and informative messages written to  
 9774           standard output.

9775 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

9776 **ASYNCHRONOUS EVENTS**

9777 Default.

9778 **STDOUT**

9779 In the POSIX locale, results of the comparison shall be written to standard output. When no  
9780 options are used, the format shall be:

9781 "%s %s differ: char %d, line %d\n", *file1*, *file2*,  
9782 <byte number>, <line number>

9783 When the **-I** option is used, the format shall be:

9784 "%d %o %o\n", <byte number>, <differing byte>,  
9785 <differing byte>

9786 for each byte that differs. The first <differing byte> number is from *file1* while the second is from  
9787 *file2*. In both cases, <byte number> shall be relative to the beginning of the file, beginning with 1.

9788 No output shall be written to standard output when the **-s** option is used.

9789 **STDERR**

9790 The standard error shall be used only for diagnostic messages. If *file1* and *file2* are identical for  
9791 the entire length of the shorter file, in the POSIX locale the following diagnostic message shall be  
9792 written, unless the **-s** option is specified:

9793 "cmp: EOF on %s%s\n", <name of shorter file>, <additional info>

9794 The <additional info> field shall either be null or a string that starts with a <blank> and contains  
9795 no <newline>s. Some implementations report on the number of lines in this case.

9796 **OUTPUT FILES**

9797 None.

9798 **EXTENDED DESCRIPTION**

9799 None.

9800 **EXIT STATUS**

9801 The following exit values shall be returned:

9802 0 The files are identical.

9803 1 The files are different; this includes the case where one file is identical to the first part of the  
9804 other.

9805 >1 An error occurred.

9806 **CONSEQUENCES OF ERRORS**

9807 Default.

9808 **APPLICATION USAGE**

9809 Although input files to *cmp* can be any type, the results might not be what would be expected on  
9810 character special device files or on file types not described by the System Interfaces volume of  
9811 IEEE Std 1003.1-200x. Since this volume of IEEE Std 1003.1-200x does not specify the block size  
9812 used when doing input, comparisons of character special files need not compare all of the data  
9813 in those files.

9814 For files which are not text files, line numbers simply reflect the presence of a <newline>,  
9815 without any implication that the file is organized into lines.

9816 **EXAMPLES**

9817       None.

9818 **RATIONALE**

9819       The global language in Section 1.11 (on page 2221) indicates that using two mutually-exclusive  
9820       options together produces unspecified results. Some System V implementations consider the  
9821       option usage:

9822       cmp -l -s ...

9823       to be an error. They also treat:

9824       cmp -s -l ...

9825       as if no options were specified. Both of these behaviors are considered bugs, but are allowed.

9826       The word **char** in the standard output format comes from historical usage, even though it is  
9827       actually a byte number. When *cmp* is supported in other locales, implementations are  
9828       encouraged to use the word *byte* or its equivalent in another language. Users should not  
9829       interpret this difference to indicate that the functionality of the utility changed between locales.

9830       Some implementations report on the number of lines in the identical-but-shorter file case. This is |  
9831       allowed by the inclusion of the *<additional info>* fields in the output format. The restriction on |  
9832       having a leading *<blank>* and no *<newline>*s is to make parsing for the filename easier. It is |  
9833       recognized that some filenames containing white-space characters make parsing difficult  
9834       anyway, but the restriction does aid programs used on systems where the names are  
9835       predominantly well behaved.

9836 **FUTURE DIRECTIONS**

9837       None.

9838 **SEE ALSO**9839       *comm*, *diff*9840 **CHANGE HISTORY**

9841       First released in Issue 2.

9842 **NAME**

9843           comm — select or reject lines common to two files

9844 **SYNOPSIS**9845           comm [-123] *file1 file2*9846 **DESCRIPTION**9847           The *comm* utility shall read *file1* and *file2*, which should be ordered in the current collating  
9848           sequence, and produce three text columns as output: lines only in *file1*, lines only in *file2*, and  
9849           lines in both files.9850           If the lines in both files are not ordered according to the collating sequence of the current locale,  
9851           the results are unspecified.9852 **OPTIONS**9853           The *comm* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section  
9854           12.2, Utility Syntax Guidelines.

9855           The following options shall be supported:

9856           -1           Suppress the output column of lines unique to *file1*.9857           -2           Suppress the output column of lines unique to *file2*.9858           -3           Suppress the output column of lines duplicated in *file1* and *file2*.9859 **OPERANDS**

9860           The following operands shall be supported:

9861           *file1*        A pathname of the first file to be compared. If *file1* is '-', the standard input shall  
9862                           be used. |9863           *file2*        A pathname of the second file to be compared. If *file2* is '-', the standard input  
9864                           shall be used. |9865           If both *file1* and *file2* refer to standard input or to the same FIFO special, block special, or  
9866           character special file, the results are undefined.9867 **STDIN**9868           The standard input shall be used only if one of the *file1* or *file2* operands refers to standard input.  
9869           See the INPUT FILES section.9870 **INPUT FILES**

9871           The input files shall be text files.

9872 **ENVIRONMENT VARIABLES**9873           The following environment variables shall affect the execution of *comm*:9874           *LANG*        Provide a default value for the internationalization variables that are unset or null.  
9875                           (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
9876                           Internationalization Variables for the precedence of internationalization variables  
9877                           used to determine the values of locale categories.)9878           *LC\_ALL*        If set to a non-empty string value, override the values of all the other  
9879                           internationalization variables.9880           *LC\_COLLATE*9881                           Determine the locale for the collating sequence *comm* expects to have been used  
9882                           when the input files were sorted.9883           *LC\_CTYPE*    Determine the locale for the interpretation of sequences of bytes of text data as  
9884                           characters (for example, single-byte as opposed to multi-byte characters in

9885 arguments and input files).

9886 **LC\_MESSAGES**

9887 Determine the locale that should be used to affect the format and contents of

9888 diagnostic messages written to standard error.

9889 XSI **NLSPATH** Determine the location of message catalogs for the processing of **LC\_MESSAGES**.

9890 **ASYNCHRONOUS EVENTS**

9891 Default.

9892 **STDOUT**

9893 The *comm* utility shall produce output depending on the options selected. If the **-1**, **-2**, and **-3**

9894 options are all selected, *comm* shall write nothing to standard output.

9895 If the **-1** option is not selected, lines contained only in *file1* shall be written using the format:

9896 "%s\n", <line in file1>

9897 If the **-2** option is not selected, lines contained only in *file2* are written using the format:

9898 "%s%s\n", <lead>, <line in file2>

9899 where the string <lead> is as follows:

9900 <tab> The **-1** option is not selected.

9901 null string The **-1** option is selected.

9902 If the **-3** option is not selected, lines contained in both files shall be written using the format:

9903 "%s%s\n", <lead>, <line in both>

9904 where the string <lead> is as follows:

9905 <tab><tab> Neither the **-1** nor the **-2** option is selected.

9906 <tab> Exactly one of the **-1** and **-2** options is selected.

9907 null string Both the **-1** and **-2** options are selected.

9908 If the input files were ordered according to the collating sequence of the current locale, the lines

9909 written shall be in the collating sequence of the original lines.

9910 **STDERR**

9911 The standard error shall be used only for diagnostic messages.

9912 **OUTPUT FILES**

9913 None.

9914 **EXTENDED DESCRIPTION**

9915 None.

9916 **EXIT STATUS**

9917 The following exit values shall be returned:

9918 0 All input files were successfully output as specified.

9919 >0 An error occurred.

9920 **CONSEQUENCES OF ERRORS**

9921 Default.

9922 **APPLICATION USAGE**

9923 If the input files are not properly presorted, the output of *comm* might not be useful.

9924 **EXAMPLES**

9925 If a file named **xcu** contains a sorted list of the utilities in this volume of IEEE Std 1003.1-200x, a  
9926 file named **xpg3** contains a sorted list of the utilities specified in the X/Open Portability Guide,  
9927 Issue 3, and a file named **svid89** contains a sorted list of the utilities in the System V Interface  
9928 Definition Third Edition:

9929 `comm -23 xcu xpg3 | comm -23 - svid89`

9930 would print a list of utilities in this volume of IEEE Std 1003.1-200x not specified by either of the  
9931 other documents:

9932 `comm -12 xcu xpg3 | comm -12 - svid89`

9933 would print a list of utilities specified by all three documents, and:

9934 `comm -12 xpg3 svid89 | comm -23 - xcu`

9935 would print a list of utilities specified by both XPG3 and the SVID, but not specified in this  
9936 volume of IEEE Std 1003.1-200x.

9937 **RATIONALE**

9938 None.

9939 **FUTURE DIRECTIONS**

9940 None.

9941 **SEE ALSO**

9942 *cmp, diff, sort, uniq*

9943 **CHANGE HISTORY**

9944 First released in Issue 2.

9945 **Issue 6**

9946 The normative text is reworded to avoid use of the term “must” for application requirements.

9947 **NAME**9948 `command` — execute a simple command9949 **SYNOPSIS**9950 `command [-p] command_name [argument ...]`9951 UP `command [ -v | -V ] command_name`

9952

9953 **DESCRIPTION**9954 The *command* utility shall cause the shell to treat the arguments as a simple command, suppressing the shell function lookup that is described in Section 2.9.1.1 (on page 2249), item 1b.9956 If the *command\_name* is the same as the name of one of the special built-in utilities, the special properties in the enumerated list at the beginning of Section 2.14 (on page 2266) shall not occur. In every other respect, if *command\_name* is not the name of a function, the effect of *command* (with no options) shall be the same as omitting *command*.9960 On systems supporting the User Portability Utilities option, the *command* utility also shall provide information concerning how a command name is interpreted by the shell; see `-v` and `-V`.9963 **OPTIONS**9964 The *command* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2, Utility Syntax Guidelines.

9966 The following options shall be supported:

9967 `-p` Perform the command search using a default value for *PATH* that is guaranteed to find all of the standard utilities.9969 `-v` (On systems supporting the User Portability Utilities option.) Write a string to standard output that indicates the pathname or command that will be used by the shell, in the current shell execution environment (see Section 2.12 (on page 2263)), to invoke *command\_name*, but do not invoke *command\_name*.9973 • Utilities, regular built-in utilities, *command\_names* including a slash character, and any implementation-defined functions that are found using the *PATH* variable (as described in Section 2.9.1.1 (on page 2249)), shall be written as absolute pathnames.9977 • Shell functions, special built-in utilities, regular built-in utilities not associated with a *PATH* search, and shell reserved words shall be written as just their names.

9980 • An alias shall be written as a command line that represents its alias definition.

9981 • Otherwise, no output shall be written and the exit status shall reflect that the name was not found.

9983 `-V` (On systems supporting the User Portability Utilities option.) Write a string to standard output that indicates how the name given in the *command\_name* operand will be interpreted by the shell, in the current shell execution environment (see Section 2.12 (on page 2263)), but do not invoke *command\_name*. Although the format of this string is unspecified, it shall indicate in which of the following categories *command\_name* falls and shall include the information stated:9989 • Utilities, regular built-in utilities, and any implementation-defined functions that are found using the *PATH* variable (as described in Section 2.9.1.1 (on page 2249)), shall be identified as such and include the absolute pathname in the

- 9992 string.
- 9993 • Other shell functions shall be identified as functions.
- 9994 • Aliases shall be identified as aliases and their definitions included in the string.
- 9995 • Special built-in utilities shall be identified as special built-in utilities.
- 9996 • Regular built-in utilities not associated with a *PATH* search shall be identified
- 9997 as regular built-in utilities. (The term “regular” need not be used.)
- 9998 • Shell reserved words shall be identified as reserved words.
- 9999 **OPERANDS**
- 10000 The following operands shall be supported:
- 10001 *argument* One of the strings treated as an argument to *command\_name*.
- 10002 *command\_name*
- 10003 The name of a utility or a special built-in utility.
- 10004 **STDIN**
- 10005 Not used.
- 10006 **INPUT FILES**
- 10007 None.
- 10008 **ENVIRONMENT VARIABLES**
- 10009 The following environment variables shall affect the execution of *command*:
- 10010 *LANG* Provide a default value for the internationalization variables that are unset or null.
- 10011 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
- 10012 Internationalization Variables for the precedence of internationalization variables
- 10013 used to determine the values of locale categories.)
- 10014 *LC\_ALL* If set to a non-empty string value, override the values of all the other
- 10015 internationalization variables.
- 10016 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
- 10017 characters (for example, single-byte as opposed to multi-byte characters in
- 10018 arguments).
- 10019 *LC\_MESSAGES*
- 10020 Determine the locale that should be used to affect the format and contents of
- 10021 diagnostic messages written to standard error and informative messages written to
- 10022 standard output.
- 10023 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 10024 *PATH* Determine the search path used during the command search described in Section
- 10025 2.9.1.1 (on page 2249), except as described under the **-p** option.
- 10026 **ASYNCHRONOUS EVENTS**
- 10027 Default.
- 10028 **STDOUT**
- 10029 When the **-v** option is specified, standard output shall be formatted as:
- 10030 "%s\n", *<pathname or command>*
- 10031 When the **-V** option is specified, standard output shall be formatted as:

10032 "%s\n", <unspecified>

### 10033 **STDERR**

10034 The standard error shall be used only for diagnostic messages.

### 10035 **OUTPUT FILES**

10036 None.

### 10037 **EXTENDED DESCRIPTION**

10038 None.

### 10039 **EXIT STATUS**

10040 When the `-v` or `-V` options are specified, the following exit values shall be returned:

10041 0 Successful completion.

10042 >0 The *command\_name* could not be found or an error occurred.

10043 Otherwise, the following exit values shall be returned:

10044 126 The utility specified by *command\_name* was found but could not be invoked.

10045 127 An error occurred in the *command* utility or the utility specified by *command\_name* could not  
10046 be found.

10047 Otherwise, the exit status of *command* shall be that of the simple command specified by the  
10048 arguments to *command*.

### 10049 **CONSEQUENCES OF ERRORS**

10050 Default.

### 10051 **APPLICATION USAGE**

10052 The order for command search allows functions to override regular built-ins and path searches.  
10053 This utility is necessary to allow functions that have the same name as a utility to call the utility  
10054 (instead of a recursive call to the function).

10055 The system default path is available using *getconf*; however, since *getconf* may need to have the  
10056 *PATH* set up before it can be called itself, the following can be used:

```
10057 command -p getconf _CS_PATH
```

10058 There are some advantages to suppressing the special characteristics of special built-ins on  
10059 occasion. For example:

```
10060 command exec > unwritable-file
```

10061 does not cause a non-interactive script to abort, so that the output status can be checked by the  
10062 script.

10063 The *command*, *env*, *nohup*, *time*, and *xargs* utilities have been specified to use exit code 127 if an  
10064 error occurs so that applications can distinguish “failure to find a utility” from “invoked utility  
10065 exited with an error indication”. The value 127 was chosen because it is not commonly used for  
10066 other meanings; most utilities use small values for “normal error conditions” and the values  
10067 above 128 can be confused with termination due to receipt of a signal. The value 126 was chosen  
10068 in a similar manner to indicate that the utility could be found, but not invoked. Some scripts  
10069 produce meaningful error messages differentiating the 126 and 127 cases. The distinction  
10070 between exit codes 126 and 127 is based on KornShell practice that uses 127 when all attempts to  
10071 *exec* the utility fail with [ENOENT], and uses 126 when any attempt to *exec* the utility fails for  
10072 any other reason.

10073 Since the `-v` and `-V` options of *command* produce output in relation to the current shell execution  
10074 environment, *command* is generally provided as a shell regular built-in. If it is called in a subshell

10075 or separate utility execution environment, such as one of the following:

```
10076 (PATH=foo command -v)
10077 nohup command -v
```

10078 it does not necessarily produce correct results. For example, when called with *nohup* or an *exec*  
10079 function, in a separate utility execution environment, most implementations are not able to  
10080 identify aliases, functions, or special built-ins.

10081 Two types of regular built-ins could be encountered on a system and these are described  
10082 separately by *command*. The description of command search in Section 2.9.1.1 (on page 2249)  
10083 allows for a standard utility to be implemented as a regular built-in as long as it is found in the  
10084 appropriate place in a *PATH* search. So, for example, *command -v true* might yield */bin/true* or  
10085 some similar pathname. Other implementation-defined utilities that are not defined by this  
10086 volume of IEEE Std 1003.1-200x might exist only as built-ins and have no pathname associated  
10087 with them. These produce output identified as (regular) built-ins. Applications encountering  
10088 these are not able to count on *execing* them, using them with *nohup*, overriding them with a  
10089 different *PATH*, and so on.

#### 10090 EXAMPLES

10091 1. Make a version of *cd* that always prints out the new working directory exactly once:

```
10092 cd() {
10093     command cd "$@" >/dev/null
10094     pwd
10095 }
```

10096 2. Start off a “secure shell script” in which the script avoids being spoofed by its parent:

```
10097 IFS='
10098 '
10099 # The preceding value should be <space><tab><newline>.
10100 # Set IFS to its default value.

10101 \unalias -a
10102 # Unset all possible aliases.
10103 # Note that unalias is escaped to prevent an alias
10104 # being used for unalias.

10105 unset -f command
10106 # Ensure command is not a user function.

10107 PATH="$(command -p getconf _CS_PATH):$PATH"
10108 # Put on a reliable PATH prefix.

10109 # ...
```

10110 At this point, given correct permissions on the directories called by *PATH*, the script has  
10111 the ability to ensure that any utility it calls is the intended one. It is being very cautious  
10112 because it assumes that implementation extensions may be present that would allow user  
10113 functions to exist when it is invoked; this capability is not specified by this volume of  
10114 IEEE Std 1003.1-200x, but it is not prohibited as an extension. For example, the *ENV*  
10115 variable precedes the invocation of the script with a user start-up script. Such a script  
10116 could define functions to spoof the application.

10117 **RATIONALE**

10118 Since *command* is a regular built-in utility it is always found prior to the *PATH* search.

10119 There is nothing in the description of *command* that implies the command line is parsed any  
10120 differently from that of any other simple command. For example:

10121 `command a | b ; c`

10122 is not parsed in any special way that causes ' | ' or ' ; ' to be treated other than a pipe operator  
10123 or semicolon or that prevents function lookup on **b** or **c**.

10124 The *command* utility is somewhat similar to the Eighth Edition shell *builtin* command, but since  
10125 *command* also goes to the file system to search for utilities, the name *builtin* would not be  
10126 intuitive.

10127 The *command* utility is most likely to be provided as a regular built-in. It is not listed as a special  
10128 built-in for the following reasons:

10129 • The removal of exportable functions made the special precedence of a special built-in  
10130 unnecessary.

10131 • A special built-in has special properties (see Section 2.14 (on page 2266)) that were  
10132 inappropriate for invoking other utilities. For example, two commands such as:

10133 `date > unwritable-file`

10134 `command date > unwritable-file`

10135 would have entirely different results; in a non-interactive script, the former would continue  
10136 to execute the next command, the latter would abort. Introducing this semantic difference  
10137 along with suppressing functions was seen to be non-intuitive.

10138 The **-p** option is present because it is useful to be able to ensure a safe path search that finds all  
10139 the POSIX Shell and Utilities standard utilities. This search might not be identical to the one that  
10140 occurs through one of the POSIX System Interfaces *exec* functions when *PATH* is unset. At the  
10141 very least, this feature is required to allow the script to access the correct version of *getconf* so  
10142 that the value of the default path can be accurately retrieved.

10143 The *command* **-v** and **-V** options were added to satisfy requirements from users that are  
10144 currently accomplished by three different historical utilities: *type* in the System V shell, *whence* in  
10145 the KornShell, and *which* in the C shell. Since there is no historical agreement on how and what  
10146 to accomplish here, the POSIX *command* utility was enhanced and the historical utilities were left  
10147 unmodified. The C shell *which* merely conducts a path search. The KornShell *whence* is more  
10148 elaborate—in addition to the categories required by POSIX, it also reports on tracked aliases,  
10149 exported aliases, and undefined functions.

10150 The output format of **-V** was left mostly unspecified because human users are its only audience.  
10151 Applications should not be written to care about this information; they can use the output of **-v**  
10152 to differentiate between various types of commands, but the additional information that may be  
10153 emitted by the more verbose **-V** is not needed and should not be arbitrarily constrained in its  
10154 verbosity or localization for application parsing reasons.

10155 **FUTURE DIRECTIONS**

10156 None.

10157 **SEE ALSO**

10158 *sh*, *type*

10159 **CHANGE HISTORY**

10160 First released in Issue 4.

10161 **NAME**10162 `compress` — compress data10163 **SYNOPSIS**10164 xSI `compress [-fv][-b bits][file ...]`10165 `compress [-cfv][-b bits][file]`

10166

10167 **DESCRIPTION**10168 The `compress` utility shall attempt to reduce the size of the named files by using adaptive  
10169 Lempel-Ziv coding algorithm.10170 **Note:** Lempel-Ziv is US Patent 4464650, issued to William Eastman, Abraham Lempel, Jacob Ziv,  
10171 Martin Cohn on August 7th, 1984, and assigned to Sperry Corporation.10172 Lempel-Ziv-Welch compression is covered by US Patent 4558302, issued to Terry A. Welch on  
10173 December 10th, 1985, and assigned to Sperry Corporation.10174 On systems not supporting adaptive Lempel-Ziv coding algorithm, the input files shall not be  
10175 changed and an error value greater than two shall be returned. Except when the output is to the  
10176 standard output, each file shall be replaced by one with the extension `.Z`. If the invoking process  
10177 has appropriate privileges, the ownership, modes, access time, and modification time of the  
10178 original file are preserved. If appending the `.Z` to the filename would make the name exceed  
10179 `{NAME_MAX}` bytes, the command shall fail. If no files are specified, the standard input shall be  
10180 compressed to the standard output.10181 **OPTIONS**10182 The `compress` utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x,  
10183 Section 12.2, Utility Syntax Guidelines.

10184 The following options shall be supported:

10185 **-b *bits*** Specify the maximum number of bits to use in a code. For a conforming  
10186 application, the *bits* argument shall be:10187  $9 \leq bits \leq 14$ 10188 The implementation may allow *bits* values of greater than 14. The default is 14, 15,  
10189 or 16.10190 **-c** Cause `compress` to write to the standard output; the input file is not changed, and  
10191 no `.Z` files are created.10192 **-f** Force compression of *file*, even if it does not actually reduce the size of the file, or if  
10193 the corresponding *file.Z* file already exists. If the `-f` option is not given, and the  
10194 process is not running in the background, the user is prompted as to whether an  
10195 existing *file.Z* file should be overwritten.10196 **-v** Write the percentage reduction of each file to standard error.10197 **OPERANDS**

10198 The following operand shall be supported:

10199 ***file*** A pathname of a file to be compressed.10200 **STDIN**10201 The standard input shall be used only if no *file* operands are specified, or if a *file* operand is `'-'`.

10202 **INPUT FILES**

10203 If *file* operands are specified, the input files contain the data to be compressed.

10204 **ENVIRONMENT VARIABLES**

10205 The following environment variables shall affect the execution of *compress*:

10206 *LANG* Provide a default value for the internationalization variables that are unset or null.  
10207 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
10208 Internationalization Variables for the precedence of internationalization variables  
10209 used to determine the values of locale categories.)

10210 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
10211 internationalization variables.

10212 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
10213 characters (for example, single-byte as opposed to multi-byte characters in  
10214 arguments).

10215 *LC\_MESSAGES*

10216 Determine the locale that should be used to affect the format and contents of  
10217 diagnostic messages written to standard error.

10218 *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

10219 **ASYNCHRONOUS EVENTS**

10220 Default.

10221 **STDOUT**

10222 If no *file* operands are specified, or if a *file* operand is '-', or if the *-c* option is specified, the  
10223 standard output contains the compressed output.

10224 **STDERR**

10225 The standard error shall be used only for diagnostic and prompt messages and the output from  
10226 *-v*.

10227 **OUTPUT FILES**

10228 The output files shall contain the compressed output. The format of compressed files is  
10229 unspecified and interchange of such files between implementations (including access via  
10230 unspecified file sharing mechanisms) is not required by IEEE Std 1003.1-200x.

10231 **EXTENDED DESCRIPTION**

10232 None.

10233 **EXIT STATUS**

10234 The following exit values shall be returned:

10235 0 Successful completion.

10236 1 An error occurred.

10237 2 One or more files were not compressed because they would have increased in size (and the  
10238 *-f* option was not specified).

10239 >2 An error occurred.

10240 **CONSEQUENCES OF ERRORS**

10241 The input file shall remain unmodified.

10242 **APPLICATION USAGE**

10243           The amount of compression obtained depends on the size of the input, the number of *bits* per  
10244           code, and the distribution of common substrings. Typically, text such as source code or English  
10245           is reduced by 50-60%. Compression is generally much better than that achieved by Huffman  
10246           coding or adaptive Huffman coding (*compact*), and takes less time to compute.

10247           Although *compress* strictly follows the default actions upon receipt of a signal or when an error  
10248           occurs, some unexpected results may occur. In some implementations it is likely that a partially  
10249           compressed file is left in place, alongside its uncompressed input file. Since the general  
10250           operation of *compress* is to delete the uncompressed file only after the *.Z* file has been  
10251           successfully filled, an application should always carefully check the exit status of *compress* before  
10252           arbitrarily deleting files that have like-named neighbors with *.Z* suffixes.

10253           The limit of 14 on the *bits* option-argument is to achieve portability to all systems (within the |  
10254           restrictions imposed by the lack of an explicit published file format). Some implementations |  
10255           based on 16-bit architectures cannot support 15 or 16-bit uncompression. |

10256 **EXAMPLES**

10257           None.

10258 **RATIONALE**

10259           None.

10260 **FUTURE DIRECTIONS**

10261           None.

10262 **SEE ALSO**

10263           *uncompress*, *zcat*

10264 **CHANGE HISTORY**

10265           First released in Issue 4.

10266 **Issue 6**

10267           The normative text is reworded to avoid use of the term “must” for application requirements.

10268           An error case is added for systems not supporting adaptive Lempel-Ziv coding.

## 10269 NAME

10270 cp — copy files

## 10271 SYNOPSIS

10272 cp [-fip] *source\_file target\_file*10273 cp [-fip] *source\_file ... target*10274 cp -R [-H | -L | -P][-fip] *source\_file ... target*10275 OB cp -r [-H | -L | -P][-fip] *source\_file ... target*

## 10276 DESCRIPTION

10277 The first synopsis form is denoted by two operands, neither of which are existing files of type  
 10278 directory. The *cp* utility shall copy the contents of *source\_file* (or, if *source\_file* is a file of type  
 10279 symbolic link, the contents of the file referenced by *source\_file*) to the destination path named by  
 10280 *target\_file*.

10281 The second synopsis form is denoted by two or more operands where the **-R** or **-r** options are  
 10282 not specified and the first synopsis form is not applicable. It shall be an error if any *source\_file* is a  
 10283 file of type directory, if *target* does not exist, or if *target* is a file of a type defined by the System  
 10284 Interfaces volume of IEEE Std 1003.1-200x, but is not a file of type directory. The *cp* utility shall  
 10285 copy the contents of each *source\_file* (or, if *source\_file* is a file of type symbolic link, the contents  
 10286 of the file referenced by *source\_file*) to the destination path named by the concatenation of *target*,  
 10287 a slash character, and the last component of *source\_file*.

10288 The third and fourth synopsis forms are denoted by two or more operands where the **-R** or **-r**  
 10289 options are specified. The *cp* utility shall copy each file in the file hierarchy rooted in each  
 10290 *source\_file* to a destination path named as follows:

- 10291 • If *target* exists and is a file of type directory, the name of the corresponding destination path |
- 10292 for each file in the file hierarchy shall be the concatenation of *target*, a slash character, and the |
- 10293 pathname of the file relative to the directory containing *source\_file*. |
- 10294 • If *target* does not exist and two operands are specified, the name of the corresponding |
- 10295 destination path for *source\_file* shall be *target*; the name of the corresponding destination path |
- 10296 for all other files in the file hierarchy shall be the concatenation of *target*, a slash character, |
- 10297 and the pathname of the file relative to *source\_file*. |

10298 It shall be an error if *target* does not exist and more than two operands are specified, or if *target* |

10299 exists and is a file of a type defined by the System Interfaces volume of IEEE Std 1003.1-200x, but |

10300 is not a file of type directory.

10301 In the following description, the term *dest\_file* refers to the file named by the destination path.  
 10302 The term *source\_file* refers to the file that is being copied, whether specified as an operand or a  
 10303 file in a file hierarchy rooted in a *source\_file* operand. If *source\_file* is a file of type symbolic link:

- 10304 • If neither the **-R** nor **-r** options were specified, *cp* shall take actions based on the type and  
 10305 contents of the file referenced by the symbolic link, and not by the symbolic link itself.
- 10306 • If the **-R** option was specified:
  - 10307 — If none of the options **-H**, **-L**, nor **-P** were specified, it is unspecified which of **-H**, **-L**, or  
 10308 **-P** will be used as a default.
  - 10309 — If the **-H** option was specified, *cp* shall take actions based on the type and contents of the  
 10310 file referenced by any symbolic link specified as a *source\_file* operand.
  - 10311 — If the **-L** option was specified, *cp* shall take actions based on the type and contents of the  
 10312 file referenced by any symbolic link specified as a *source\_file* operand or any symbolic

- 10313 links encountered during traversal of a file hierarchy.
- 10314 — If the **-P** option was specified, *cp* shall copy any symbolic link specified as a *source\_file*  
 10315 operand and any symbolic links encountered during traversal of a file hierarchy, and shall  
 10316 not follow any symbolic links.
- 10317 • If the **-r** option was specified, the behavior is implementation-defined.
- 10318 For each *source\_file*, the following steps shall be taken:
- 10319 1. If *source\_file* references the same file as *dest\_file*, *cp* may write a diagnostic message to  
 10320 standard error; it shall do nothing more with *source\_file* and shall go on to any remaining  
 10321 files.
- 10322 2. If *source\_file* is of type directory, the following steps shall be taken:
- 10323 a. If neither the **-R** or **-r** options were specified, *cp* shall write a diagnostic message to  
 10324 standard error, do nothing more with *source\_file*, and go on to any remaining files.
- 10325 b. If *source\_file* was not specified as an operand and *source\_file* is dot or dot-dot, *cp* shall  
 10326 do nothing more with *source\_file* and go on to any remaining files.
- 10327 c. If *dest\_file* exists and it is a file type not specified by the System Interfaces volume of  
 10328 IEEE Std 1003.1-200x, the behavior is implementation-defined.
- 10329 d. If *dest\_file* exists and it is not of type directory, *cp* shall write a diagnostic message to  
 10330 standard error, do nothing more with *source\_file* or any files below *source\_file* in the  
 10331 file hierarchy, and go on to any remaining files.
- 10332 e. If the directory *dest\_file* does not exist, it shall be created with file permission bits set  
 10333 to the same value as those of *source\_file*, modified by the file creation mask of the  
 10334 user if the **-p** option was not specified, and then bitwise-inclusively OR'ed with  
 10335 S\_IRWXU. If *dest\_file* cannot be created, *cp* shall write a diagnostic message to  
 10336 standard error, do nothing more with *source\_file*, and go on to any remaining files. It  
 10337 is unspecified if *cp* attempts to copy files in the file hierarchy rooted in *source\_file*.
- 10338 f. The files in the directory *source\_file* shall be copied to the directory *dest\_file*, taking |  
 10339 the four steps (1 to 4) listed here with the files as *source\_files*. |
- 10340 g. If *dest\_file* was created, its file permission bits shall be changed (if necessary) to be the  
 10341 same as those of *source\_file*, modified by the file creation mask of the user if the **-p**  
 10342 option was not specified.
- 10343 h. The *cp* utility shall do nothing more with *source\_file* and go on to any remaining files.
- 10344 3. If *source\_file* is of type regular file, the following steps shall be taken:
- 10345 a. If *dest\_file* exists, the following steps shall be taken:
- 10346 i. If the **-i** option is in effect, the *cp* utility shall write a prompt to the standard  
 10347 error and read a line from the standard input. If the response is not affirmative,  
 10348 *cp* shall do nothing more with *source\_file* and go on to any remaining files.
- 10349 ii. A file descriptor for *dest\_file* shall be obtained by performing actions equivalent  
 10350 to the *open()* function defined in the System Interfaces volume of  
 10351 IEEE Std 1003.1-200x called using *dest\_file* as the *path* argument, and the  
 10352 bitwise-inclusive OR of O\_WRONLY and O\_TRUNC as the *oflag* argument.
- 10353 iii. If the attempt to obtain a file descriptor fails and the **-f** option is in effect, *cp*  
 10354 shall attempt to remove the file by performing actions equivalent to the  
 10355 *unlink()* function defined in the System Interfaces volume of

- 10356 IEEE Std 1003.1-200x called using *dest\_file* as the *path* argument. If this attempt  
10357 succeeds, *cp* shall continue with step 3b.
- 10358 b. If *dest\_file* does not exist, a file descriptor shall be obtained by performing actions  
10359 equivalent to the *open()* function defined in the System Interfaces volume of  
10360 IEEE Std 1003.1-200x called using *dest\_file* as the *path* argument, and the bitwise-  
10361 inclusive OR of *O\_WRONLY* and *O\_CREAT* as the *oflag* argument. The file  
10362 permission bits of *source\_file* shall be the *mode* argument.
- 10363 c. If the attempt to obtain a file descriptor fails, *cp* shall write a diagnostic message to  
10364 standard error, do nothing more with *source\_file*, and go on to any remaining files.
- 10365 d. The contents of *source\_file* shall be written to the file descriptor. Any write errors  
10366 shall cause *cp* to write a diagnostic message to standard error and continue to step 3e.
- 10367 e. The file descriptor shall be closed.
- 10368 f. The *cp* utility shall do nothing more with *source\_file*. If a write error occurred in step  
10369 3d, it is unspecified if *cp* continues with any remaining files. If no write error  
10370 occurred in step 3d, *cp* shall go on to any remaining files.
- 10371 4. Otherwise, the following steps shall be taken:
- 10372 a. If the **-r** option was specified, the behavior is implementation-defined.
- 10373 b. If the **-R** option was specified, the following steps shall be taken:
- 10374 i. The *dest\_file* shall be created with the same file type as *source\_file*.
- 10375 ii. If *source\_file* is a file of type FIFO, the file permission bits shall be the same as  
10376 those of *source\_file*, modified by the file creation mask of the user if the **-p**  
10377 option was not specified. Otherwise, the permissions, owner ID, and group ID  
10378 of *dest\_file* are implementation-defined.
- 10379 If this creation fails for any reason, *cp* shall write a diagnostic message to  
10380 standard error, do nothing more with *source\_file*, and go on to any remaining  
10381 files.
- 10382 iii. If *source\_file* is a file of type symbolic link, the pathname contained in *dest\_file*  
10383 shall be the same as the pathname contained in *source\_file*.
- 10384 If this fails for any reason, *cp* shall write a diagnostic message to standard error,  
10385 do nothing more with *source\_file*, and go on to any remaining files.
- 10386 If the implementation provides additional or alternate access control mechanisms (see the Base  
10387 Definitions volume of IEEE Std 1003.1-200x, Section 4.4, File Access Permissions), their effect on  
10388 copies of files is implementation-defined.

#### 10389 OPTIONS

- 10390 The *cp* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2,  
10391 Utility Syntax Guidelines.
- 10392 The following options shall be supported:
- 10393 **-f** If a file descriptor for a destination file cannot be obtained, as described in step  
10394 3.a.ii., attempt to unlink the destination file and proceed.
- 10395 **-H** Take actions based on the type and contents of the file referenced by any symbolic  
10396 link specified as a *source\_file* operand.
- 10397 **-i** Write a prompt to standard error before copying to any existing destination file. If  
10398 the response from the standard input is affirmative, the copy shall be attempted;

- 10399 otherwise, it shall not.
- 10400 **-L** Take actions based on the type and contents of the file referenced by any symbolic  
10401 link specified as a *source\_file* operand or any symbolic links encountered during  
10402 traversal of a file hierarchy. |
- 10403 **-P** Take actions on any symbolic link specified as a *source\_file* operand or any |  
10404 symbolic link encountered during traversal of a file hierarchy. |
- 10405 **-p** Duplicate the following characteristics of each source file in the corresponding  
10406 destination file:
- 10407 1. The time of last data modification and time of last access. If this duplication  
10408 fails for any reason, *cp* shall write a diagnostic message to standard error.
  - 10409 2. The user ID and group ID. If this duplication fails for any reason, it is  
10410 unspecified whether *cp* writes a diagnostic message to standard error.
  - 10411 3. The file permission bits and the S\_ISUID and S\_ISGID bits. Other,  
10412 implementation-defined, bits may be duplicated as well. If this duplication  
10413 fails for any reason, *cp* shall write a diagnostic message to standard error.
- 10414 If the user ID or the group ID cannot be duplicated, the file permission bits  
10415 S\_ISUID and S\_ISGID shall be cleared. If these bits are present in the source file but  
10416 are not duplicated in the destination file, it is unspecified whether *cp* writes a  
10417 diagnostic message to standard error.
- 10418 The order in which the preceding characteristics are duplicated is unspecified. The  
10419 *dest\_file* shall not be deleted if these characteristics cannot be preserved.
- 10420 **-R** Copy file hierarchies.
- 10421 OB **-r** Copy file hierarchies. The treatment of special files is implementation-defined.
- 10422 Specifying more than one of the mutually-exclusive options **-H**, **-L**, and **-P** shall not be  
10423 considered an error. The last option specified shall determine the behavior of the utility.
- 10424 **OPERANDS**
- 10425 The following operands shall be supported:
- 10426 *source\_file* A pathname of a file to be copied.
- 10427 *target\_file* A pathname of an existing or nonexistent file, used for the output when a single  
10428 file is copied.
- 10429 *target* A pathname of a directory to contain the copied files.
- 10430 **STDIN**
- 10431 The standard input shall be used to read an input line in response to each prompt specified in |  
10432 the STDERR section. Otherwise, the standard input shall not be used. |
- 10433 **INPUT FILES**
- 10434 The input files specified as operands may be of any file type.
- 10435 **ENVIRONMENT VARIABLES**
- 10436 The following environment variables shall affect the execution of *cp*:
- 10437 *LANG* Provide a default value for the internationalization variables that are unset or null.  
10438 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
10439 Internationalization Variables for the precedence of internationalization variables  
10440 used to determine the values of locale categories.)

- 10441 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
10442 internationalization variables.
- 10443 *LC\_COLLATE*  
10444 Determine the locale for the behavior of ranges, equivalence classes, and multi-  
10445 character collating elements used in the extended regular expression defined for  
10446 the **yesexpr** locale keyword in the *LC\_MESSAGES* category.
- 10447 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
10448 characters (for example, single-byte as opposed to multi-byte characters in  
10449 arguments and input files) and the behavior of character classes used in the  
10450 extended regular expression defined for the **yesexpr** locale keyword in the  
10451 *LC\_MESSAGES* category.
- 10452 *LC\_MESSAGES*  
10453 Determine the locale for the processing of affirmative responses that should be  
10454 used to affect the format and contents of diagnostic messages written to standard  
10455 error.
- 10456 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 10457 **ASYNCHRONOUS EVENTS**  
10458 Default.
- 10459 **STDOUT**  
10460 Not used.
- 10461 **STDERR**  
10462 A prompt shall be written to standard error under the conditions specified in the DESCRIPTION  
10463 section. The prompt shall contain the destination pathname, but its format is otherwise  
10464 unspecified. Otherwise, the standard error shall be used only for diagnostic messages.
- 10465 **OUTPUT FILES**  
10466 The output files may be of any type.
- 10467 **EXTENDED DESCRIPTION**  
10468 None.
- 10469 **EXIT STATUS**  
10470 The following exit values shall be returned:  
10471 0 All files were copied successfully.  
10472 >0 An error occurred.
- 10473 **CONSEQUENCES OF ERRORS**  
10474 If *cp* is prematurely terminated by a signal or error, files or file hierarchies may be only partially  
10475 copied and files and directories may have incorrect permissions or access and modification  
10476 times.

10477 **APPLICATION USAGE**

10478 The difference between **-R** and **-r** is in the treatment by *cp* of file types other than regular and  
10479 directory. The original **-r** flag, for historic reasons, does not handle special files any differently  
10480 from regular files, but always reads the file and copies its contents. This has obvious problems in  
10481 the presence of special file types; for example, character devices, FIFOs, and sockets. The **-R**  
10482 option is intended to recreate the file hierarchy and the **-r** option supports historical practice. It  
10483 was anticipated that a future version of this volume of IEEE Std 1003.1-200x would deprecate  
10484 the **-r** option, and for that reason, there has been no attempt to fix its behavior with respect to  
10485 FIFOs or other file types where copying the file is clearly wrong. However, some  
10486 implementations support **-r** with the same abilities as the **-R** defined in this volume of  
10487 IEEE Std 1003.1-200x. To accommodate them as well as systems that do not, the differences  
10488 between **-r** and **-R** are implementation-defined. Implementations may make them identical.  
10489 The **-r** option is now marked obsolescent.

10490 The set-user-ID and set-group-ID bits are explicitly cleared when files are created. This is to  
10491 prevent users from creating programs that are set-user-ID or set-group-ID to them when  
10492 copying files or to make set-user-ID or set-group-ID files accessible to new groups of users. For  
10493 example, if a file is set-user-ID and the copy has a different group ID than the source, a new  
10494 group of users has execute permission to a set-user-ID program than did previously. In  
10495 particular, this is a problem for superusers copying users' trees.

10496 **EXAMPLES**

10497 None.

10498 **RATIONALE**

10499 The **-i** option exists on BSD systems, giving applications and users a way to avoid accidentally  
10500 removing files when copying. Although the 4.3 BSD version does not prompt if the standard  
10501 input is not a terminal, the standard developers decided that use of **-i** is a request for interaction,  
10502 so when the destination path exists, the utility takes instructions from whatever responds on  
10503 standard input.

10504 The exact format of the interactive prompts is unspecified. Only the general nature of the  
10505 contents of prompts are specified because implementations may desire more descriptive  
10506 prompts than those used on historical implementations. Therefore, an application using the **-i**  
10507 option relies on the system to provide the most suitable dialog directly with the user, based on  
10508 the behavior specified.

10509 The **-p** option is historical practice on BSD systems, duplicating the time of last data  
10510 modification and time of last access. This volume of IEEE Std 1003.1-200x extends it to preserve  
10511 the user and group IDs, as well as the file permissions. This requirement has obvious problems  
10512 in that the directories are almost certainly modified after being copied. This volume of  
10513 IEEE Std 1003.1-200x requires that the modification times be preserved. The statement that the  
10514 order in which the characteristics are duplicated is unspecified is to permit implementations to  
10515 provide the maximum amount of security for the user. Implementations should take into  
10516 account the obvious security issues involved in setting the owner, group, and mode in the  
10517 wrong order or creating files with an owner, group, or mode different from the final value.

10518 It is unspecified whether *cp* writes diagnostic messages when the user and group IDs cannot be  
10519 set due to the widespread practice of users using **-p** to duplicate some portion of the file  
10520 characteristics, indifferent to the duplication of others. Historic implementations only write  
10521 diagnostic messages on errors other than [EPERM].

10522 The **-r** option is historical practice on BSD and BSD-derived systems, copying file hierarchies as  
10523 opposed to single files. This functionality is used heavily in historical applications, and its loss  
10524 would significantly decrease consensus. The **-R** option was added as a close synonym to the **-r**  
10525 option, selected for consistency with all other options in this volume of IEEE Std 1003.1-200x

10526 that do recursive directory descent.

10527 When a failure occurs during the copying of a file hierarchy, *cp* is required to attempt to copy  
10528 files that are on the same level in the hierarchy or above the file where the failure occurred. It is  
10529 unspecified if *cp* shall attempt to copy files below the file where the failure occurred (which  
10530 cannot succeed in any case).

10531 Permissions, owners, and groups of created special file types have been deliberately left as  
10532 implementation-defined. This is to allow systems to satisfy special requirements (for example,  
10533 allowing users to create character special devices, but requiring them to be owned by a certain  
10534 group). In general, it is strongly suggested that the permissions, owner, and group be the same  
10535 as if the user had run the historical *mknod*, *ln*, or other utility to create the file. It is also probable  
10536 that additional privileges are required to create block, character, or other implementation-  
10537 defined special file types.

10538 Additionally, the **-p** option explicitly requires that all set-user-ID and set-group-ID permissions  
10539 be discarded if any of the owner or group IDs cannot be set. This is to keep users from  
10540 unintentionally giving away special privilege when copying programs.

10541 When creating regular files, historical versions of *cp* use the mode of the source file as modified  
10542 by the file mode creation mask. Other choices would have been to use the mode of the source file  
10543 unmodified by the creation mask or to use the same mode as would be given to a new file  
10544 created by the user (plus the execution bits of the source file) and then modify it by the file mode  
10545 creation mask. In the absence of any strong reason to change historic practice, it was in large part  
10546 retained.

10547 When creating directories, historical versions of *cp* use the mode of the source directory, plus  
10548 read, write, and search bits for the owner, as modified by the file mode creation mask. This is  
10549 done so that *cp* can copy trees where the user has read permission, but the owner does not. A  
10550 side effect is that if the file creation mask denies the owner permissions, *cp* fails. Also, once the  
10551 copy is done, historical versions of *cp* set the permissions on the created directory to be the same  
10552 as the source directory, unmodified by the file creation mask.

10553 This behavior has been modified so that *cp* is always able to create the contents of the directory,  
10554 regardless of the file creation mask. After the copy is done, the permissions are set to be the same  
10555 as the source directory, as modified by the file creation mask. This latter change from historical  
10556 behavior is to prevent users from accidentally creating directories with permissions beyond  
10557 those they would normally set and for consistency with the behavior of *cp* in creating files.

10558 It is not a requirement that *cp* detect attempts to copy a file to itself; however, implementations  
10559 are strongly encouraged to do so. Historical implementations have detected the attempt in most  
10560 cases.

10561 There are two methods of copying subtrees in this volume of IEEE Std 1003.1-200x. The other  
10562 method is described as part of the *pax* utility (see *pax* (on page 2900)). Both methods are  
10563 historical practice. The *cp* utility provides a simpler, more intuitive interface, while *pax* offers a  
10564 finer granularity of control. Each provides additional functionality to the other; in particular, *pax*  
10565 maintains the hard-link structure of the hierarchy, while *cp* does not. It is the intention of the  
10566 standard developers that the results be similar (using appropriate option combinations in both  
10567 utilities). The results are not required to be identical; there seemed insufficient gain to  
10568 applications to balance the difficulty of implementations having to guarantee that the results  
10569 would be exactly identical.

10570 The wording allowing *cp* to copy a directory to implementation-defined file types not specified  
10571 by the System Interfaces volume of IEEE Std 1003.1-200x is provided so that implementations  
10572 supporting symbolic links are not required to prohibit copying directories to symbolic links.  
10573 Other extensions to the System Interfaces volume of IEEE Std 1003.1-200x file types may need to

10574 use this loophole as well.

10575 **FUTURE DIRECTIONS**

10576 The `-r` option may be removed; use `-R` instead.

10577 **SEE ALSO**

10578 *mv, find, ln, pax*

10579 **CHANGE HISTORY**

10580 First released in Issue 2.

10581 **Issue 6**

10582 The `-r` option is marked obsolescent.

10583 The new options `-H`, `-L`, and `-P` are added to align with the IEEE P1003.2b draft standard. These  
10584 options affect the processing of symbolic links. |

10585 IEEE PASC Interpretation 1003.2 #194 is applied, adding a description of the `-P` option. |

## 10586 NAME

10587 crontab — schedule periodic background work

## 10588 SYNOPSIS

10589 UP crontab [*file*]

10590 crontab [ -e | -l | -r ]

10591

## 10592 DESCRIPTION

10593 The *crontab* utility shall create, replace, or edit a user's *crontab* entry; a crontab entry is a list of  
 10594 commands and the times at which they shall be executed. The new crontab entry can be input by  
 10595 specifying *file* or input from standard input if no *file* operand is specified, or by using an editor, if  
 10596 **-e** is specified.

10597 Upon execution of a command from a crontab entry, the implementation shall supply a default  
 10598 environment, defining at least the following environment variables:

10599 **HOME** A pathname of the user's home directory.

10600 **LOGNAME** The user's login name.

10601 **PATH** A string representing a search path guaranteed to find all of the standard utilities.

10602 **SHELL** A pathname of the command interpreter. When *crontab* is invoked as specified by  
 10603 this volume of IEEE Std 1003.1-200x, the value shall be a pathname for *sh*.

10604 The values of these variables when *crontab* is invoked as specified by this volume of  
 10605 IEEE Std 1003.1-200x shall not affect the default values provided when the scheduled command  
 10606 is run.

10607 If standard output and standard error are not redirected by commands executed from the  
 10608 crontab entry, any generated output or errors shall be mailed, via an implementation-defined  
 10609 method, to the user.

10610 XSI Users shall be permitted to use *crontab* if their names appear in the file **/usr/lib/cron/cron.allow**. |  
 10611 If that file does not exist, the file **/usr/lib/cron/cron.deny** shall be checked to determine whether |  
 10612 the user shall be denied access to *crontab*. If neither file exists, only a process with appropriate |  
 10613 privileges shall be allowed to submit a job. If only **cron.deny** exists and is empty, global usage |  
 10614 shall be permitted. The **cron.allow** and **cron.deny** files shall consist of one user name per line. |

## 10615 OPTIONS

10616 The *crontab* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section  
 10617 12.2, Utility Syntax Guidelines.

10618 The following options shall be supported:

10619 **-e** Edit a copy of the invoking user's crontab entry, or create an empty entry to edit if  
 10620 the crontab entry does not exist. When editing is complete, the entry shall be  
 10621 installed as the user's crontab entry.

10622 **-l** (The letter ell.) List the invoking user's crontab entry.

10623 **-r** Remove the invoking user's crontab entry.

## 10624 OPERANDS

10625 The following operand shall be supported:

10626 **file** The pathname of a file that contains specifications, in the format defined in the  
 10627 INPUT FILES section, for crontab entries.

10628 **STDIN**

10629 See the INPUT FILES section.

10630 **INPUT FILES**

10631 In the POSIX locale, the user or application shall ensure that a crontab entry is a text file  
 10632 consisting of lines of six fields each. The fields shall be separated by <blank>s. The first five  
 10633 fields shall be integer patterns that specify the following:

10634 1. Minute [0,59] |

10635 2. Hour [0,23] |

10636 3. Day of the month [1,31] |

10637 4. Month of the year [1,12] |

10638 5. Day of the week ([0,6] with 0=Sunday) |

10639 Each of these patterns can be either an asterisk (meaning all valid values), an element, or a list of  
 10640 elements separated by commas. An element shall be either a number or two numbers separated  
 10641 by a hyphen (meaning an inclusive range). The specification of days can be made by two fields  
 10642 (day of the month and day of the week). If month, day of month, and day of week are all  
 10643 asterisks, every day shall be matched. If either the month or day of month is specified as an  
 10644 element or list, but the day of week is an asterisk, the month and day of month fields shall  
 10645 specify the days that match. If both month and day of month are specified as asterisk, but day of  
 10646 week is an element or list, then only the specified days of the week match. Finally, if either the  
 10647 month or day of month is specified as an element or list, and the day of week is also specified as  
 10648 an element or list, then any day matching either the month and day of month, or the day of  
 10649 week, shall be matched.

10650 The sixth field of a line in a crontab entry is a string that shall be executed by *sh* at the specified  
 10651 times. A percent sign character in this field shall be translated to a <newline>. Any character  
 10652 preceded by a backslash (including the '%'') shall cause that character to be treated literally.  
 10653 Only the first line (up to a '%' or end-of-line) of the command field shall be executed by the  
 10654 command interpreter. The other lines shall be made available to the command as standard input.

10655 Blank lines and those whose first non-&lt;blank&gt; is '#' shall be ignored.

10656 XSI The text files `/usr/lib/cron/cron.allow` and `/usr/lib/cron/cron.deny` shall contain zero or more |  
 10657 user names, one per line, of users who are, respectively, authorized or denied access to the |  
 10658 service underlying the *crontab* utility. |

10659 **ENVIRONMENT VARIABLES**10660 The following environment variables shall affect the execution of *crontab*:

10661 *EDITOR* Determine the editor to be invoked when the `-e` option is specified. The default  
 10662 editor shall be *vi*.

10663 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 10664 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
 10665 Internationalization Variables for the precedence of internationalization variables  
 10666 used to determine the values of locale categories.)

10667 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 10668 internationalization variables.

10669 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 10670 characters (for example, single-byte as opposed to multi-byte characters in  
 10671 arguments and input files).

10672 *LC\_MESSAGES*  
10673 Determine the locale that should be used to affect the format and contents of  
10674 diagnostic messages written to standard error.

10675 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

10676 **ASYNCHRONOUS EVENTS**  
10677 Default.

10678 **STDOUT**  
10679 If the `-l` option is specified, the crontab entry shall be written to the standard output.

10680 **STDERR**  
10681 The standard error shall be used only for diagnostic messages.

10682 **OUTPUT FILES**  
10683 None.

10684 **EXTENDED DESCRIPTION**  
10685 None.

10686 **EXIT STATUS**  
10687 The following exit values shall be returned:  
10688 0 Successful completion.  
10689 >0 An error occurred.

10690 **CONSEQUENCES OF ERRORS**  
10691 The user's crontab entry is not submitted, removed, edited, or listed.

10692 **APPLICATION USAGE**  
10693 The format of the *crontab* entry shown here is guaranteed only for the POSIX locale. Other  
10694 cultures may be supported with substantially different interfaces, although implementations are  
10695 encouraged to provide comparable levels of functionality.

10696 The default settings of the *HOME*, *LOGNAME*, *PATH*, and *SHELL* variables that are given to the  
10697 scheduled job are not affected by the settings of those variables when *crontab* is run; as stated,  
10698 they are defaults. The text about "invoked as specified by this volume of IEEE Std 1003.1-200x"  
10699 means that the implementation may provide extensions that allow these variables to be affected  
10700 at runtime, but that the user has to take explicit action in order to access the extension, such as  
10701 give a new option flag or modify the format of the crontab entry.

10702 A typical user error is to type only *crontab*; this causes the system to wait for the new crontab  
10703 entry on standard input. If end-of-file is typed (generally `<control>-D`), the crontab entry is  
10704 replaced by an empty file. In this case, the user should type the interrupt character, which  
10705 prevents the crontab entry from being replaced.

10706 **EXAMPLES**  
10707 1. Clean up **core** files every weekday morning at 3:15 am:  
10708 `15 3 * * 1-5 find $HOME -name core 2>/dev/null | xargs rm -f`  
10709 2. Mail a birthday greeting:  
10710 `0 12 14 2 * mailx john%Happy Birthday!%Time for lunch.`  
10711 3. As an example of specifying the two types of days:  
10712 `0 0 1,15 * 1`

10713 would run a command on the first and fifteenth of each month, as well as on every  
10714 Monday. To specify days by only one field, the other field should be set to '\*'; for  
10715 example:

10716 0 0 \* \* 1

10717 would run a command only on Mondays.

#### 10718 RATIONALE

10719 All references to a *cron* daemon and to *cron files* have been omitted. Although historical  
10720 implementations have used this arrangement, there is no reason to limit future implementations.

10721 This description of *crontab* is designed to support only users with normal privileges. The format  
10722 of the input is based on the System V *crontab*; however, there is no requirement here that the  
10723 actual system database used by the *cron* daemon (or a similar mechanism) use this format  
10724 internally. For example, systems derived from BSD are likely to have an additional field  
10725 appended that indicates the user identity to be used when the job is submitted.

10726 The `-e` option was adopted from the SVID as a user convenience, although it does not exist in all  
10727 historical implementations.

#### 10728 FUTURE DIRECTIONS

10729 None.

#### 10730 SEE ALSO

10731 *at*

#### 10732 CHANGE HISTORY

10733 First released in Issue 2.

#### 10734 Issue 6

10735 This utility is now marked as part of the User Portability Utilities option.

10736 The normative text is reworded to avoid use of the term “must” for application requirements.

## 10737 NAME

10738 csplit — split files based on context

## 10739 SYNOPSIS

10740 UP csplit [-ks][-f *prefix*][-n *number*] *file* *arg1* ...*argn*

10741

## 10742 DESCRIPTION

10743 The *csplit* utility shall read the file named by the *file* operand, write all or part of that file into  
10744 other files as directed by the *arg* operands, and write the sizes of the files.

## 10745 OPTIONS

10746 The *csplit* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section  
10747 12.2, Utility Syntax Guidelines.

10748 The following options shall be supported:

10749 **-f *prefix*** Name the created files *prefix00*, *prefix01*, ..., *prefixn*. The default is **xx00** ... **xxn**. If  
10750 the *prefix* argument would create a filename exceeding {NAME\_MAX} bytes, an  
10751 error shall result, *csplit* shall exit with a diagnostic message and no files shall be  
10752 created.10753 **-k** Leave previously created files intact. By default, *csplit* shall remove created files if  
10754 an error occurs.10755 **-n *number*** Use *number* decimal digits to form filenames for the file pieces. The default shall be  
10756 2.10757 **-s** Suppress the output of file size messages.

## 10758 OPERANDS

10759 The following operands shall be supported:

10760 ***file*** The pathname of a text file to be split. If *file* is '-', the standard input shall be  
10761 used.10762 The operands *arg1* ... *argn* can be a combination of the following:10763 ***/rexp/[offset]***10764 A file shall be created using the content of the lines from the current line up to, but  
10765 not including, the line that results from the evaluation of the regular expression  
10766 with *offset*, if any, applied. The regular expression *rexp* shall follow the rules for  
10767 basic regular expressions described in the Base Definitions volume of  
10768 IEEE Std 1003.1-200x, Section 9.3, Basic Regular Expressions. The application shall  
10769 use the sequence "\/" to specify a slash character within the *rexp*. The optional  
10770 *offset* shall be a positive or negative integer value representing a number of lines.  
10771 A positive integer value can be preceded by '+'. If the selection of lines from an  
10772 *offset* expression of this type would create a file with zero lines, or one with greater  
10773 than the number of lines left in the input file, the results are unspecified. After the  
10774 section is created, the current line shall be set to the line that results from the  
10775 evaluation of the regular expression with any *offset* applied. If the current line is  
10776 the first line in the file and a regular expression operation has not yet been  
10777 performed, the pattern match of *rexp* shall be applied from the current line to the  
10778 end of the file. Otherwise, the pattern match of *rexp* shall be applied from the line  
10779 following the current line to the end of the file.10780 ***%rexp%[offset]***10781 Equivalent to */rexp/[offset]*, except that no file shall be created for the selected  
10782 section of the input file. The application shall use the sequence "\%" to specify a

- 10783 percent-sign character within the *rexp*.
- 10784 *line\_no* Create a file from the current line up to (but not including) the line number *line\_no*.  
 10785 Lines in the file shall be numbered starting at one. The current line becomes  
 10786 *line\_no*.
- 10787 *{num}* Repeat operand. This operand can follow any of the operands described  
 10788 previously. If it follows a *rexp* type operand, that operand shall be applied *num*  
 10789 more times. If it follows a *line\_no* operand, the file shall be split every *line\_no* lines,  
 10790 *num* times, from that point.
- 10791 An error shall be reported if an operand does not reference a line between the current position  
 10792 and the end of the file.
- 10793 **STDIN**
- 10794 See the INPUT FILES section.
- 10795 **INPUT FILES**
- 10796 The input file shall be a text file.
- 10797 **ENVIRONMENT VARIABLES**
- 10798 The following environment variables shall affect the execution of *csplit*:
- 10799 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 10800 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
 10801 Internationalization Variables for the precedence of internationalization variables  
 10802 used to determine the values of locale categories.)
- 10803 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 10804 internationalization variables.
- 10805 *LC\_COLLATE*  
 10806 Determine the locale for the behavior of ranges, equivalence classes, and multi-  
 10807 character collating elements within regular expressions.
- 10808 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 10809 characters (for example, single-byte as opposed to multi-byte characters in  
 10810 arguments and input files) and the behavior of character classes within regular  
 10811 expressions.
- 10812 *LC\_MESSAGES*  
 10813 Determine the locale that should be used to affect the format and contents of  
 10814 diagnostic messages written to standard error.
- 10815 *XSI* *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 10816 **ASYNCHRONOUS EVENTS**
- 10817 If the *-k* option is specified, created files shall be retained. Otherwise, the default action occurs.
- 10818 **STDOUT**
- 10819 Unless the *-s* option is used, the standard output shall consist of one line per file created, with a  
 10820 format as follows:  
 10821 "%d\n", <file size in bytes>
- 10822 **STDERR**
- 10823 The standard error shall be used only for diagnostic messages.

10824 **OUTPUT FILES**

10825 The output files shall contain portions of the original input file; otherwise, unchanged.

10826 **EXTENDED DESCRIPTION**

10827 None.

10828 **EXIT STATUS**

10829 The following exit values shall be returned:

10830 0 Successful completion.

10831 >0 An error occurred.

10832 **CONSEQUENCES OF ERRORS**

10833 By default, created files shall be removed if an error occurs. When the **-k** option is specified,  
10834 created files shall not be removed if an error occurs.

10835 **APPLICATION USAGE**

10836 None.

10837 **EXAMPLES**

10838 1. This example creates four files, **cobol00 ... cobol03**:

10839 `csplit -f cobol file '/procedure division/' /par5./ /par16./`

10840 After editing the split files, they can be recombined as follows:

10841 `cat cobol0[0-3] > file`

10842 Note that this example overwrites the original file.

10843 2. This example would split the file after the first 99 lines, and every 100 lines thereafter, up  
10844 to 9999 lines; this is because lines in the file are numbered from 1 rather than zero, for  
10845 historical reasons:

10846 `csplit -k file 100 {99}`

10847 3. Assuming that **prog.c** follows the C-language coding convention of ending routines with a  
10848 `'}'` at the beginning of the line, this example creates a file containing each separate C  
10849 routine (up to 21) in **prog.c**:

10850 `csplit -k prog.c '%main(%' '/^}/+1' {20}`

10851 **RATIONALE**

10852 The **-n** option was added to extend the range of filenames that could be handled.

10853 Consideration was given to adding a **-a** flag to use the alphabetic filename generation used by  
10854 the historical *split* utility, but the functionality added by the **-n** option was deemed to make  
10855 alphabetic naming unnecessary.

10856 **FUTURE DIRECTIONS**

10857 None.

10858 **SEE ALSO**

10859 *sed*, *split*

10860 **CHANGE HISTORY**

10861 First released in Issue 2.

10862 **Issue 5**

10863 FUTURE DIRECTIONS section added.

10864 **Issue 6**

10865 This utility is now marked as part of the User Portability Utilities option.

10866 The APPLICATION USAGE section is added.

10867 The description of regular expression operands is changed to align with the IEEE P1003.2b draft  
10868 standard.

10869 The normative text is reworded to avoid use of the term “must” for application requirements.

## 10870 NAME

10871 ctags — create a tags file (DEVELOPMENT, FORTRAN)

## 10872 SYNOPSIS

10873 UP `ctags [-a][-f tagsfile] pathname ...`

10874 `ctags -x pathname ...`

10875

## 10876 DESCRIPTION

10877 The *ctags* utility shall be provided on systems that support the User Portability Utilities option,  
10878 the Software Development Utilities option, and either or both of the C-Language Development  
10879 Utilities option and FORTRAN Development Utilities option. On other systems, it is optional.

10880 The *ctags* utility shall write a *tags* file or an index of objects from C-language or FORTRAN  
10881 source files specified by the *pathname* operands. The tags file shall list the locators of language-  
10882 specific objects within the source files. A locator consists of a name, pathname, and either a  
10883 search pattern or a line number that can be used in searching for the object definition. The  
10884 objects that shall be recognized are specified in the EXTENDED DESCRIPTION section.

## 10885 OPTIONS

10886 The *ctags* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section  
10887 12.2, Utility Syntax Guidelines.

10888 The following options shall be supported:

10889 **-a** Append to tags file.

10890 **-f *tagsfile*** Write the object locator lists into *tagsfile* instead of the default file named **tags** in  
10891 the current directory.

10892 **-x** Produce a list of object names, the line number, and filename in which each is  
10893 defined, as well as the text of that line, and write this to the standard output. A  
10894 **tags** file shall not be created when **-x** is specified.

## 10895 OPERANDS

10896 The following *pathname* operands are supported:

10897 ***file.c*** Files with basenames ending with the **.c** suffix shall be treated as C-language  
10898 source code. Such files that are not valid input to *c99* produce unspecified results.

10899 ***file.h*** Files with basenames ending with the **.h** suffix shall be treated as C-language  
10900 source code. Such files that are not valid input to *c99* produce unspecified results.

10901 ***file.f*** Files with basenames ending with the **.f** suffix shall be treated as FORTRAN-  
10902 language source code. Such files that are not valid input to *fort77* produce  
10903 unspecified results.

10904 The handling of other files is implementation-defined.

## 10905 STDIN

10906 See the INPUT FILES section.

## 10907 INPUT FILES

10908 The input files shall be text files containing source code in the language indicated by the operand  
10909 filename suffixes.

10910 **ENVIRONMENT VARIABLES**

10911 The following environment variables shall affect the execution of *ctags*:

10912 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 10913 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
 10914 Internationalization Variables for the precedence of internationalization variables  
 10915 used to determine the values of locale categories.)

10916 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 10917 internationalization variables.

10918 *LC\_COLLATE*

10919 Determine the order in which output is sorted for the *-x* option. The POSIX locale  
 10920 determines the order in which the tags file is written.

10921 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 10922 characters (for example, single-byte as opposed to multi-byte characters in  
 10923 arguments and input files). When processing C-language source code, if the locale  
 10924 is not compatible with the C locale described by the ISO C standard, the results are  
 10925 unspecified.

10926 *LC\_MESSAGES*

10927 Determine the locale that should be used to affect the format and contents of  
 10928 diagnostic messages written to standard error.

10929 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

10930 **ASYNCHRONOUS EVENTS**

10931 Default.

10932 **STDOUT**

10933 The list of object name information produced by the *-x* option shall be written to standard  
 10934 output in the following format:

10935 "%s %d %s %s", <object-name>, <line-number>, <filename>,  
 10936 <text>

10937 where <text> is the text of line <line-number> of file <filename>.

10938 **STDERR**

10939 The standard error shall be used only for diagnostic messages.

10940 **OUTPUT FILES**

10941 When the *-x* option is not specified, the format of the output file shall be:

10942 "%s\t%s\t/%s/\n", <identifier>, <filename>, <pattern>

10943 where <pattern> is a search pattern that could be used by an editor to find the defining instance  
 10944 of <identifier> in <filename> (where *defining instance* is indicated by the declarations listed in the  
 10945 EXTENDED DESCRIPTION).

10946 An optional circumflex ('^') can be added as a prefix to <pattern>, and an optional dollar sign  
 10947 can be appended to <pattern> to indicate that the pattern is anchored to the beginning (end) of a  
 10948 line of text. Any slash or backslash characters in <pattern> shall be preceded by a backslash  
 10949 character. The anchoring circumflex, dollar sign, and escaping backslash characters shall not be  
 10950 considered part of the search pattern. All other characters in the search pattern shall be  
 10951 considered literal characters.

10952 An alternative format is:

- 10953        "%s\t%s\t?%s?\n", <identifier>, <filename>, <pattern>
- 10954        which is identical to the first format except that slashes in <pattern> shall not be preceded by  
10955        escaping backslash characters, and question mark characters in <pattern> shall be preceded by  
10956        backslash characters.
- 10957        A second alternative format is:
- 10958        "%s\t%s\t%d\n", <identifier>, <filename>, <lineno>
- 10959        where <lineno> is a decimal line number that could be used by an editor to find <identifier> in  
10960        <filename>.
- 10961        Neither alternative format shall be produced by *ctags* when it is used as described by  
10962        IEEE Std 1003.1-200x, but the standard utilities that process tags files shall be able to process  
10963        those formats as well as the first format.
- 10964        In any of these formats, the file shall be sorted by identifier, based on the collation sequence in  
10965        the POSIX locale.
- 10966        **EXTENDED DESCRIPTION**
- 10967        If the operand identifies C-language source, the *ctags* utility shall attempt to produce an output  
10968        line for each of the following objects:
- 10969        • Function definitions
  - 10970        • Type definitions
  - 10971        • Macros with arguments
- 10972        It may also produce output for any of the following objects:
- 10973        • Function prototypes
  - 10974        • Structures
  - 10975        • Unions
  - 10976        • Global variable definitions
  - 10977        • Enumeration types
  - 10978        • Macros without arguments
  - 10979        • **#define** statements
  - 10980        • **#line** statements
- 10981        Any **#if** and **#ifdef** statements shall produce no output. The tag **main** is treated specially in C  
10982        programs. The tag formed shall be created by prefixing **M** to the name of the file, with the  
10983        trailing **.c**, and leading pathname components (if any) removed.
- 10984        On systems that do not support the C-Language Development Utilities option, *ctags* produces |  
10985        undefined results for C-language source code files. It should write to standard error a message |  
10986        identifying this condition and cause a non-zero exit status to be produced. |
- 10987        If the operand identifies FORTRAN source, the *ctags* utility shall produce an output line for each  
10988        function definition. It may also produce output for any of the following objects:
- 10989        • Subroutine definitions
  - 10990        • COMMON statements
  - 10991        • PARAMETER statements

- 10992           • DATA and BLOCK DATA statements
- 10993           • Statement numbers
- 10994           On systems that do not support the FORTRAN Development Utilities option, *ctags* produces
- 10995           unspecified results for FORTRAN source code files. It should write to standard error a message
- 10996           identifying this condition and cause a non-zero exit status to be produced.
- 10997           It is implementation-defined what other objects (including duplicate identifiers) produce output.
- 10998 **EXIT STATUS**
- 10999           The following exit values shall be returned:
- 11000           0   Successful completion.
- 11001           >0  An error occurred.
- 11002 **CONSEQUENCES OF ERRORS**
- 11003           Default.
- 11004 **APPLICATION USAGE**
- 11005           The output with *-x* is meant to be a simple index that can be written out as an off-line readable
- 11006           function index. If the input files to *ctags* (such as *.c* files) were not created using the same locale
- 11007           as that in effect when *ctags -x* is run, results might not be as expected.
- 11008           The description of C-language processing says “attempts to” because the C language can be
- 11009           greatly confused, especially through the use of *#defines*, and this utility would be of no use if
- 11010           the real C preprocessor were run to identify them. The output from *ctags* may be fooled and
- 11011           incorrect for various constructs.
- 11012 **EXAMPLES**
- 11013           None.
- 11014 **RATIONALE**
- 11015           The option list was significantly reduced from that provided by historical implementations. The
- 11016           *-F* option was omitted as redundant, since it is the default. The *-B* option was omitted as being
- 11017           of very limited usefulness. The *-t* option was omitted since the recognition of typedefs is now
- 11018           required for C source files. The *-u* option was omitted because the update function was judged
- 11019           to be not only inefficient, but also rarely needed.
- 11020           An early proposal included a *-w* option to suppress warning diagnostics. Since the types of such
- 11021           diagnostics could not be described, the option was omitted as being not useful.
- 11022           The text for *LC\_CTYPE* about compatibility with the C locale acknowledges that the ISO C
- 11023           standard imposes requirements on the locale used to process C source. This could easily be a
- 11024           superset of that known as “the C locale” by way of implementation extensions, or one of a few
- 11025           alternative locales for systems supporting different codesets. No statement is made for
- 11026           FORTRAN because the ANSI X3.9-1978 standard (FORTRAN 77) does not (yet) define a similar
- 11027           locale concept. However, a general rule in this volume of IEEE Std 1003.1-200x is that any time
- 11028           that locales do not match (preparing a file for one locale and processing it in another), the results
- 11029           are suspect.
- 11030           The collation sequence of the tags file is not affected by *LC\_COLLATE* because it is typically not
- 11031           used by human readers, but only by programs such as *vi* to locate the tag within the source files.
- 11032           Using the POSIX locale eliminates some of the problems of coordinating locales between the
- 11033           *ctags* file creator and the *vi* file reader.
- 11034           Historically, the tags file has been used only by *ex* and *vi*. However, the format of the tags file
- 11035           has been published to encourage other programs to use the tags in new ways. The format allows
- 11036           either patterns or line numbers to find the identifiers because the historical *vi* recognizes either.

- 11037 The *ctags* utility does not produce the format using line numbers because it is not useful  
11038 following any source file changes that add or delete lines. The documented search patterns  
11039 match historical practice. It should be noted that literal leading circumflex or trailing dollar-sign  
11040 characters in the search pattern will only behave correctly if anchored to the beginning of the  
11041 line or end of the line by an additional circumflex or dollar-sign character.
- 11042 Historical implementations also understand the objects used by the languages Pascal and  
11043 sometimes LISP, and they understand the C source output by *lex* and *yacc*. The *ctags* utility is  
11044 not required to accommodate these languages, although implementors are encouraged to do so.
- 11045 The following historical option was not specified, as *vgrind* is not included in this volume of  
11046 IEEE Std 1003.1-200x:
- 11047 **-v** If the **-v** flag is given, an index of the form expected by *vgrind* is produced on the  
11048 standard output. This listing contains the function name, filename, and page  
11049 number (assuming 64-line pages). Since the output is sorted into lexicographic  
11050 order, it may be desired to run the output through *sort -f*. Sample use:
- 11051 `ctags -v files | sort -f > index vgrind -x index`
- 11052 The special treatment of the tag **main** makes the use of *ctags* practical in directories with more  
11053 than one program.
- 11054 **FUTURE DIRECTIONS**
- 11055 None.
- 11056 **SEE ALSO**
- 11057 *c99, fort77, vi*
- 11058 **CHANGE HISTORY**
- 11059 First released in Issue 4.
- 11060 **Issue 5**
- 11061 FUTURE DIRECTIONS section added.
- 11062 **Issue 6**
- 11063 This utility is now marked as part of the User Portability Utilities option.
- 11064 The OUTPUT FILES section is changed to align with the IEEE P1003.2b draft standard.
- 11065 The normative text is reworded to avoid use of the term “must” for application requirements.
- 11066 IEEE PASC Interpretation 1003.2 #168 is applied, changing “create” to “write” in the  
11067 DESCRIPTION.

## 11068 NAME

11069 cut — cut out selected fields of each line of a file

## 11070 SYNOPSIS

11071 cut -b *list* [-n] [*file* ...]

11072 cut -c *list* [*file* ...]

11073 cut -f *list* [-d *delim*][-s][*file* ...]

## 11074 DESCRIPTION

11075 The *cut* utility shall cut out bytes (**-b** option), characters (**-c** option) or character-delimited fields  
 11076 (**-f** option) from each line in one or more files, concatenate them, and write them to standard  
 11077 output.

## 11078 OPTIONS

11079 The *cut* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section  
 11080 12.2, Utility Syntax Guidelines.

11081 The application shall ensure that the option-argument *list* (see options **-b**, **-c**, and **-f** below) is a  
 11082 comma-separated list or <blank>-separated list of positive numbers and ranges. Ranges can be  
 11083 in three forms. The first is two positive numbers separated by a hyphen (*low-high*), which  
 11084 represents all fields from the first number to the second number. The second is a positive  
 11085 number preceded by a hyphen (*-high*), which represents all fields from field number 1 to that  
 11086 number. The third is a positive number followed by a hyphen (*low-*), which represents that  
 11087 number to the last field, inclusive. The elements in *list* can be repeated, can overlap, and can be  
 11088 specified in any order, but the bytes, characters, or fields selected shall be written in the order of  
 11089 the input data. If an element appears in the selection list more than once, it shall be written  
 11090 exactly once.

11091 The following options shall be supported:

11092 **-b list** Cut based on a *list* of bytes. Each selected byte shall be output unless the **-n** option  
 11093 is also specified. It shall not be an error to select bytes not present in the input line.

11094 **-c list** Cut based on a *list* of characters. Each selected character shall be output. It shall  
 11095 not be an error to select characters not present in the input line.

11096 **-d delim** Set the field delimiter to the character *delim*. The default is the <tab>.

11097 **-f list** Cut based on a *list* of fields, assumed to be separated in the file by a delimiter  
 11098 character (see **-d**). Each selected field shall be output. Output fields shall be  
 11099 separated by a single occurrence of the field delimiter character. Lines with no field  
 11100 delimiters shall be passed through intact, unless **-s** is specified. It shall not be an  
 11101 error to select fields not present in the input line.

11102 **-n** Do not split characters. When specified with the **-b** option, each element in *list* of  
 11103 the form *low-high* (hyphen-separated numbers) shall be modified as follows:

- 11104 • If the byte selected by *low* is not the first byte of a character, *low* shall be  
 11105 decremented to select the first byte of the character originally selected by *low*.  
 11106 If the byte selected by *high* is not the last byte of a character, *high* shall be  
 11107 decremented to select the last byte of the character prior to the character  
 11108 originally selected by *high*, or zero if there is no prior character. If the resulting  
 11109 range element has *high* equal to zero or *low* greater than *high*, the list element  
 11110 shall be dropped from *list* for that input line without causing an error.

11111 Each element in *list* of the form *low-* shall be treated as above with *high* set to the  
 11112 number of bytes in the current line, not including the terminating <newline>. Each

11113 element in *list* of the form *-high* shall be treated as above with *low* set to 1. Each  
 11114 element in *list* of the form *num* (a single number) shall be treated as above with *low*  
 11115 set to *num* and *high* set to *num*.

11116 **-s** Suppress lines with no delimiter characters, when used with the **-f** option. Unless  
 11117 specified, lines with no delimiters shall be passed through untouched.

#### 11118 OPERANDS

11119 The following operand shall be supported:

11120 **file** A pathname of an input file. If no **file** operands are specified, or if a **file** operand is  
 11121 **'-'**, the standard input shall be used.

#### 11122 STDIN

11123 The standard input shall be used only if no **file** operands are specified, or if a **file** operand is **'-'**.  
 11124 See the INPUT FILES section.

#### 11125 INPUT FILES

11126 The input files shall be text files, except that line lengths shall be unlimited.

#### 11127 ENVIRONMENT VARIABLES

11128 The following environment variables shall affect the execution of *cut*:

11129 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 11130 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
 11131 Internationalization Variables for the precedence of internationalization variables  
 11132 used to determine the values of locale categories.)

11133 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 11134 internationalization variables.

11135 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 11136 characters (for example, single-byte as opposed to multi-byte characters in  
 11137 arguments and input files).

#### 11138 LC\_MESSAGES

11139 Determine the locale that should be used to affect the format and contents of  
 11140 diagnostic messages written to standard error.

11141 **XSI** **NLSPATH** Determine the location of message catalogs for the processing of **LC\_MESSAGES**.

#### 11142 ASYNCHRONOUS EVENTS

11143 Default.

#### 11144 STDOUT

11145 The *cut* utility output shall be a concatenation of the selected bytes, characters, or fields (one of  
 11146 the following):

11147 "%s\n", <concatenation of bytes>

11148 "%s\n", <concatenation of characters>

11149 "%s\n", <concatenation of fields and field delimiters>

#### 11150 STDERR

11151 The standard error shall be used only for diagnostic messages.

#### 11152 OUTPUT FILES

11153 None.

11154 **EXTENDED DESCRIPTION**

11155 None.

11156 **EXIT STATUS**

11157 The following exit values shall be returned:

11158 0 All input files were output successfully.

11159 &gt;0 An error occurred.

11160 **CONSEQUENCES OF ERRORS**

11161 Default.

11162 **APPLICATION USAGE**

11163 Earlier versions of the *cut* utility worked in an environment where bytes and characters were considered equivalent (modulo <backspace> and <tab> processing in some implementations). In  
 11164 the extended world of multi-byte characters, the new **-b** option has been added. The **-n** option  
 11165 (used with **-b**) allows it to be used to act on bytes rounded to character boundaries. The  
 11166 algorithm specified for **-n** guarantees that:  
 11167

11168 `cut -b 1-500 -n file > file1`11169 `cut -b 501- -n file > file2`

11170 ends up with all the characters in **file** appearing exactly once in **file1** or **file2**. (There is,  
 11171 however, a <newline> in both **file1** and **file2** for each <newline> in **file**.)

11172 **EXAMPLES**

11173 Examples of the option qualifier list:

11174 1,4,7 Select the first, fourth, and seventh bytes, characters, or fields and field delimiters.

11175 1-3,8 Equivalent to 1,2,3,8.

11176 -5,10 Equivalent to 1,2,3,4,5,10.

11177 3- Equivalent to third to last, inclusive.

11178 The *low-high* forms are not always equivalent when used with **-b** and **-n** and multi-byte  
 11179 characters; see the description of **-n**.

11180 The following command:

11181 `cut -d : -f 1,6 /etc/passwd`

11182 reads the System V password file (user database) and produces lines of the form:

11183 `<user ID>:<home directory>`

11184 Most utilities in this volume of IEEE Std 1003.1-200x work on text files. The *cut* utility can be  
 11185 used to turn files with arbitrary line lengths into a set of text files containing the same data. The  
 11186 *paste* utility can be used to create (or recreate) files with arbitrary line lengths. For example, if **file**  
 11187 contains long lines:

11188 `cut -b 1-500 -n file > file1`11189 `cut -b 501- -n file > file2`

11190 creates **file1** (a text file) with lines no longer than 500 bytes (plus the <newline>) and **file2** that  
 11191 contains the remainder of the data from **file**. (Note that **file2** is not a text file if there are lines in  
 11192 **file** that are longer than 500 + {LINE\_MAX} bytes.) The original file can be recreated from **file1**  
 11193 and **file2** using the command:

11194 `paste -d "\0" file1 file2 > file`

**11195 RATIONALE**

11196 Some historical implementations do not count <backspace>s in determining character counts  
11197 with the `-c` option. This may be useful for using `cut` for processing `nroff` output. It was  
11198 deliberately decided not to have the `-c` option treat either <backspace>s or <tab>s in any special  
11199 fashion. The `fold` utility does treat these characters specially.

11200 Unlike other utilities, some historical implementations of `cut` exit after not finding an input file,  
11201 rather than continuing to process the remaining `file` operands. This behavior is prohibited by this  
11202 volume of IEEE Std 1003.1-200x, where only the exit status is affected by this problem.

11203 The behavior of `cut` when provided with either mutually-exclusive options or options that do  
11204 not work logically together has been deliberately left unspecified in favor of global wording in  
11205 Section 1.11 (on page 2221).

11206 The OPTIONS section was changed in response to P1003.2-N149. The change represents  
11207 historical practice on all known systems. The original standard was ambiguous on the nature of  
11208 the output.

11209 The `list` option-arguments are historically used to select the portions of the line to be written, but  
11210 do not affect the order of the data. For example:

```
11211 echo abcdefghi | cut -c6,2,4-7,1
```

11212 yields "abdefg".

11213 A proposal to enhance `cut` with the following option:

11214 `-o` Preserve the selected field order. When this option is specified, each byte, character, or field  
11215 (or ranges of such) shall be written in the order specified by the `list` option-argument, even if  
11216 this requires multiple outputs of the same bytes, characters, or fields.

11217 was rejected because this type of enhancement is outside the scope of the IEEE P1003.2b draft  
11218 standard.

**11219 FUTURE DIRECTIONS**

11220 None.

**11221 SEE ALSO**

11222 `grep`, `paste`, Section 2.5 (on page 2235)

**11223 CHANGE HISTORY**

11224 First released in Issue 2.

**11225 Issue 6**

11226 The OPTIONS section is changed to align with the IEEE P1003.2b draft standard.

11227 The normative text is reworded to avoid use of the term “must” for application requirements.

## 11228 NAME

11229 cxref — generate a C-language program cross-reference table (**DEVELOPMENT**)

## 11230 SYNOPSIS

```
11231 xsi cxref [-cs][-o file][-w num] [-D name[=def]]...[-I dir]...
11232 [-U name]... file ...
11233
```

## 11234 DESCRIPTION

11235 The *cxref* utility shall analyze a collection of C-language *files* and attempt to build a cross-  
 11236 reference table. Information from **#define** lines shall be included in the symbol table. A sorted  
 11237 listing shall be written to standard output of all symbols (auto, static, and global) in each *file*  
 11238 separately, or with the *-c* option, in combination. Each symbol shall contain an asterisk before  
 11239 the declaring reference.

## 11240 OPTIONS

11241 The *cxref* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section  
 11242 12.2, Utility Syntax Guidelines, except that the order of the *-D*, *-I*, and *-U* options (which are  
 11243 identical to their interpretation by *c99*) is significant. The following options shall be supported:

|       |                |                                                                                         |  |
|-------|----------------|-----------------------------------------------------------------------------------------|--|
| 11244 | <i>-c</i>      | Write a combined cross-reference of all input files.                                    |  |
| 11245 | <i>-s</i>      | Operate silently; do not print input filenames.                                         |  |
| 11246 | <i>-o file</i> | Direct output to named <i>file</i> .                                                    |  |
| 11247 | <i>-w num</i>  | Format output no wider than <i>num</i> (decimal) columns. This option defaults to 80 if |  |
| 11248 |                | <i>num</i> is not specified or is less than 51.                                         |  |
| 11249 | <i>-D</i>      | Equivalent to <i>c99</i> .                                                              |  |
| 11250 | <i>-I</i>      | Equivalent to <i>c99</i> .                                                              |  |
| 11251 | <i>-U</i>      | Equivalent to <i>c99</i> .                                                              |  |

## 11252 OPERANDS

11253 The following operand shall be supported:

|       |             |                                         |
|-------|-------------|-----------------------------------------|
| 11254 | <i>file</i> | A pathname of a C-language source file. |
|-------|-------------|-----------------------------------------|

## 11255 STDIN

11256 Not used.

## 11257 INPUT FILES

11258 The input files are C-language source files.

## 11259 ENVIRONMENT VARIABLES

11260 The following environment variables shall affect the execution of *cxref*:

|       |                   |                                                                                                                                                                                                                                                                                                                                     |
|-------|-------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 11261 | <i>LANG</i>       | Provide a default value for the internationalization variables that are unset or null.<br>(See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,<br>11262 Internationalization Variables for the precedence of internationalization variables<br>11263 used to determine the values of locale categories.)<br>11264 |
| 11265 | <i>LC_ALL</i>     | If set to a non-empty string value, override the values of all the other<br>11266 internationalization variables.                                                                                                                                                                                                                   |
| 11267 | <i>LC_COLLATE</i> |                                                                                                                                                                                                                                                                                                                                     |
| 11268 |                   | Determine the locale for the ordering of the output.                                                                                                                                                                                                                                                                                |
| 11269 | <i>LC_CTYPE</i>   | Determine the locale for the interpretation of sequences of bytes of text data as<br>11270 characters (for example, single-byte as opposed to multi-byte characters in                                                                                                                                                              |

- 11271 arguments and input files).
- 11272 **LC\_MESSAGES**
- 11273 Determine the locale that should be used to affect the format and contents of  
11274 diagnostic messages written to standard error.
- 11275 **NLSPATH** Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 11276 **ASYNCHRONOUS EVENTS**
- 11277 Default.
- 11278 **STDOUT**
- 11279 The standard output shall be used for the cross-reference listing, unless the **-o** option is used to  
11280 select a different output file.
- 11281 The format of standard output is unspecified, except that the following information shall be  
11282 included:
- 11283 • If the **-c** option is not specified, each portion of the listing shall start with the name of the |  
11284 input file on a separate line. |
  - 11285 • The name line shall be followed by a sorted list of symbols, each with its associated location |  
11286 pathname, the name of the function in which it appears (if it is not a function name itself), |  
11287 and line number references. |
  - 11288 • Each line number may be preceded by an asterisk ( '\*' ) flag, meaning that this is the  
11289 declaring reference. Other single-character flags, with implementation-defined meanings,  
11290 may be included.
- 11291 **STDERR**
- 11292 The standard error shall be used only for diagnostic messages. |
- 11293 **OUTPUT FILES**
- 11294 The output file named by the **-o** option shall be used instead of standard output.
- 11295 **EXTENDED DESCRIPTION**
- 11296 None.
- 11297 **EXIT STATUS**
- 11298 The following exit values shall be returned:
- 11299 0 Successful completion.
- 11300 >0 An error occurred.
- 11301 **CONSEQUENCES OF ERRORS**
- 11302 Default.
- 11303 **APPLICATION USAGE**
- 11304 None.
- 11305 **EXAMPLES**
- 11306 None.
- 11307 **RATIONALE**
- 11308 None.
- 11309 **FUTURE DIRECTIONS**
- 11310 None.

11311 **SEE ALSO**

11312 *c99*

11313 **CHANGE HISTORY**

11314 First released in Issue 2.

11315 **Issue 5**

11316 In the SYNOPSIS, [-U *dir*] is changed to [-U *name*].

11317 **Issue 6**

11318 The APPLICATION USAGE section is added.

11319 **NAME**

11320           date — write the date and time

11321 **SYNOPSIS**

11322           date [-u] [+format]

11323 xSI        date [-u] *mmddhhmm*[[*cc*]*yy*]

11324

11325 **DESCRIPTION**

11326 xSI        The *date* utility shall write the date and time to standard output or attempt to set the system date  
 11327           and time. By default, the current date and time shall be written. If an operand beginning with  
 11328           '+' is specified, the output format of *date* shall be controlled by the conversion specifications  
 11329           and other text in the operand.

11330 **OPTIONS**

11331           The *date* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section  
 11332           12.2, Utility Syntax Guidelines.

11333           The following option shall be supported:

11334           **-u**        Perform operations as if the *TZ* environment variable was set to the string "UTC0",  
 11335                        or its equivalent historical value of "GMT0". Otherwise, *date* shall use the  
 11336                        timezone indicated by the *TZ* environment variable or the system default if that  
 11337                        variable is unset or null.

11338 **OPERANDS**

11339           The following operands shall be supported:

11340           **+format**    When the format is specified, each conversion specifier shall be replaced in the  
 11341                        standard output by its corresponding value. All other characters shall be copied to  
 11342                        the output without change. The output always shall be terminated with a  
 11343                        <newline>.

11344           **Conversion Specifications**

|       |    |                                                                          |  |
|-------|----|--------------------------------------------------------------------------|--|
| 11345 | %a | Locale's abbreviated weekday name.                                       |  |
| 11346 | %A | Locale's full weekday name.                                              |  |
| 11347 | %b | Locale's abbreviated month name.                                         |  |
| 11348 | %B | Locale's full month name.                                                |  |
| 11349 | %c | Locale's appropriate date and time representation.                       |  |
| 11350 | %C | Century (a year divided by 100 and truncated to an integer) as a decimal |  |
| 11351 |    | number [00,99].                                                          |  |
| 11352 | %d | Day of the month as a decimal number [01,31].                            |  |
| 11353 | %D | Date in the format <i>mm/dd/yy</i> .                                     |  |
| 11354 | %e | Day of the month as a decimal number [1,31] in a two-digit field with    |  |
| 11355 |    | leading space character fill.                                            |  |
| 11356 | %h | A synonym for %b.                                                        |  |
| 11357 | %H | Hour (24-hour clock) as a decimal number [00,23].                        |  |
| 11358 | %I | Hour (12-hour clock) as a decimal number [01,12].                        |  |

|       |     |                                                                                                           |  |
|-------|-----|-----------------------------------------------------------------------------------------------------------|--|
| 11359 | %j  | Day of the year as a decimal number [001,366].                                                            |  |
| 11360 | %m  | Month as a decimal number [01,12].                                                                        |  |
| 11361 | %M  | Minute as a decimal number [00,59].                                                                       |  |
| 11362 | %n  | A <newline>.                                                                                              |  |
| 11363 | %p  | Locale's equivalent of either AM or PM.                                                                   |  |
| 11364 | %r  | 12-hour clock time [01,12] using the AM/PM notation; in the POSIX                                         |  |
| 11365 |     | locale, this shall be equivalent to %I:%M:%S %p.                                                          |  |
| 11366 | %S  | Seconds as a decimal number [00,60].                                                                      |  |
| 11367 | %t  | A <tab>.                                                                                                  |  |
| 11368 | %T  | 24-hour clock time [00,23] in the format <i>HH:MM:SS</i> .                                                |  |
| 11369 | %u  | Weekday as a decimal number [1,7] (1=Monday).                                                             |  |
| 11370 | %U  | Week of the year (Sunday as the first day of the week) as a decimal                                       |  |
| 11371 |     | number [00,53]. All days in a new year preceding the first Sunday shall be                                |  |
| 11372 |     | considered to be in week 0.                                                                               |  |
| 11373 | %V  | Week of the year (Monday as the first day of the week) as a decimal                                       |  |
| 11374 |     | number [01,53]. If the week containing January 1 has four or more days in                                 |  |
| 11375 |     | the new year, then it shall be considered week 1; otherwise, it shall be the                              |  |
| 11376 |     | last week of the previous year, and the next week shall be week 1.                                        |  |
| 11377 | %w  | Weekday as a decimal number [0,6] (0=Sunday).                                                             |  |
| 11378 | %W  | Week of the year (Monday as the first day of the week) as a decimal                                       |  |
| 11379 |     | number [00,53]. All days in a new year preceding the first Monday shall                                   |  |
| 11380 |     | be considered to be in week 0.                                                                            |  |
| 11381 | %x  | Locale's appropriate date representation.                                                                 |  |
| 11382 | %X  | Locale's appropriate time representation.                                                                 |  |
| 11383 | %y  | Year within century [00,99].                                                                              |  |
| 11384 | %Y  | Year with century as a decimal number.                                                                    |  |
| 11385 | %Z  | Timezone name, or no characters if no timezone is determinable.                                           |  |
| 11386 | %%  | A percent sign character.                                                                                 |  |
| 11387 |     | See the Base Definitions volume of IEEE Std 1003.1-200x, Section 7.3.5, <i>LC_TIME</i>                    |  |
| 11388 |     | for the conversion specifier values in the POSIX locale.                                                  |  |
| 11389 |     | <b>Modified Conversion Specifications</b>                                                                 |  |
| 11390 |     | Some conversion specifiers can be modified by the <i>E</i> and <i>O</i> modifier characters to            |  |
| 11391 |     | indicate a different format or specification as specified in the <i>LC_TIME</i> locale                    |  |
| 11392 |     | description (see the Base Definitions volume of IEEE Std 1003.1-200x, Section 7.3.5,                      |  |
| 11393 |     | <i>LC_TIME</i> ). If the corresponding keyword (see <i>era</i> , <i>era_year</i> , <i>era_d_fmt</i> , and |  |
| 11394 |     | <i>alt_digits</i> in the Base Definitions volume of IEEE Std 1003.1-200x, Section 7.3.5,                  |  |
| 11395 |     | <i>LC_TIME</i> ) is not specified or not supported for the current locale, the unmodified                 |  |
| 11396 |     | conversion specifier value shall be used.                                                                 |  |
| 11397 | %Ec | Locale's alternative appropriate date and time representation.                                            |  |

|       |       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|-------|-------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 11398 | %EC   | The name of the base year (period) in the locale's alternative representation.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 11399 |       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 11400 | %Ex   | Locale's alternative date representation.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 11401 | %EX   | Locale's alternative time representation.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 11402 | %Ey   | Offset from %EC (year only) in the locale's alternative representation.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 11403 | %EY   | Full alternative year representation.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 11404 | %Od   | Day of month using the locale's alternative numeric symbols.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 11405 | %Oe   | Day of month using the locale's alternative numeric symbols.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 11406 | %OH   | Hour (24-hour clock) using the locale's alternative numeric symbols.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 11407 | %OI   | Hour (12-hour clock) using the locale's alternative numeric symbols.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 11408 | %Om   | Month using the locale's alternative numeric symbols.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 11409 | %OM   | Minutes using the locale's alternative numeric symbols.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 11410 | %OS   | Seconds using the locale's alternative numeric symbols.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 11411 | %Ou   | Weekday as a number in the locale's alternative representation (Monday = 1).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 11412 |       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 11413 | %OU   | Week number of the year (Sunday as the first day of the week) using the locale's alternative numeric symbols.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 11414 |       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 11415 | %OV   | Week number of the year (Monday as the first day of the week, rules corresponding to %v), using the locale's alternative numeric symbols.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 11416 |       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 11417 | %Ow   | Weekday as a number in the locale's alternative representation (Sunday = 0).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 11418 |       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 11419 | %OW   | Week number of the year (Monday as the first day of the week) using the locale's alternative numeric symbols.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 11420 |       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 11421 | %Oy   | Year (offset from %C) in alternative representation.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 11422 | XSI   | <b><i>mmddhhmm[[cc]yy ]</i></b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 11423 |       | Attempt to set the system date and time from the value given in the operand. This is only possible if the user has appropriate privileges and the system permits the setting of the system date and time. The first <i>mm</i> is the month (number); <i>dd</i> is the day (number); <i>hh</i> is the hour (number, 24-hour system); the second <i>mm</i> is the minute (number); <i>cc</i> is the century and is the first two digits of the year (this is optional); <i>yy</i> is the last two digits of the year and is optional. If century is not specified, then values in the range [69,99] shall refer to years 1969 to 1999 inclusive, and values in the range [00,68] shall refer to years 2000 to 2068 inclusive. The current year is the default if <i>yy</i> is omitted. |
| 11424 |       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 11425 |       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 11426 |       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 11427 |       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 11428 |       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 11429 |       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 11430 |       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 11431 |       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 11432 |       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 11433 |       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 11434 |       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 11435 | STDIN |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 11436 |       | Not used.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |

11437 **INPUT FILES**

11438       None.

11439 **ENVIRONMENT VARIABLES**11440       The following environment variables shall affect the execution of *date*:

11441       *LANG*       Provide a default value for the internationalization variables that are unset or null.  
 11442                   (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
 11443                   Internationalization Variables for the precedence of internationalization variables  
 11444                   used to determine the values of locale categories.)

11445       *LC\_ALL*      If set to a non-empty string value, override the values of all the other  
 11446                   internationalization variables.

11447       *LC\_CTYPE*   Determine the locale for the interpretation of sequences of bytes of text data as  
 11448                   characters (for example, single-byte as opposed to multi-byte characters in  
 11449                   arguments).

11450       *LC\_MESSAGES*

11451                   Determine the locale that should be used to affect the format and contents of  
 11452                   diagnostic messages written to standard error.

11453       *LC\_TIME*     Determine the format and contents of date and time strings written by *date*.

11454 XSI       *NLSPATH*   Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

11455       *TZ*           Determine the timezone in which the time and date are written, unless the *-u*  
 11456                   option is specified. If the *TZ* variable is unset or null and the *-u* is not specified, an  
 11457                   unspecified system default timezone is used.

11458 **ASYNCHRONOUS EVENTS**

11459       Default.

11460 **STDOUT**

11461       When no formatting operand is specified, the output in the POSIX locale shall be equivalent to  
 11462       specifying:

11463       date "+%a %b %e %H:%M:%S %Z %Y"

11464 **STDERR**

11465       The standard error shall be used only for diagnostic messages. |

11466 **OUTPUT FILES**

11467       None.

11468 **EXTENDED DESCRIPTION**

11469       None.

11470 **EXIT STATUS**

11471       The following exit values shall be returned:

11472       0   The date was written successfully.

11473       >0   An error occurred.

11474 **CONSEQUENCES OF ERRORS**

11475       Default.

## 11476 APPLICATION USAGE

11477 Conversion specifiers are of unspecified format when not in the POSIX locale. Some of them can  
 11478 contain <newline>s in some locales, so it may be difficult to use the format shown in standard  
 11479 output for parsing the output of *date* in those locales.

11480 The range of values for %S extends from 0 to 60 seconds to accommodate the occasional leap  
 11481 second.

11482 Although certain of the conversion specifiers in the POSIX locale (such as the name of the  
 11483 month) are shown with initial capital letters, this need not be the case in other locales. Programs  
 11484 using these fields may need to adjust the capitalization if the output is going to be used at the  
 11485 beginning of a sentence.

11486 The date string formatting capabilities are intended for use in Gregorian-style calendars,  
 11487 possibly with a different starting year (or years). The %x and %c conversion specifications,  
 11488 however, are intended for local representation; these may be based on a different, non-Gregorian  
 11489 calendar.

11490 The %C conversion specification was introduced to allow a fallback for the %EC (alternative year  
 11491 format base year); it can be viewed as the base of the current subdivision in the Gregorian  
 11492 calendar. The century number is calculated as the year divided by 100 and truncated to an  
 11493 integer; it should not be confused with the use of ordinal numbers for centuries (for example,  
 11494 "twenty-first century".) Both the %EY and %Y can then be viewed as the offset from %EC and %C,  
 11495 respectively.

11496 The E and O modifiers modify the traditional conversion specifiers, so that they can always be  
 11497 used, even if the implementation (or the current locale) does not support the modifier.

11498 The E modifier supports alternative date formats, such as the Japanese Emperor's Era, as long as  
 11499 these are based on the Gregorian calendar system. Extending the E modifiers to other date  
 11500 elements may provide an implementation-defined extension capable of supporting other  
 11501 calendar systems, especially in combination with the O modifier.

11502 The O modifier supports time and date formats using the locale's alternative numerical symbols,  
 11503 such as Kanji or Hindi digits or ordinal number representation.

11504 Non-European locales, whether they use Latin digits in computational items or not, often have  
 11505 local forms of the digits for use in date formats. This is not totally unknown even in Europe; a  
 11506 variant of dates uses Roman numerals for the months: the third day of September 1991 would be  
 11507 written as 3.IX.1991. In Japan, Kanji digits are regularly used for dates; in Arabic-speaking  
 11508 countries, Hindi digits are used. The %d, %e, %H, %I, %m, %S, %U, %w, %W, and %Y conversion  
 11509 specifications always return the date and time field in Latin digits (that is, 0 to 9). The %O  
 11510 modifier was introduced to support the use for display purposes of non-Latin digits. In the  
 11511 *LC\_TIME* category in *localedef*, the optional **alt\_digits** keyword is intended for this purpose. As  
 11512 an example, assume the following (partial) *localedef* source:

```
11513 alt_digits  "";"I";"II";"III";"IV";"V";"VI";"VII";"VIII" \
11514           "IX";"X";"XI";"XII"
11515 d_fmt      "%e.%Om.%Y"
```

11516 With the above date, the command:

```
11517 date "+%x"
```

11518 would yield 3.IX.1991. With the same *d\_fmt*, but without the **alt\_digits**, the command would  
 11519 yield 3.9.1991.

## 11520 EXAMPLES

11521 1. The following are input/output examples of *date* used at arbitrary times in the POSIX  
11522 locale:

11523 \$ date

11524 **Tue Jun 26 09:58:10 PDT 1990**

11525 \$ date "+DATE: %m/%d/%y%nTIME: %H:%M:%S"

11526 **DATE: 11/02/91**

11527 **TIME: 13:36:16**

11528 \$ date "+TIME: %r"

11529 **TIME: 01:36:32 PM**

11530 2. Examples for Denmark, where the default date and time format is %a %d %b %Y %T %Z:

11531 \$ LANG=da\_DK.iso\_8859-1 date

11532 **ons 02 okt 1991 15:03:32 CET**

11533 \$ LANG=da\_DK.iso\_8859-1 \

11534 date "+DATO: %A den %e. %B %Y%nKLOKKEN: %H:%M:%S" |

11535 **DATO: onsdag den 2. oktober 1991** |

11536 **KLOKKEN: 15:03:56**

11537 3. Examples for Germany, where the default date and time format is %a %d.%h.%Y, %T %Z:

11538 \$ LANG=De\_DE.88591 date

11539 **Mi 02.Okt.1991, 15:01:21 MEZ**

11540 \$ LANG=De\_DE.88591 date "+DATUM: %A, %d. %B %Y%nZEIT: %H:%M:%S" |

11541 **DATUM: Mittwoch, 02. Oktober 1991** |

11542 **ZEIT: 15:02:02**

11543 4. Examples for France, where the default date and time format is %a %d %h %Y %Z %T:

11544 \$ LANG=Fr\_FR.88591 date

11545 **Mer 02 oct 1991 MET 15:03:32**

11546 \$ LANG=Fr\_FR.88591 date "+JOUR: %A %d %B %Y%nHEURE: %H:%M:%S" |

11547 **JOUR: Mercredi 02 octobre 1991** |

11548 **HEURE: 15:03:56**

## 11549 RATIONALE

11550 Some of the new options for formatting are from the ISO C standard. The **-u** option was  
11551 introduced to allow portable access to Coordinated Universal Time (UTC). The string "GMT0" is  
11552 allowed as an equivalent TZ value to be compatible with all of the systems using the BSD  
11553 implementation, where this option originated.

11554 The %e format conversion specifications (adopted from System V) was added because the ISO C  
11555 standard conversion specifications did not provide any way to produce the historical default  
11556 *date* output during the first nine days of any month.

11557 There are two varieties of day and week numbering supported (in addition to any others created  
11558 with the locale-dependent %E and %O modifier characters):

- 11559 • The historical variety in which Sunday is the first day of the week and the weekdays
- 11560 preceding the first Sunday of the year are considered week 0. These are represented by %w
- 11561 and %U. A variant of this is %W, using Monday as the first day of the week, but still referring
- 11562 to week 0. This view of the calendar was retained because so many historical applications
- 11563 depend on it and the ISO C standard *strftime()* function, on which many *date*

- 11564 implementations are based, was defined in this way.
- 11565 • The international standard, based on the ISO 8601:2000 standard where Monday is the first  
11566 weekday and the algorithm for the first week number is more complex: If the week (Monday  
11567 to Sunday) containing January 1 has four or more days in the new year, then it is week 1;  
11568 otherwise, it is week 53 of the previous year, and the next week is week 1. These are  
11569 represented by the new conversion specifications %u and %V, added as a result of  
11570 international comments.
- 11571 **FUTURE DIRECTIONS**
- 11572 None.
- 11573 **SEE ALSO**
- 11574 The System Interfaces volume of IEEE Std 1003.1-200x, *printf()*, *strptime()*
- 11575 **CHANGE HISTORY**
- 11576 First released in Issue 2.
- 11577 **Issue 5**
- 11578 Changes are made for Year 2000 alignment.
- 11579 **Issue 6**
- 11580 The following new requirements on POSIX implementations derive from alignment with the  
11581 Single UNIX Specification:
- 11582 • The setting of system date and time is described, including how to interpret two-digit year  
11583 values if a century is not given.
- 11584 • The %EX modified conversion specification is added.
- 11585 The Open Group Corrigendum U048/2 is applied, correcting the examples.
- 11586 The DESCRIPTION is updated to refer to conversion specifications, instead of field descriptors  
11587 for consistency with the *LC\_TIME* category.
- 11588 A clarification is made such that the current year is the default if the *yy* argument is omitted  
11589 when setting the system date and time.

## 11590 NAME

11591 dd — convert and copy a file

## 11592 SYNOPSIS

11593 dd [*operand* ...]

## 11594 DESCRIPTION

11595 The *dd* utility shall copy the specified input file to the specified output file with possible  
 11596 conversions using specific input and output block sizes. It shall read the input one block at a  
 11597 time, using the specified input block size; it shall then process the block of data actually  
 11598 returned, which could be smaller than the requested block size. It shall apply any conversions  
 11599 that have been specified and write the resulting data to the output in blocks of the specified  
 11600 output block size. If the **bs=expr** operand is specified and no conversions other than **sync**,  
 11601 **noerror**, or **notrunc** are requested, the data returned from each input block shall be written as a  
 11602 separate output block; if the read returns less than a full block and the **sync** conversion is not  
 11603 specified, the resulting output block shall be the same size as the input block. If the **bs=expr**  
 11604 operand is not specified, or a conversion other than **sync**, **noerror**, or **notrunc** is requested, the  
 11605 input shall be processed and collected into full-sized output blocks until the end of the input is  
 11606 reached.

11607 The processing order shall be as follows:

- 11608 1. An input block is read.
- 11609 2. If the input block is shorter than the specified input block size and the **sync** conversion is  
 11610 specified, null bytes shall be appended to the input data up to the specified size. (If either  
 11611 **block** or **unblock** is also specified, <space>*s* shall be appended instead of null bytes.) The  
 11612 remaining conversions and output shall include the pad characters as if they had been read  
 11613 from the input.
- 11614 3. If the **bs=expr** operand is specified and no conversion other than **sync** or **noerror** is  
 11615 requested, the resulting data shall be written to the output as a single block, and the  
 11616 remaining steps are omitted.
- 11617 4. If the **swab** conversion is specified, each pair of input data bytes shall be swapped. If there  
 11618 is an odd number of bytes in the input block, the last byte in the input record shall not be  
 11619 swapped.
- 11620 5. Any remaining conversions (**block**, **unblock**, **lcase**, and **ucase**) shall be performed. These  
 11621 conversions shall operate on the input data independently of the input blocking; an input  
 11622 or output fixed-length record may span block boundaries.
- 11623 6. The data resulting from input or conversion or both shall be aggregated into output blocks  
 11624 of the specified size. After the end of input is reached, any remaining output shall be  
 11625 written as a block without padding if **conv=sync** is not specified; thus, the final output  
 11626 block may be shorter than the output block size.

## 11627 OPTIONS

11628 None.

## 11629 OPERANDS

11630 All of the operands shall be processed before any input is read. The following operands shall be  
 11631 supported:

- 11632 **if=file** Specify the input pathname; the default is standard input.
- 11633 **of=file** Specify the output pathname; the default is standard output. If the **seek=expr**  
 11634 conversion is not also specified, the output file shall be truncated before the copy |  
 11635 begins if an explicit **of=file** operand is specified, unless **conv=notrunc** is specified. |

|           |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|-----------|-------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 11636     |                               | If <b>seek=expr</b> is specified, but <b>conv=notrunc</b> is not, the effect of the copy shall be to preserve the blocks in the output file over which <i>dd</i> seeks, but no other portion of the output file shall be preserved. (If the size of the seek plus the size of the input file is less than the previous size of the output file, the output file shall be shortened by the copy.)                                                                                                                                                                                                                                               |
| 11637     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 11638     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 11639     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 11640     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 11641     | <b>ibs=expr</b>               | Specify the input block size, in bytes, by <i>expr</i> (default is 512).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 11642     | <b>obs=expr</b>               | Specify the output block size, in bytes, by <i>expr</i> (default is 512).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 11643     | <b>bs=expr</b>                | Set both input and output block sizes to <i>expr</i> bytes, superseding <b>ibs=</b> and <b>obs=</b> . If no conversion other than <b>sync</b> , <b>noerror</b> , and <b>notrunc</b> is specified, each input block shall be copied to the output as a single block without aggregating short blocks.                                                                                                                                                                                                                                                                                                                                           |
| 11644     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 11645     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 11646     | <b>cbs=expr</b>               | Specify the conversion block size for <b>block</b> and <b>unblock</b> in bytes by <i>expr</i> (default is zero). If <b>cbs=</b> is omitted or given a value of zero, using <b>block</b> or <b>unblock</b> produces unspecified results.                                                                                                                                                                                                                                                                                                                                                                                                        |
| 11647     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 11648     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 11649 XSI |                               | The application shall ensure that this operand is also specified if the <b>conv=</b> operand is specified with a value of <b>ascii</b> , <b>ebcdic</b> , or <b>ibm</b> . For a <b>conv=</b> operand with an <b>ascii</b> value, the input is handled as described for the <b>unblock</b> value, except that characters are converted to ASCII before any trailing <space>s are deleted. For <b>conv=</b> operands with <b>ebcdic</b> or <b>ibm</b> values, the input is handled as described for the <b>block</b> value except that the characters are converted to EBCDIC or IBM EBCDIC, respectively, after any trailing <space>s are added. |
| 11650     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 11651     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 11652     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 11653     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 11654     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 11655     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 11656     | <b>skip=n</b>                 | Skip <i>n</i> input blocks (using the specified input block size) before starting to copy. On seekable files, the implementation shall read the blocks or seek past them; on non-seekable files, the blocks shall be read and the data shall be discarded.                                                                                                                                                                                                                                                                                                                                                                                     |
| 11657     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 11658     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 11659     | <b>seek=n</b>                 | Skip <i>n</i> blocks (using the specified output block size) from beginning of the output file before copying. On non-seekable files, existing blocks shall be read and space from the current end-of-file to the specified offset, if any, filled with null bytes; on seekable files, the implementation shall seek to the specified offset or read the blocks as described for non-seekable files.                                                                                                                                                                                                                                           |
| 11660     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 11661     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 11662     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 11663     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 11664     | <b>count=n</b>                | Copy only <i>n</i> input blocks.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 11665     | <b>conv=value[,value ...]</b> |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 11666     |                               | Where <i>values</i> are comma-separated symbols from the following list:                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 11667 XSI | <b>ascii</b>                  | Convert EBCDIC to ASCII; see Table 4-6 (on page 2506).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 11668 XSI | <b>ebcdic</b>                 | Convert ASCII to EBCDIC; see Table 4-6 (on page 2506).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 11669 XSI | <b>ibm</b>                    | Convert ASCII to a different EBCDIC set; see Table 4-7 (on page 2507).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 11670     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 11671     |                               | The <b>ascii</b> , <b>ebcdic</b> , and <b>ibm</b> values are mutually-exclusive.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 11672     | <b>block</b>                  | Treat the input as a sequence of <newline>-terminated or end-of-file-terminated variable-length records independent of the input block boundaries. Each record shall be converted to a record with a fixed length specified by the conversion block size. Any <newline> shall be removed from the input line; <space>s shall be appended to lines that are shorter than their conversion block size to fill the block. Lines that are longer than the conversion block size shall be truncated to the largest number of characters that fit into that size; the number of truncated lines shall be reported (see the STDERR                    |
| 11673     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 11674     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 11675     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 11676     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 11677     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 11678     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 11679     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 11680     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |

|       |                |                                                                                                                       |
|-------|----------------|-----------------------------------------------------------------------------------------------------------------------|
| 11681 |                | section).                                                                                                             |
| 11682 |                | The <b>block</b> and <b>unblock</b> values are mutually-exclusive.                                                    |
| 11683 | <b>unblock</b> | Convert fixed-length records to variable length. Read a number of                                                     |
| 11684 |                | bytes equal to the conversion block size (or the number of bytes                                                      |
| 11685 |                | remaining in the input, if less than the conversion block size), delete                                               |
| 11686 |                | all trailing <space>s, and append a <newline>.                                                                        |
| 11687 | <b>lcase</b>   | Map uppercase characters specified by the <i>LC_CTYPE</i> keyword                                                     |
| 11688 |                | <b>tolower</b> to the corresponding lowercase character. Characters for                                               |
| 11689 |                | which no mapping is specified shall not be modified by this                                                           |
| 11690 |                | conversion.                                                                                                           |
| 11691 |                | The <b>lcase</b> and <b>ucase</b> symbols are mutually-exclusive.                                                     |
| 11692 | <b>ucase</b>   | Map lowercase characters specified by the <i>LC_CTYPE</i> keyword                                                     |
| 11693 |                | <b>toupper</b> to the corresponding uppercase character. Characters for                                               |
| 11694 |                | which no mapping is specified shall not be modified by this                                                           |
| 11695 |                | conversion.                                                                                                           |
| 11696 | <b>swab</b>    | Swap every pair of input bytes.                                                                                       |
| 11697 | <b>noerror</b> | Do not stop processing on an input error. When an input error                                                         |
| 11698 |                | occurs, a diagnostic message shall be written on standard error,                                                      |
| 11699 |                | followed by the current input and output block counts in the same                                                     |
| 11700 |                | format as used at completion (see the <i>STDERR</i> section). If the <b>sync</b>                                      |
| 11701 |                | conversion is specified, the missing input shall be replaced with null                                                |
| 11702 |                | bytes and processed normally; otherwise, the input block shall be                                                     |
| 11703 |                | omitted from the output.                                                                                              |
| 11704 | <b>notrunc</b> | Do not truncate the output file. Preserve blocks in the output file not                                               |
| 11705 |                | explicitly written by this invocation of the <i>dd</i> utility. (See also the                                         |
| 11706 |                | preceding <b>of=file</b> operand.)                                                                                    |
| 11707 | <b>sync</b>    | Pad every input block to the size of the <b>ibs=</b> buffer, appending null                                           |
| 11708 |                | bytes. (If either <b>block</b> or <b>unblock</b> is also specified, append <space>s,                                  |
| 11709 |                | rather than null bytes.)                                                                                              |
| 11710 |                | The behavior is unspecified if operands other than <b>conv=</b> are specified more than once.                         |
| 11711 |                | For the <b>bs=</b> , <b>cbs=</b> , <b>ibs=</b> , and <b>obs=</b> operands, the application shall supply an expression |
| 11712 |                | specifying a size in bytes. The expression, <i>expr</i> , can be:                                                     |
| 11713 |                | 1. A positive decimal number                                                                                          |
| 11714 |                | 2. A positive decimal number followed by <i>k</i> , specifying multiplication by 1 024                                |
| 11715 |                | 3. A positive decimal number followed by <i>b</i> , specifying multiplication by 512                                  |
| 11716 |                | 4. Two or more positive decimal numbers (with or without <i>k</i> or <i>b</i> ) separated by <i>x</i> , specifying    |
| 11717 |                | the product of the indicated values                                                                                   |
| 11718 |                | All of the operands are processed before any input is read.                                                           |
| 11719 | XSI            | The following two tables display the octal number character values used for the <b>ascii</b> and <b>ebcdic</b>        |
| 11720 |                | conversions (first table) and for the <b>ibm</b> conversion (second table). In both tables, the ASCII                 |
| 11721 |                | values are the row and column headers and the EBCDIC values are found at their intersections.                         |
| 11722 |                | For example, ASCII 0012 (LF) is the second row, third column, yielding 0045 in EBCDIC. The                            |
| 11723 |                | inverted tables (for EBCDIC to ASCII conversion) are not shown, but are in one-to-one                                 |
| 11724 |                | correspondence with these tables. The differences between the two tables are highlighted by                           |

11725

11726

11727

small boxes drawn around five entries.

Table 4-6 ASCII to EBCDIC Conversion

|             | 0        | 1        | 2        | 3        | 4        | 5        | 6        | 7        |
|-------------|----------|----------|----------|----------|----------|----------|----------|----------|
| <b>0000</b> | 0000 NUL | 0001 SOH | 0002 STX | 0003 ETX | 0067 EOT | 0055 ENQ | 0056 ACK | 0057 BEL |
| <b>0010</b> | 0026 BS  | 0005 HT  | 0045 LF  | 0013 VT  | 0014 FF  | 0015 CR  | 0016 SO  | 0017 SI  |
| <b>0020</b> | 0020 DLE | 0021 DC1 | 0022 DC2 | 0023 DC3 | 0074 DC4 | 0075 NAK | 0062 SYN | 0046 ETB |
| <b>0030</b> | 0030 CAN | 0031 EM  | 0077 SUB | 0047 ESC | 0034 IFS | 0035 IGS | 0036 IRS | 0037 ITB |
| <b>0040</b> | 0100 Sp  | 0132 !   | 0177 "   | 0173 #   | 0133 \$  | 0154 %   | 0120 &   | 0175 '   |
| <b>0050</b> | 0115 (   | 0135 )   | 0134 *   | 0116 +   | 0153 ,   | 0140 -   | 0113 .   | 0141 /   |
| <b>0060</b> | 0360 0   | 0361 1   | 0362 2   | 0363 3   | 0364 4   | 0365 5   | 0366 6   | 0367 7   |
| <b>0070</b> | 0370 8   | 0371 9   | 0172 :   | 0136 ;   | 0114 <   | 0176 =   | 0156 >   | 0157 ?   |
| <b>0100</b> | 0174 @   | 0301 A   | 0302 B   | 0303 C   | 0304 D   | 0305 E   | 0306 F   | 0307 G   |
| <b>0110</b> | 0310 H   | 0311 I   | 0321 J   | 0322 K   | 0323 L   | 0324 M   | 0325 N   | 0326 O   |
| <b>0120</b> | 0327 P   | 0330 Q   | 0331 R   | 0342 S   | 0343 T   | 0344 U   | 0345 V   | 0346 W   |
| <b>0130</b> | 0347 X   | 0350 Y   | 0351 Z   | 0255 [   | 0340 \   | 0275 ]   | 0232     | 0155 _   |
| <b>0140</b> | 0171 `   | 0201 a   | 0202 b   | 0203 c   | 0204 d   | 0205 e   | 0206 f   | 0207 g   |
| <b>0150</b> | 0210 h   | 0211 i   | 0221 j   | 0222 k   | 0223 ]   | 0224 m   | 0225 n   | 0226 o   |
| <b>0160</b> | 0227 p   | 0230 q   | 0231 r   | 0242 s   | 0243 t   | 0244 u   | 0245 v   | 0246 w   |
| <b>0170</b> | 0247 x   | 0250 y   | 0251 z   | 0300 {   | 0117     | 0320 }   | 0137 ~   | 0007 DEL |
| <b>0200</b> | 0040 DS  | 0041 SOS | 0042 FS  | 0043 WUS | 0044 BYP | 0025 NL  | 0006 RNL | 0027 POC |
| <b>0210</b> | 0050 SA  | 0051 SFE | 0052 SM  | 0053 CSP | 0054 MFA | 0011 SPS | 0012 RPT | 0033 CU1 |
| <b>0220</b> | 0060     | 0061     | 0032 UBS | 0063 IR  | 0064 PP  | 0065 TRN | 0066 NBS | 0010 GE  |
| <b>0230</b> | 0070 SBS | 0071 IT  | 0072 RFF | 0073 CU3 | 0004 SEL | 0024 RES | 0076     | 0341     |
| <b>0240</b> | 0101     | 0102     | 0103     | 0104     | 0105     | 0106     | 0107     | 0110     |
| <b>0250</b> | 0111     | 0121     | 0122     | 0123     | 0124     | 0125     | 0126     | 0127     |
| <b>0260</b> | 0130     | 0131     | 0142     | 0143     | 0144     | 0145     | 0146     | 0147     |
| <b>0270</b> | 0150     | 0151     | 0160     | 0161     | 0162     | 0163     | 0164     | 0165     |
| <b>0300</b> | 0166     | 0167     | 0170     | 0200     | 0212     | 0213     | 0214     | 0215     |
| <b>0310</b> | 0216     | 0217     | 0220     | 0152 ;   | 0233     | 0234     | 0235     | 0236     |
| <b>0320</b> | 0237     | 0240     | 0252     | 0253     | 0254     | 0112 ¢   | 0256     | 0257     |
| <b>0330</b> | 0260     | 0261     | 0262     | 0263     | 0264     | 0265     | 0266     | 0267     |
| <b>0340</b> | 0270     | 0271     | 0272     | 0273     | 0274     | 0241     | 0276     | 0277     |
| <b>0350</b> | 0312     | 0313     | 0314 ¶   | 0315     | 0316 ¶   | 0317     | 0332     | 0333     |
| <b>0360</b> | 0334     | 0335     | 0336     | 0337     | 0352     | 0353     | 0354 H   | 0355     |
| <b>0370</b> | 0356     | 0357     | 0372     | 0373     | 0374     | 0375     | 0376     | 0377 EO  |

Table 4-7 ASCII to IBM EBCDIC Conversion

|             | 0        | 1        | 2        | 3        | 4        | 5        | 6        | 7        |
|-------------|----------|----------|----------|----------|----------|----------|----------|----------|
| <b>0000</b> | 0000 NUL | 0001 SOH | 0002 STX | 0003 ETX | 0067 EOT | 0055 ENQ | 0056 ACK | 0057 BEL |
| <b>0010</b> | 0026 BS  | 0005 HT  | 0045 LF  | 0013 VT  | 0014 FF  | 0015 CR  | 0016 SO  | 0017 SI  |
| <b>0020</b> | 0020 DLE | 0021 DC1 | 0022 DC2 | 0023 DC3 | 0074 DC4 | 0075 NAK | 0062 SYN | 0046 ETB |
| <b>0030</b> | 0030 CAN | 0031 EM  | 0077 SUB | 0047 ESC | 0034 IFS | 0035 IGS | 0036 IRS | 0037 ITB |
| <b>0040</b> | 0100 Sp  | 0132 !   | 0177 "   | 0173 #   | 0133 \$  | 0154 %   | 0120 &   | 0175 '   |
| <b>0050</b> | 0115 (   | 0135 )   | 0134 *   | 0116 +   | 0153 ,   | 0140 -   | 0113 .   | 0141 /   |
| <b>0060</b> | 0360 0   | 0361 1   | 0362 2   | 0363 3   | 0364 4   | 0365 5   | 0366 6   | 0367 7   |
| <b>0070</b> | 0370 8   | 0371 9   | 0172 :   | 0136 ;   | 0114 <   | 0176 =   | 0156 >   | 0157 ?   |
| <b>0100</b> | 0174 @   | 0301 A   | 0302 B   | 0303 C   | 0304 D   | 0305 E   | 0306 F   | 0307 G   |
| <b>0110</b> | 0310 H   | 0311 I   | 0321 J   | 0322 K   | 0323 L   | 0324 M   | 0325 N   | 0326 O   |
| <b>0120</b> | 0327 P   | 0330 Q   | 0331 R   | 0342 S   | 0343 T   | 0344 U   | 0345 V   | 0346 W   |
| <b>0130</b> | 0347 X   | 0350 Y   | 0351 Z   | 0255 [   | 0340 \   | 0275 ]   | 0137 _   | 0155 _   |
| <b>0140</b> | 0171 `   | 0201 a   | 0202 b   | 0203 c   | 0204 d   | 0205 e   | 0206 f   | 0207 g   |
| <b>0150</b> | 0210 h   | 0211 i   | 0221 j   | 0222 k   | 0223 ]   | 0224 m   | 0225 n   | 0226 o   |
| <b>0160</b> | 0227 p   | 0230 q   | 0231 r   | 0242 s   | 0243 t   | 0244 u   | 0245 v   | 0246 w   |
| <b>0170</b> | 0247 x   | 0250 y   | 0251 z   | 0300 {   | 0117     | 0320 }   | 0241     | 0007 DEL |
| <b>0200</b> | 0040 DS  | 0041 SOS | 0042 FS  | 0043 WUS | 0044 BYP | 0025 NL  | 0006 RNL | 0027 POC |
| <b>0210</b> | 0050 SA  | 0051 SFE | 0052 SM  | 0053 CSP | 0054 MFA | 0011 SPS | 0012 RPT | 0033 CU1 |
| <b>0220</b> | 0060     | 0061     | 0032 UBS | 0063 IR  | 0064 PP  | 0065 TRN | 0066 NBS | 0010 GE  |
| <b>0230</b> | 0070 SBS | 0071 IT  | 0072 RFF | 0073 CU3 | 0004 SEL | 0024 RES | 0076     | 0341     |
| <b>0240</b> | 0101     | 0102     | 0103     | 0104     | 0105     | 0106     | 0107     | 0110     |
| <b>0250</b> | 0111     | 0121     | 0122     | 0123     | 0124     | 0125     | 0126     | 0127     |
| <b>0260</b> | 0130     | 0131     | 0142     | 0143     | 0144     | 0145     | 0146     | 0147     |
| <b>0270</b> | 0150     | 0151     | 0160     | 0161     | 0162     | 0163     | 0164     | 0165     |
| <b>0300</b> | 0166     | 0167     | 0170     | 0200     | 0212     | 0213     | 0214     | 0215     |
| <b>0310</b> | 0216     | 0217     | 0220     | 0232     | 0233     | 0234     | 0235     | 0236     |
| <b>0320</b> | 0237     | 0240     | 0252     | 0253     | 0254     | 0255 [   | 0256     | 0257     |
| <b>0330</b> | 0260     | 0261     | 0262     | 0263     | 0264     | 0265     | 0266     | 0267     |
| <b>0340</b> | 0270     | 0271     | 0272     | 0273     | 0274     | 0275 ]   | 0276     | 0277     |
| <b>0350</b> | 0312     | 0313     | 0314 J   | 0315     | 0316 Y   | 0317     | 0332     | 0333     |
| <b>0360</b> | 0334     | 0335     | 0336     | 0337     | 0352     | 0353     | 0354 H   | 0355     |
| <b>0370</b> | 0356     | 0357     | 0372     | 0373     | 0374     | 0375     | 0376     | 0377 EO  |

11730 **STDIN**

11731 If no **if=** operand is specified, the standard input shall be used. See the INPUT FILES section.

11732 **INPUT FILES**

11733 The input file can be any file type.

11734 **ENVIRONMENT VARIABLES**

11735 The following environment variables shall affect the execution of *dd*:

11736 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 11737 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
 11738 Internationalization Variables for the precedence of internationalization variables  
 11739 used to determine the values of locale categories.)

11740 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 11741 internationalization variables.

11742 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 11743 characters (for example, single-byte as opposed to multi-byte characters in  
 11744 arguments and input files), the classification of characters as uppercase or  
 11745 lowercase, and the mapping of characters from one case to the other.

11746 **LC\_MESSAGES**

11747 Determine the locale that should be used to affect the format and contents of  
 11748 diagnostic messages written to standard error and informative messages written to  
 11749 standard output.

11750 **XS1** **NLS\_PATH** Determine the location of message catalogs for the processing of **LC\_MESSAGES**.

11751 **ASYNCHRONOUS EVENTS**

11752 For SIGINT, the *dd* utility shall interrupt its current processing, write status information to  
 11753 standard error, and exit as though terminated by SIGINT. It shall take the standard action for all  
 11754 other signals; see the ASYNCHRONOUS EVENTS section in Section 1.11 (on page 2221).

11755 **STDOUT**

11756 If no **of=** operand is specified, the standard output shall be used. The nature of the output  
 11757 depends on the operands selected.

11758 **STDERR**

11759 On completion, *dd* shall write the number of input and output blocks to standard error. In the  
 11760 POSIX locale the following formats shall be used:

11761 "%u+%u records in\n", <number of whole input blocks>,  
 11762 <number of partial input blocks>

11763 "%u+%u records out\n", <number of whole output blocks>,  
 11764 <number of partial output blocks>

11765 A partial input block is one for which *read()* returned less than the input block size. A partial  
 11766 output block is one that was written with fewer bytes than specified by the output block size.

11767 In addition, when there is at least one truncated block, the number of truncated blocks shall be  
 11768 written to standard error. In the POSIX locale, the format shall be:

11769 "%u truncated %s\n", <number of truncated blocks>, "record" (if  
 11770 <number of truncated blocks> is one) "records" (otherwise)

11771 Diagnostic messages may also be written to standard error.

11772 **OUTPUT FILES**

11773 If the **of=** operand is used, the output shall be the same as described in the STDOUT section.

11774 **EXTENDED DESCRIPTION**

11775 None.

11776 **EXIT STATUS**

11777 The following exit values shall be returned:

11778 0 The input file was copied successfully.

11779 >0 An error occurred.

11780 **CONSEQUENCES OF ERRORS**

11781 If an input error is detected and the **noerror** conversion has not been specified, any partial  
 11782 output block shall be written to the output file, a diagnostic message shall be written, and the  
 11783 copy operation shall be discontinued. If some other error is detected, a diagnostic message shall  
 11784 be written and the copy operation shall be discontinued.

11785 **APPLICATION USAGE**

11786 The input and output block size can be specified to take advantage of raw physical I/O.

11787 There are many different versions of the EBCDIC codesets. The ASCII and EBCDIC conversions  
 11788 specified for the *dd* utility perform conversions for the version specified by the tables.

11789 **EXAMPLES**

11790 The following command:

```
11791 dd if=/dev/rmt0h of=/dev/rmt1h
```

11792 copies from tape drive 0 to tape drive 1, using a common historical device naming convention.

11793 The following command:

```
11794 dd ibs=10 skip=1
```

11795 strips the first 10 bytes from standard input.

11796 This example reads an EBCDIC tape blocked ten 80-byte EBCDIC card images per block into the  
 11797 ASCII file **x**:

```
11798 dd if=/dev/tape of=x ibs=800 cbs=80 conv=ascii,lcase
```

11799 **RATIONALE**

11800 The OPTIONS section is listed as “None” because there are no options recognized by historical  
 11801 *dd* utilities. Certainly, many of the operands could have been designed to use the Utility Syntax  
 11802 Guidelines, which would have resulted in the classic hyphenated option letters. In this version  
 11803 of this volume of IEEE Std 1003.1-200x, *dd* retains its curious JCL-like syntax due to the large  
 11804 number of applications that depend on the historical implementation.

11805 A suggested implementation technique for **conv=noerror,sync** is to zero (or <space>-fill, if  
 11806 **blocking** or **unblocking**) the input buffer before each read and to write the contents of the input  
 11807 buffer to the output even after an error. In this manner, any data transferred to the input buffer  
 11808 before the error was detected is preserved. Another point is that a failed read on a regular file or  
 11809 a disk generally does not increment the file offset, and *dd* must then seek past the block on which  
 11810 the error occurred; otherwise, the input error occurs repetitively. When the input is a magnetic  
 11811 tape, however, the tape normally has passed the block containing the error when the error is  
 11812 reported, and thus no seek is necessary.

11813 The default **ibs=** and **obs=** sizes are specified as 512 bytes because there are historical (largely  
 11814 portable) scripts that assume these values. If they were left unspecified, unusual results could

- 11815 occur if an implementation chose an odd block size.
- 11816 Historical implementations of *dd* used *creat()* when processing *of=file*. This makes the *seek=*  
 11817 operand unusable except on special files. The *conv=notrunc* feature was added because more  
 11818 recent BSD-based implementations use *open()* (without *O\_TRUNC*) instead of *creat()*, but they  
 11819 fail to delete output file contents after the data copied.
- 11820 The *w* multiplier (historically meaning *word*), is used in System V to mean 2 and in 4.2 BSD to  
 11821 mean 4. Since *word* is inherently non-portable, its use is not supported by this volume of  
 11822 IEEE Std 1003.1-200x.
- 11823 Standard EBCDIC does not have the characters ' [ ' and ' ] '. The values used in the table are  
 11824 taken from a common print train that does contain them. Other than those characters, the print  
 11825 train values are not filled in, but appear to provide some of the motivation for the historical  
 11826 choice of translations reflected here.
- 11827 The Standard EBCDIC table provides a 1:1 translation for all 256 bytes.
- 11828 The IBM EBCDIC table does not provide such a translation. The marked cells in the tables differ  
 11829 in such a way that:
- 11830 1. EBCDIC 0112 ( ' ϕ ' ) and 0152 (broken pipe) do not appear in the table.
  - 11831 2. EBCDIC 0137 ( ' ¬ ' ) translates to/from ASCII 0236 ( ' ^ ' ). In the standard table, EBCDIC  
 11832 0232 (no graphic) is used.
  - 11833 3. EBCDIC 0241 ( ' ~ ' ) translates to/from ASCII 0176 ( ' ~ ' ). In the standard table, EBCDIC  
 11834 0137 ( ' ¬ ' ) is used.
  - 11835 4. 0255 ( ' [ ' ) and 0275 ( ' ] ' ) appear twice, once in the same place as for the standard table  
 11836 and once in place of 0112 ( ' ϕ ' ) and 0241 ( ' ~ ' ).
- 11837 In net result:
- 11838 EBCDIC 0275 ( ' ] ' ) displaced EBCDIC 0241 ( ' ~ ' ) in cell 0345. |
- 11839 That displaced EBCDIC 0137 ( ' ¬ ' ) in cell 0176. |
- 11840 That displaced EBCDIC 0232 (no graphic) in cell 0136. |
- 11841 That replaced EBCDIC 0152 (broken pipe) in cell 0313. |
- 11842 EBCDIC 0255 ( ' [ ' ) replaced EBCDIC 0112 ( ' ϕ ' ). |
- 11843 This translation, however, reflects historical practice that (ASCII) ' ~ ' and ' ¬ ' were often  
 11844 mapped to each other, as were ' [ ' and ' ϕ ' ; and ' ] ' and (EBCDIC) ' ~ ' .
- 11845 The **chs** operand is required if any of the **ascii**, **ebcdic**, or **ibm** operands are specified. For the  
 11846 **ascii** operand, the input is handled as described for the **unblock** operand except that characters  
 11847 are converted to ASCII before the trailing <space>s are deleted. For the **ebcdic** and **ibm**  
 11848 operands, the input is handled as described for the **block** operand except that the characters are  
 11849 converted to EBCDIC or IBM EBCDIC after the trailing <space>s are added.
- 11850 The **block** and **unblock** keywords are from historical BSD practice.
- 11851 The consistent use of the word **record** in standard error messages matches most historical  
 11852 practice. An earlier version of System V used **block**, but this has been updated in more recent  
 11853 releases.
- 11854 Early proposals only allowed two numbers separated by **x** to be used in a product when  
 11855 specifying **bs=**, **chs=**, **ibs=**, and **obs=** sizes. This was changed to reflect the historical practice of  
 11856 allowing multiple numbers in the product as provided by Version 7 and all releases of System V

- 11857 and BSD.
- 11858 A change to the *swab* conversion is required to match historical practice and is the result of IEEE  
11859 PASC Interpretation 1003.2 #03 and #04, submitted for the ISO POSIX-2: 1993 standard.
- 11860 A change to the handling of SIGINT is required to match historical practice and is the result of  
11861 IEEE PASC Interpretation 1003.2 #06 submitted for the ISO POSIX-2: 1993 standard.
- 11862 **FUTURE DIRECTIONS**
- 11863 None.
- 11864 **SEE ALSO**
- 11865 *sed, tr*
- 11866 **CHANGE HISTORY**
- 11867 First released in Issue 2.
- 11868 **Issue 5**
- 11869 The second paragraph of the **cbs=** description is reworded and marked EX.
- 11870 **FUTURE DIRECTIONS** section added.
- 11871 **Issue 6**
- 11872 Changes are made to *swab* conversion and SIGINT handling to align with the IEEE P1003.2b  
11873 draft standard.
- 11874 The normative text is reworded to avoid use of the term “must” for application requirements. |
- 11875 IEEE PASC Interpretation 1003.2 #209 is applied, clarifying the interaction between *dd of=file* and |
- 11876 **conv=notrunc**. |

## 11877 NAME

11878 delta — make a delta (change) to an SCCS file (**DEVELOPMENT**)

## 11879 SYNOPSIS

```
11880 xSI delta [-nps][-g list][-m mrlist][-r SID][-y[comment]] file...
```

11881

## 11882 DESCRIPTION

11883 The *delta* utility shall be used to permanently introduce into the named SCCS files changes that  
11884 were made to the files retrieved by *get* (called the *g-files*, or generated files).

## 11885 OPTIONS

11886 The *delta* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section  
11887 12.2, Utility Syntax Guidelines, except that the *-y* option has an optional option-argument. This  
11888 optional option-argument shall not be presented as a separate argument.

11889 The following options shall be supported:

11890 *-r SID* Uniquely identify which delta is to be made to the SCCS file. The use of this option |  
11891 shall be necessary only if two or more outstanding *get* commands for editing (*get* |  
11892 *-e*) on the same SCCS file were done by the same person (login name). The SID |  
11893 value specified with the *-r* option can be either the SID specified on the *get* |  
11894 command line or the SID to be made as reported by the *get* utility; see *get* (on page |  
11895 2675).

11896 *-s* Suppress the report to standard output of the activity associated with each *file*.  
11897 See the STDOUT section.

11898 *-n* Specify retention of the edited *g-file* (normally removed at completion of delta  
11899 processing).

11900 *-g list* Specify a *list*, (see *get* (on page 2675) for the definition of *list*) of deltas that shall be  
11901 ignored when the file is accessed at the change level (SID) created by this delta.

11902 *-m mrlist* Specify a modification request (MR) number that the application shall supply as |  
11903 the reason for creating the new delta. This shall be used if the SCCS file has the *v* |  
11904 flag set; see *admin* (on page 2328).

11905 If *-m* is not used and *'-'* is not specified as a file argument, and the standard |  
11906 input is a terminal, the prompt described in the STDOUT section shall be written |  
11907 to standard output before the standard input is read; if the standard input is not a |  
11908 terminal, no prompt shall be issued.

11909 MRs in a list shall be separated by <blank>s or escaped <newline>s. An |  
11910 unescaped <newline> shall terminate the MR list. The escape character is |  
11911 <backslash>.

11912 If the *v* flag has a value, it shall be taken to be the name of a program which |  
11913 validates the correctness of the MR numbers. If a non-zero exit status is returned  
11914 from the MR number validation program, the *delta* utility shall terminate. (It is  
11915 assumed that the MR numbers were not all valid.)

11916 *-y[comment]* Describe the reason for making the delta. The *comment* shall be an arbitrary group  
11917 of lines that would meet the definition of a text file. Implementations shall support  
11918 *comments* from zero to 512 bytes and may support longer values. A null string  
11919 (specified as either *-y*, *-y" "*, or in response to a prompt for a comment) shall be |  
11920 considered a valid *comment*.

11921 If `-y` is not specified and `'-'` is not specified as a file argument, and the standard  
 11922 input is a terminal, the prompt described in the `STDOUT` section shall be written  
 11923 to standard output before the standard input is read; if the standard input is not a  
 11924 terminal, no prompt shall be issued. An unescaped `<newline>` shall terminate the  
 11925 comment text. The escape character is `<backslash>`.

11926 The `-y` option shall be required if the *file* operand is specified as `'-'`.

11927 **-p** Write (to standard output) the SCCS file differences before and after the delta is  
 11928 applied in *diff* format; see *diff* (on page 2520).

#### 11929 OPERANDS

11930 The following operand shall be supported:

11931 *file* A pathname of an existing SCCS file or a directory. If *file* is a directory, the *delta*  
 11932 utility shall behave as though each file in the directory were specified as a named  
 11933 file, except that non-SCCS files (last component of the pathname does not begin  
 11934 with `s.`) and unreadable files shall be silently ignored.

11935 If exactly one *file* operand appears, and it is `'-'`, the standard input shall be read;  
 11936 each line of the standard input shall be taken to be the name of an SCCS file to be  
 11937 processed. Non-SCCS files and unreadable files shall be silently ignored.

#### 11938 STDIN

11939 The standard input shall be a text file used only in the following cases:

- 11940 • To read an *mrlist* or a *comment* (see the `-m` and `-y` options).
- 11941 • A *file* operand shall be specified as `'-'`. In this case, the `-y` option must be used to specify  
 11942 the comment, and if the SCCS file has the `v` flag set, the `-m` option must also be used to  
 11943 specify the MR list.

#### 11944 INPUT FILES

11945 Input files shall be text files whose data is to be included in the SCCS files. If the first character of  
 11946 any line of an input file is `<SOH>` in the POSIX locale, the results are unspecified. If this file  
 11947 contains more than 99 999 lines, the number of lines recorded in the header for this file shall be  
 11948 99 999 for this delta.

#### 11949 ENVIRONMENT VARIABLES

11950 The following environment variables shall affect the execution of *delta*:

11951 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 11952 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
 11953 Internationalization Variables for the precedence of internationalization variables  
 11954 used to determine the values of locale categories.)

11955 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 11956 internationalization variables.

11957 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 11958 characters (for example, single-byte as opposed to multi-byte characters in  
 11959 arguments and input files).

11960 *LC\_MESSAGES*

11961 Determine the locale that should be used to affect the format and contents of  
 11962 diagnostic messages written to standard error, and informative messages written  
 11963 to standard output.

11964 *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

- 11965            *TZ*            Determine the timezone in which the time and date are written in the SCCS file. If |  
11966                            the *TZ* variable is unset or NULL, an unspecified system default timezone is used. |
- 11967 **ASYNCHRONOUS EVENTS**
- 11968            If SIGINT is caught, temporary files shall be cleaned up and *delta* shall exit with a non-zero exit |  
11969            code. The standard action shall be taken for all other signals; see Section 1.11 (on page 2221). |
- 11970 **STDOUT**
- 11971            The standard output shall be used only for the following messages in the POSIX locale:
- 11972            • Prompts (see the *-m* and *-y* options) in the following formats:
- 11973                    "MRs? "
- 11974                    "comments? "
- 11975            The MR prompt, if written, shall always precede the comments prompt.
- 11976            • A report of each *file*'s activities (unless the *-s* option is specified) in the following format:
- 11977                    "%s\n%d inserted\n%d deleted\n%d unchanged\n", <New SID>,  
11978                    <number of lines inserted>, <number of lines deleted>,  
11979                    <number of lines unchanged>
- 11980 **STDERR**
- 11981            The standard error shall be used only for diagnostic messages. |
- 11982 **OUTPUT FILES**
- 11983            Any SCCS files updated shall be files of an unspecified format. |
- 11984 **EXTENDED DESCRIPTION**
- 11985            **System Date and Time** |
- 11986            When a *delta* is added to an SCCS file, the system date and time shall be recorded for the new |  
11987            *delta*. If a *get* is performed using an SCCS file with a date recorded apparently in the future, the |  
11988            behavior is unspecified. |
- 11989 **EXIT STATUS**
- 11990            The following exit values shall be returned:
- 11991            0    Successful completion.
- 11992            >0    An error occurred.
- 11993 **CONSEQUENCES OF ERRORS**
- 11994            Default.
- 11995 **APPLICATION USAGE**
- 11996            Problems can arise if the system date and time have been modified (for example, put forward |  
11997            and then back again, or unsynchronized clocks across a network) and can also arise when |  
11998            different values of the *TZ* environment variable are used. |
- 11999            Problems of a similar nature can also arise for the operation of the *get* utility, which records the |  
12000            date and time in the file body. |
- 12001 **EXAMPLES**
- 12002            None.

12003 **RATIONALE**

12004 None.

12005 **FUTURE DIRECTIONS**

12006 None.

12007 **SEE ALSO**12008 *admin, diff, get, prs, rmdel*12009 **CHANGE HISTORY**

12010 First released in Issue 2.

12011 **Issue 5**

12012 The output format description in the STDOUT section is corrected.

12013 **Issue 6**

12014 The APPLICATION USAGE section is added.

12015 The normative text is reworded to avoid use of the term “must” for application requirements.

12016 The normative text is reworded to emphasize the term “shall” for implementation requirements. |

12017 The Open Group Base Resolution bwg2001-007 is applied as follows: |

12018 • The use of ‘-’ as a file argument is clarified. |

12019 • The use of STDIN is added. |

12020 • The ASYNCHRONOUS EVENTS section is updated to remove the implicit requirement that |  
12021 implementations re-signal themselves when catching a normally fatal signal. |12022 • New text is added to the INPUT FILES section warning that the maximum lines recorded in |  
12023 the file is 99 999. |12024 New text is added to the EXTENDED DESCRIPTION and APPLICAION USAGE sections |  
12025 regarding how the system date and time may be taken into account, and the TZ environment |  
12026 variable is added to the ENVIRONMENT VARIABLES section as per The Open Group Base |  
12027 Resolution bwg2001-007. |

12028 **NAME**

12029 df — report free disk space

12030 **SYNOPSIS**12031 UP XSI df [-k][-P|-t][*file...*]

12032

12033 **DESCRIPTION**

12034 XSI The *df* utility shall write the amount of available space and file slots for file systems on which the  
 12035 invoking user has appropriate read access. File systems shall be specified by the *file* operands;  
 12036 when none are specified, information shall be written for all file systems. The format of the  
 12037 default output from *df* is unspecified, but all space figures are reported in 512-byte units, unless  
 12038 the **-k** option is specified. This output shall contain at least the file system names, amount of  
 12039 XSI available space on each of these file systems, and the number of free file slots, or *inodes*,  
 12040 available; when **-t** is specified, the output shall contain the total allocated space as well.

12041 **OPTIONS**

12042 The *df* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2,  
 12043 Utility Syntax Guidelines.

12044 The following options shall be supported:

12045 **-k** Use 1024-byte units, instead of the default 512-byte units, when writing space  
 12046 figures.

12047 **-P** Produce output in the format described in the STDOUT section.

12048 XSI **-t** Include total allocated-space figures in the output.

12049 **OPERANDS**

12050 The following operand shall be supported:

12051 *file* A pathname of a file within the hierarchy of the desired file system. If a file other  
 12052 XSI than a FIFO, a regular file, a directory or a special file representing the device  
 12053 containing the file system (for example, */dev/dsk/0s1*) is specified, the results are  
 12054 unspecified. Otherwise, *df* shall write the amount of free space in the file system  
 12055 containing the specified *file* operand.

12056 **STDIN**

12057 Not used.

12058 **INPUT FILES**

12059 None.

12060 **ENVIRONMENT VARIABLES**12061 The following environment variables shall affect the execution of *df*:

12062 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 12063 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
 12064 Internationalization Variables for the precedence of internationalization variables  
 12065 used to determine the values of locale categories.)

12066 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 12067 internationalization variables.

12068 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 12069 characters (for example, single-byte as opposed to multi-byte characters in  
 12070 arguments).

12071 **LC\_MESSAGES**  
 12072 Determine the locale that should be used to affect the format and contents of  
 12073 diagnostic messages written to standard error and informative messages written to  
 12074 standard output.

12075 XSI **NLSPATH** Determine the location of message catalogs for the processing of **LC\_MESSAGES**.

12076 **ASYNCHRONOUS EVENTS**  
 12077 Default.

12078 **STDOUT**  
 12079 When both the **-k** and **-P** options are specified, the following header line shall be written (in the  
 12080 POSIX locale):  
 12081 "Filesystem 1024-blocks Used Available Capacity Mounted on\n"

12082 When the **-P** option is specified without the **-k** option, the following header line shall be written  
 12083 (in the POSIX locale):  
 12084 "Filesystem 512-blocks Used Available Capacity Mounted on\n"

12085 The implementation may adjust the spacing of the header line and the individual data lines so  
 12086 that the information is presented in orderly columns.

12087 The remaining output with **-P** shall consist of one line of information for each specified file  
 12088 system. These lines shall be formatted as follows:  
 12089 "%s %d %d %d %d%% %s\n", <file system name>, <total space>,  
 12090 <space used>, <space free>, <percentage used>,  
 12091 <file system root>

12092 In the following list, all quantities expressed in 512-byte units (1 024-byte when **-k** is specified)  
 12093 shall be rounded up to the next higher unit. The fields are:  
 12094 <file system name>  
 12095 The name of the file system, in an implementation-defined format.

12096 <total space> The total size of the file system in 512-byte units. The exact meaning of this figure  
 12097 is implementation-defined, but should include <space used>, <space free>, plus any  
 12098 space reserved by the system not normally available to a user.

12099 <space used> The total amount of space allocated to existing files in the file system, in 512-byte  
 12100 units.

12101 <space free> The total amount of space available within the file system for the creation of new  
 12102 files by unprivileged users, in 512-byte units. When this figure is less than or equal  
 12103 to zero, it shall not be possible to create any new files on the file system without  
 12104 first deleting others, unless the process has appropriate privileges. The figure  
 12105 written may be less than zero.

12106 <percentage used>  
 12107 The percentage of the normally available space that is currently allocated to all  
 12108 files on the file system. This shall be calculated using the fraction:  
 12109 
$$\frac{\text{<space used>}}{\text{<space used> + <space free>}}$$
  
 12110 expressed as a percentage. This percentage may be greater than 100 if <space free>  
 12111 is less than zero. The percentage value shall be expressed as a positive integer,  
 12112 with any fractional result causing it to be rounded to the next highest integer.

- 12113 <file system root>  
12114 The directory below which the file system hierarchy appears.
- 12115 XSI The output format is unspecified when `-t` is used.
- 12116 **STDERR**  
12117 The standard error shall be used only for diagnostic messages.
- 12118 **OUTPUT FILES**  
12119 None.
- 12120 **EXTENDED DESCRIPTION**  
12121 None.
- 12122 **EXIT STATUS**  
12123 The following exit values shall be returned:  
12124 0 Successful completion.  
12125 >0 An error occurred.
- 12126 **CONSEQUENCES OF ERRORS**  
12127 Default.
- 12128 **APPLICATION USAGE**  
12129 On most systems, the “name of the file system, in an implementation-defined format” is the  
12130 special file on which the file system is mounted.  
12131 On large file systems, the calculation specified for percentage used can create huge rounding  
12132 errors.
- 12133 **EXAMPLES**  
12134 1. The following example writes portable information about the `/usr` file system:  
12135 `df -P /usr`  
12136 2. Assuming that `/usr/src` is part of the `/usr` file system, the following produces the same  
12137 output as the previous example:  
12138 `df -P /usr/src`
- 12139 **RATIONALE**  
12140 The behavior of `df` with the `-P` option is the default action of the 4.2 BSD `df` utility. The uppercase  
12141 `-P` was selected to avoid collision with a known industry extension using `-p`.  
12142 Historical `df` implementations vary considerably in their default output. It was therefore  
12143 necessary to describe the default output in a loose manner to accommodate all known historical  
12144 implementations and to add a portable option (`-P`) to provide information in a portable format.  
12145 The use of 512-byte units is historical practice and maintains compatibility with `ls` and other  
12146 utilities in this volume of IEEE Std 1003.1-200x. This does not mandate that the file system itself  
12147 be based on 512-byte blocks. The `-k` option was added as a compromise measure. It was agreed  
12148 by the standard developers that 512 bytes was the best default unit because of its complete  
12149 historical consistency on System V (*versus* the mixed 512/1024-byte usage on BSD systems), and  
12150 that a `-k` option to switch to 1024-byte units was a good compromise. Users who prefer the  
12151 more logical 1024-byte quantity can easily alias `df` to `df -k` without breaking many historical  
12152 scripts relying on the 512-byte units.  
12153 It was suggested that `df` and the various related utilities be modified to access a `BLOCKSIZE`  
12154 environment variable to achieve consistency and user acceptance. Since this is not historical  
12155 practice on any system, it is left as a possible area for system extensions and will be re-evaluated

12156 in a future version if it is widely implemented.

12157 **FUTURE DIRECTIONS**

12158 None.

12159 **SEE ALSO**

12160 *find*

12161 **CHANGE HISTORY**

12162 First released in Issue 2.

12163 **Issue 6**

12164 This utility is now marked as part of the User Portability Utilities option.

12165 **NAME**

12166 diff — compare two files

12167 **SYNOPSIS**12168 diff [-c | -e | -f | -C *n*][-br] *file1 file2*12169 **DESCRIPTION**

12170 The *diff* utility shall compare the contents of *file1* and *file2* and write to standard output a list of  
 12171 changes necessary to convert *file1* into *file2*. This list should be minimal. No output shall be  
 12172 produced if the files are identical.

12173 **OPTIONS**

12174 The *diff* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section  
 12175 12.2, Utility Syntax Guidelines.

12176 The following options shall be supported:

- 12177 **-b** Cause any amount of white space at the end of a line to be treated as a single  
 12178 <newline> (that is, the white-space characters preceding the <newline> are  
 12179 ignored) and other strings of white-space characters, not including <newline>s, to  
 12180 compare equal.
- 12181 **-c** Produce output in a form that provides three lines of context.
- 12182 **-C *n*** Produce output in a form that provides *n* lines of context (where *n* shall be  
 12183 interpreted as a positive decimal integer).
- 12184 **-e** Produce output in a form suitable as input for the *ed* utility, which can then be  
 12185 used to convert *file1* into *file2*.
- 12186 **-f** Produce output in an alternative form, similar in format to **-e**, but not intended to  
 12187 be suitable as input for the *ed* utility, and in the opposite order.
- 12188 **-r** Apply *diff* recursively to files and directories of the same name when *file1* and *file2*  
 12189 are both directories.

12190 **OPERANDS**

12191 The following operands shall be supported:

12192 ***file1, file2*** A pathname of a file to be compared. If either the *file1* or *file2* operand is '-', the  
 12193 standard input shall be used in its place.

12194 If both *file1* and *file2* are directories, *diff* shall not compare block special files, character special  
 12195 files, or FIFO special files to any files and shall not compare regular files to directories. Further  
 12196 details are as specified in **Diff Directory Comparison Format** (on page 2521). The behavior of  
 12197 *diff* on other file types is implementation-defined when found in directories.

12198 If only one of *file1* and *file2* is a directory, *diff* shall be applied to the non-directory file and the file  
 12199 contained in the directory file with a filename that is the same as the last component of the non-  
 12200 directory file.

12201 **STDIN**

12202 The standard input shall be used only if one of the *file1* or *file2* operands references standard  
 12203 input. See the INPUT FILES section.

12204 **INPUT FILES**

12205 The input files may be of any type.

## 12206 ENVIRONMENT VARIABLES

12207 The following environment variables shall affect the execution of *diff*:

12208 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 12209 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
 12210 Internationalization Variables for the precedence of internationalization variables  
 12211 used to determine the values of locale categories.)

12212 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 12213 internationalization variables.

12214 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 12215 characters (for example, single-byte as opposed to multi-byte characters in  
 12216 arguments and input files).

12217 *LC\_MESSAGES*

12218 Determine the locale that should be used to affect the format and contents of  
 12219 diagnostic messages written to standard error and informative messages written to  
 12220 standard output.

12221 *LC\_TIME* Determine the locale for affecting the format of file timestamps written with the  
 12222 *-C* and *-c* options.

12223 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

12224 *TZ* Determine the timezone used for calculating file timestamps written with the *-C*  
 12225 and *-c* options. If *TZ* is unset or null, an unspecified default timezone shall be  
 12226 used.

## 12227 ASYNCHRONOUS EVENTS

12228 Default.

## 12229 STDOUT

12230 **Diff Directory Comparison Format**

12231 If both *file1* and *file2* are directories, the following output formats shall be used.

12232 In the POSIX locale, each file that is present in only one directory shall be reported using the  
 12233 following format:

12234 "Only in %s: %s\n", <directory pathname>, <filename>

12235 In the POSIX locale, subdirectories that are common to the two directories may be reported with  
 12236 the following format:

12237 "Common subdirectories: %s and %s\n", <directory1 pathname>,  
 12238 <directory2 pathname>

12239 For each file common to the two directories if the two files are not to be compared, the following  
 12240 format shall be used in the POSIX locale:

12241 "File %s is a %s while file %s is a %s\n", <directory1 pathname>,  
 12242 <file type of directory1 pathname>, <directory2 pathname>,  
 12243 <file type of directory2 pathname>

12244 For each file common to the two directories, if the files are compared and are identical, no output  
 12245 shall be written. If the two files differ, the following format is written:

12246 "diff %s %s %s\n", <diff\_options>, <filename1>, <filename2>

12247 where *<diff\_options>* are the options as specified on the command line.

12248 All directory pathnames listed in this section shall be relative to the original command line  
 12249 arguments. All other names of files listed in this section shall be filenames (pathname  
 12250 components).

12251 **Diff Binary Output Format**

12252 In the POSIX locale, if one or both of the files being compared are not text files, an unspecified  
 12253 format shall be used that contains the pathnames of two files being compared and the string  
 12254 "differ".

12255 If both files being compared are text files, depending on the options specified, one of the  
 12256 following formats shall be used to write the differences.

12257 **Diff Default Output Format**

12258 The default (without *-e*, *-f*, *-c*, or *-C* options) *diff* utility output shall contain lines of these  
 12259 forms:

12260 "%da%d\n", *<num1>*, *<num2>*

12261 "%da%d,%d\n", *<num1>*, *<num2>*, *<num3>*

12262 "%dd%d\n", *<num1>*, *<num2>*

12263 "%d,%dd%d\n", *<num1>*, *<num2>*, *<num3>*

12264 "%dc%d\n", *<num1>*, *<num2>*

12265 "%d,%dc%d\n", *<num1>*, *<num2>*, *<num3>*

12266 "%dc%d,%d\n", *<num1>*, *<num2>*, *<num3>*

12267 "%d,%dc%d,%d\n", *<num1>*, *<num2>*, *<num3>*, *<num4>*

12268 These lines resemble *ed* subcommands to convert *file1* into *file2*. The line numbers before the  
 12269 action letters shall pertain to *file1*; those after shall pertain to *file2*. Thus, by exchanging *a* for *d*  
 12270 and reading the line in reverse order, one can also determine how to convert *file2* into *file1*. As in  
 12271 *ed*, identical pairs (where *num1*= *num2*) are abbreviated as a single number.

12272 Following each of these lines, *diff* shall write to standard output all lines affected in the first file  
 12273 using the format:

12274 "<Δ%s", *<line>*

12275 and all lines affected in the second file using the format:

12276 ">Δ%s", *<line>*

12277 If there are lines affected in both *file1* and *file2* (as with the *c* subcommand), the changes are  
 12278 separated with a line consisting of three hyphens:

12279 "---\n"

12280 **Diff -e Output Format**

12281 With the **-e** option, a script shall be produced that shall, when provided as input to *ed*, along  
 12282 with an appended **w** (write) command, convert *file1* into *file2*. Only the **a** (append), **c** (change), **d**  
 12283 (delete), **i** (insert), and **s** (substitute) commands of *ed* shall be used in this script. Text lines,  
 12284 except those consisting of the single character period ( `' . '` ), shall be output as they appear in the  
 12285 file.

12286 **Diff -f Output Format**

12287 With the **-f** option, an alternative format of script shall be produced. It is similar to that  
 12288 produced by **-e**, with the following differences:

- 12289 1. It is expressed in reverse sequence; the output of **-e** orders changes from the end of the file  
 12290 to the beginning; the **-f** from beginning to end.
- 12291 2. The command form `<lines> <command-letter>` used by **-e** is reversed. For example,  
 12292 `10c` with **-e** would be `c10` with **-f**.
- 12293 3. The form used for ranges of line numbers is `<space>`-separated, rather than comma-  
 12294 separated.

12295 **Diff -c or -C Output Format**

12296 With the **-c** or **-C** option, the output format shall consist of affected lines along with  
 12297 surrounding lines of context. The affected lines shall show which ones need to be deleted or  
 12298 changed in *file1*, and those added from *file2*. With the **-c** option, three lines of context, if  
 12299 available, shall be written before and after the affected lines. With the **-C** option, the user can  
 12300 specify how many lines of context are written. The exact format follows.

12301 The name and last modification time of each file shall be output in the following format:

```
12302 "**** %s %s\n", file1, <file1 timestamp>  
12303 "---- %s %s\n", file2, <file2 timestamp>
```

12304 Each `<file>` field shall be the pathname of the corresponding file being compared. The pathname  
 12305 written for standard input is unspecified.

12306 In the POSIX locale, each `<timestamp>` field shall be equivalent to the output from the following  
 12307 command:

```
12308 date "+%a %b %e %T %Y"
```

12309 without the trailing `<newline>`, executed at the time of last modification of the corresponding  
 12310 file (or the current time, if the file is standard input).

12311 Then, the following output formats shall be applied for every set of changes.

12312 First, a line shall be written in the following format:

```
12313 "*****\n"
```

12314 Next, the range of lines in *file1* shall be written in the following format:

```
12315 "**** %d,%d ****\n", <beginning line number>, <ending line number>
```

12316 Next, the affected lines along with lines of context (unaffected lines) shall be written. Unaffected  
 12317 lines shall be written in the following format:

```
12318 "ΔΔ%s", <unaffected_line>
```

12319 Deleted lines shall be written as:  
12320 "-Δ%s", <deleted\_line>  
12321 Changed lines shall be written as:  
12322 "!Δ%s", <changed\_line>  
12323 Next, the range of lines in *file2* shall be written in the following format:  
12324 "--- %d,%d ----\n", <beginning line number>, <ending line number>  
12325 Then, lines of context and changed lines shall be written as described in the previous formats.  
12326 Lines added from *file2* shall be written in the following format:  
12327 "+Δ%s", <added\_line>

12328 **STDERR**  
12329 The standard error shall be used only for diagnostic messages.

12330 **OUTPUT FILES**  
12331 None.

12332 **EXTENDED DESCRIPTION**  
12333 None.

12334 **EXIT STATUS**  
12335 The following exit values shall be returned:  
12336 0 No differences were found.  
12337 1 Differences were found.  
12338 >1 An error occurred.

12339 **CONSEQUENCES OF ERRORS**  
12340 Default.

12341 **APPLICATION USAGE**  
12342 If lines at the end of a file are changed and other lines are added, *diff* output may show this as a  
12343 delete and add, as a change, or as a change and add; *diff* is not expected to know which  
12344 happened and users should not care about the difference in output as long as it clearly shows the  
12345 differences between the files.

12346 **EXAMPLES**  
12347 If *dir1* is a directory containing a directory named *x*, *dir2* is a directory containing a directory  
12348 named *x*, *dir1/x* and *dir2/x* both contain files named *date.out*, and *dir2/x* contains a file named *y*,  
12349 the command:  
12350 `diff -r dir1 dir2`  
12351 could produce output similar to:  
12352 Common subdirectories: dir1/x and dir2/x  
12353 Only in dir2/x: y  
12354 `diff -r dir1/x/date.out dir2/x/date.out`  
12355 `lc1`  
12356 `< Mon Jul 2 13:12:16 PDT 1990`  
12357 `---`  
12358 `> Tue Jun 19 21:41:39 PDT 1990`

## 12359 RATIONALE

12360 The `-h` option was omitted because it was insufficiently specified and does not add to  
12361 applications portability.

12362 Historical implementations employ algorithms that do not always produce a minimum list of  
12363 differences; the current language about making every effort is the best this volume of  
12364 IEEE Std 1003.1-200x can do, as there is no metric that could be employed to judge the quality of  
12365 implementations against any and all file contents. The statement “This list should be minimal”  
12366 clearly implies that implementations are not expected to provide the following output when  
12367 comparing two 100-line files that differ in only one character on a single line:

```
12368 1,100c1,100
12369 all 100 lines from file1 preceded with "< "
12370 ----
12371 all 100 lines from file2 preceded with "> "
```

12372 The “Only in” messages required when the `-r` option is specified are not used by most historical  
12373 implementations if the `-e` option is also specified. It is required here because it provides useful  
12374 information that must be provided to update a target directory hierarchy to match a source  
12375 hierarchy. The “Common subdirectories” messages are written by System V and 4.3 BSD when  
12376 the `-r` option is specified. They are allowed here but are not required because they are reporting  
12377 on something that is the same, not reporting a difference, and are not needed to update a target  
12378 hierarchy.

12379 The `-c` option, which writes output in a format using lines of context, has been included. The  
12380 format is useful for a variety of reasons, among them being much improved readability and the  
12381 ability to understand difference changes when the target file has line numbers that differ from  
12382 another similar, but slightly different, copy. The *patch* utility is most valuable when working  
12383 with difference listings using the context format. The BSD version of `-c` takes an optional  
12384 argument specifying the amount of context. Rather than overloading `-c` and breaking the Utility  
12385 Syntax Guidelines for *diff*, the standard developers decided to add a separate option for  
12386 specifying a context *diff* with a specified amount of context (`-C`). Also, the format for context  
12387 *diffs* was extended slightly in 4.3 BSD to allow multiple changes that are within context lines  
12388 from each other to be merged together. The output format contains an additional four asterisks  
12389 after the range of affected lines in the first filename. This was to provide a flag for old programs  
12390 (like old versions of *patch*) that only understand the old context format. The version of context  
12391 described here does not require that multiple changes within context lines be merged, but it does  
12392 not prohibit it either. The extension is upward-compatible, so any vendors that wish to retain the  
12393 old version of *diff* can do so by adding the extra four asterisks (that is, utilities that currently use  
12394 *diff* and understand the new merged format will also understand the old unmerged format, but  
12395 not *vice versa*).

12396 The substitute command was added as an additional format for the `-e` option. This was added to  
12397 provide implementations a way to fix the classic “dot alone on a line” bug present in many  
12398 versions of *diff*. Since many implementations have fixed this bug, the standard developers  
12399 decided not to standardize broken behavior, but rather to provide the necessary tool for fixing  
12400 the bug. One way to fix this bug is to output two periods whenever a lone period is needed, then  
12401 terminate the append command with a period, and then use the substitute command to convert  
12402 the two periods into one period.

12403 The BSD-derived `-r` option was added to provide a mechanism for using *diff* to compare two file  
12404 system trees. This behavior is useful, is standard practice on all BSD-derived systems, and is not  
12405 easily reproducible with the *find* utility.

12406 The requirement that *diff* not compare files in some circumstances, even though they have the  
12407 same name, is based on the actual output of historical implementations. The message specified

- 12408 here is already in use when a directory is being compared to a non-directory. It is extended here  
12409 to preclude the problems arising from running into FIFOs and other files that would cause *diff* to  
12410 hang waiting for input with no indication to the user that *diff* was hung. In most common usage,  
12411 *diff -r* should indicate differences in the file hierarchies, not the difference of contents of devices  
12412 pointed to by the hierarchies.
- 12413 Many early implementations of *diff* require seekable files. Since the System Interfaces volume of  
12414 IEEE Std 1003.1-200x supports named pipes, the standard developers decided that such a  
12415 restriction was unreasonable. Note also that the allowed filename – almost always refers to a  
12416 pipe.
- 12417 No directory search order is specified for *diff*. The historical ordering is, in fact, not optimal, in  
12418 that it prints out all of the differences at the current level, including the statements about all  
12419 common subdirectories before recursing into those subdirectories.
- 12420 The message:
- 12421 "diff %s %s %s\n", <diff\_options>, <filename1>, <filename2>
- 12422 does not vary by locale because it is the representation of a command, not an English sentence.
- 12423 **FUTURE DIRECTIONS**
- 12424 None.
- 12425 **SEE ALSO**
- 12426 *cmp, comm, ed*
- 12427 **CHANGE HISTORY**
- 12428 First released in Issue 2.
- 12429 **Issue 5**
- 12430 FUTURE DIRECTIONS section added.
- 12431 **Issue 6**
- 12432 The following new requirements on POSIX implementations derive from alignment with the  
12433 Single UNIX Specification:
- 12434
  - The *-f* option is added.
- 12435 The output format for *-c* or *-C* format is changed to align with changes to the IEEE P1003.2b  
12436 draft standard resulting from IEEE PASC Interpretation 1003.2 #71.
- 12437 The normative text is reworded to avoid use of the term “must” for application requirements.

12438 **NAME**

12439           dirname — return the directory portion of pathname

12440 **SYNOPSIS**12441           dirname *string*12442 **DESCRIPTION**

12443           The *string* operand shall be treated as a pathname, as defined in the Base Definitions volume of  
 12444           IEEE Std 1003.1-200x, Section 3.266, Pathname. The string *string* shall be converted to the name  
 12445           of the directory containing the filename corresponding to the last pathname component in  
 12446           *string*, performing actions equivalent to the following steps in order:

- 12447           1. If *string* is //, skip steps 2 to 5.
- 12448           2. If *string* consists entirely of slash characters, *string* shall be set to a single slash character. In  
 12449           this case, skip steps 3 to 8.
- 12450           3. If there are any trailing slash characters in *string*, they shall be removed.
- 12451           4. If there are no slash characters remaining in *string*, *string* shall be set to a single period  
 12452           character. In this case, skip steps 5 to 8.
- 12453           5. If there are any trailing non-slash characters in *string*, they shall be removed.
- 12454           6. If the remaining *string* is //, it is implementation-defined whether steps 7 and 8 are skipped  
 12455           or processed.
- 12456           7. If there are any trailing slash characters in *string*, they shall be removed.
- 12457           8. If the remaining *string* is empty, *string* shall be set to a single slash character.

12458           The resulting string shall be written to standard output.

12459 **OPTIONS**

12460           None.

12461 **OPERANDS**

12462           The following operand shall be supported:

12463           *string*        A string.12464 **STDIN**

12465           Not used.

12466 **INPUT FILES**

12467           None.

12468 **ENVIRONMENT VARIABLES**12469           The following environment variables shall affect the execution of *dirname*:

- |                                  |                 |                                                                                                                                                                                                                                                                                                       |
|----------------------------------|-----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 12470<br>12471<br>12472<br>12473 | <i>LANG</i>     | Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.) |
| 12474<br>12475                   | <i>LC_ALL</i>   | If set to a non-empty string value, override the values of all the other internationalization variables.                                                                                                                                                                                              |
| 12476<br>12477<br>12478          | <i>LC_CTYPE</i> | Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).                                                                                                                             |

12479 **LC\_MESSAGES**  
 12480 Determine the locale that should be used to affect the format and contents of  
 12481 diagnostic messages written to standard error.

12482 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

12483 **ASYNCHRONOUS EVENTS**  
 12484 Default.

12485 **STDOUT**  
 12486 The *dirname* utility shall write a line to the standard output in the following format:

12487 "%s\n", <resulting string>

12488 **STDERR**  
 12489 The standard error shall be used only for diagnostic messages.

12490 **OUTPUT FILES**  
 12491 None.

12492 **EXTENDED DESCRIPTION**  
 12493 None.

12494 **EXIT STATUS**  
 12495 The following exit values shall be returned:

12496 0 Successful completion.

12497 >0 An error occurred.

12498 **CONSEQUENCES OF ERRORS**  
 12499 Default.

12500 **APPLICATION USAGE**  
 12501 The definition of *pathname* specifies implementation-defined behavior for pathnames starting  
 12502 with two slash characters. Therefore, applications shall not arbitrarily add slashes to the  
 12503 beginning of a pathname unless they can ensure that there are more or less than two or are  
 12504 prepared to deal with the implementation-defined consequences.

12505 **EXAMPLES**

|       | Command                 | Results     |
|-------|-------------------------|-------------|
| 12506 | <i>dirname</i> /        | /           |
| 12507 | <i>dirname</i> //       | / or //     |
| 12508 | <i>dirname</i> /a/b/    | /a          |
| 12509 | <i>dirname</i> //a//b// | //a         |
| 12510 | <i>dirname</i>          | Unspecified |
| 12511 | <i>dirname</i> a        | .( \$? = 0) |
| 12512 | <i>dirname</i> ""       | .( \$? = 0) |
| 12513 | <i>dirname</i> /a       | /           |
| 12514 | <i>dirname</i> /a/b     | /a          |
| 12515 | <i>dirname</i> a/b      | a           |
| 12516 |                         |             |

12517 **RATIONALE**  
 12518 The *dirname* utility originated in System III. It has evolved through the System V releases to a  
 12519 version that matches the requirements specified in this description in System V Release 3. 4.3  
 12520 BSD and earlier versions did not include *dirname*.

12521 The behaviors of *basename* and *dirname* in this volume of IEEE Std 1003.1-200x have been  
 12522 coordinated so that when *string* is a valid pathname:

12523           \$(basename "*string*")

12524           would be a valid filename for the file in the directory:

12525           \$(dirname "*string*")

12526           This would not work for the versions of these utilities in early proposals due to the way  
12527           processing of trailing slashes was specified. Consideration was given to leaving processing  
12528           unspecified if there were trailing slashes, but this cannot be done; the Base Definitions volume of  
12529           IEEE Std 1003.1-200x, Section 3.266, Pathname allows trailing slashes. The *basename* and *dirname*  
12530           utilities have to specify consistent handling for all valid pathnames.

12531 **FUTURE DIRECTIONS**

12532           None.

12533 **SEE ALSO**

12534           *basename*, Section 2.5 (on page 2235)

12535 **CHANGE HISTORY**

12536           First released in Issue 2.

12537 **NAME**

12538 du — estimate file space usage

12539 **SYNOPSIS**12540 UP du [-a | -s][-kx][-H | -L][*file* ...]

12541

12542 **DESCRIPTION**

12543 By default, the *du* utility shall write to standard output the size of the file space allocated to, and  
 12544 the size of the file space allocated to each subdirectory of, the file hierarchy rooted in each of the  
 12545 specified files. By default, when a symbolic link is encountered on the command line or in the  
 12546 file hierarchy, *du* shall count the size of the symbolic link (rather than the file referenced by the  
 12547 link), and shall not follow the link to another portion of the file hierarchy. The size of the file  
 12548 space allocated to a file of type directory shall be defined as the sum total of space allocated to  
 12549 all files in the file hierarchy rooted in the directory plus the space allocated to the directory itself.

12550 When *du* cannot *stat()* files or *stat()* or read directories, it shall report an error condition and the  
 12551 final exit status is affected. Files with multiple links shall be counted and written for only one  
 12552 entry. The directory entry that is selected in the report is unspecified. By default, file sizes shall  
 12553 be written in 512-byte units, rounded up to the next 512-byte unit.

12554 **OPTIONS**

12555 The *du* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2,  
 12556 Utility Syntax Guidelines.

12557 The following options shall be supported:

12558 **-a** In addition to the default output, report the size of each file not of type directory in  
 12559 the file hierarchy rooted in the specified file. Regardless of the presence of the **-a**  
 12560 option, non-directories given as *file* operands shall always be listed.

12561 **-H** If a symbolic link is specified on the command line, *du* shall count the size of the  
 12562 file or file hierarchy referenced by the link.

12563 **-k** Write the files sizes in units of 1 024 bytes, rather than the default 512-byte units.

12564 **-L** If a symbolic link is specified on the command line or encountered during the  
 12565 traversal of a file hierarchy, *du* shall count the size of the file or file hierarchy  
 12566 referenced by the link.

12567 **-s** Instead of the default output, report only the total sum for each of the specified  
 12568 files.

12569 **-x** When evaluating file sizes, evaluate only those files that have the same device as  
 12570 the file specified by the *file* operand.

12571 Specifying more than one of the mutually-exclusive options **-H** and **-L** shall not be considered  
 12572 an error. The last option specified shall determine the behavior of the utility.

12573 **OPERANDS**

12574 The following operand shall be supported:

12575 *file* The pathname of a file whose size is to be written. If no *file* is specified, the current  
 12576 directory shall be used.

12577 **STDIN**

12578 Not used.

12579 **INPUT FILES**

12580       None.

12581 **ENVIRONMENT VARIABLES**12582       The following environment variables shall affect the execution of *du*:

12583       *LANG*       Provide a default value for the internationalization variables that are unset or null.  
 12584                   (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
 12585                   Internationalization Variables for the precedence of internationalization variables  
 12586                   used to determine the values of locale categories.)

12587       *LC\_ALL*      If set to a non-empty string value, override the values of all the other  
 12588                   internationalization variables.

12589       *LC\_CTYPE*   Determine the locale for the interpretation of sequences of bytes of text data as  
 12590                   characters (for example, single-byte as opposed to multi-byte characters in  
 12591                   arguments).

12592       *LC\_MESSAGES*

12593                   Determine the locale that should be used to affect the format and contents of  
 12594                   diagnostic messages written to standard error.

12595 *XSI*       *NLSPATH*   Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

12596 **ASYNCHRONOUS EVENTS**

12597       Default.

12598 **STDOUT**

12599       The output from *du* shall consist of the amount of the space allocated to a file and the name of  
 12600       the file, in the following format:

12601       "%d %s\n", *<size>*, *<pathname>*

12602 **STDERR**

12603       The standard error shall be used only for diagnostic messages.

12604 **OUTPUT FILES**

12605       None.

12606 **EXTENDED DESCRIPTION**

12607       None.

12608 **EXIT STATUS**

12609       The following exit values shall be returned:

12610       0   Successful completion.

12611       &gt;0  An error occurred.

12612 **CONSEQUENCES OF ERRORS**

12613       Default.

12614 **APPLICATION USAGE**

12615 None.

12616 **EXAMPLES**

12617 None.

12618 **RATIONALE**

12619 The use of 512-byte units is historical practice and maintains compatibility with *ls* and other  
12620 utilities in this volume of IEEE Std 1003.1-200x. This does not mandate that the file system itself  
12621 be based on 512-byte blocks. The **-k** option was added as a compromise measure. It was agreed  
12622 by the standard developers that 512 bytes was the best default unit because of its complete  
12623 historical consistency on System V (*versus* the mixed 512/1 024-byte usage on BSD systems), and  
12624 that a **-k** option to switch to 1 024-byte units was a good compromise. Users who prefer the  
12625 1 024-byte quantity can easily alias *du* to *du -k* without breaking the many historical scripts  
12626 relying on the 512-byte units.

12627 The **-b** option was added to an early proposal to provide a resolution to the situation where  
12628 System V and BSD systems give figures for file sizes in *blocks*, which is an implementation-  
12629 defined concept. (In common usage, the block size is 512 bytes for System V and 1 024 bytes for  
12630 BSD systems.) However, **-b** was later deleted, since the default was eventually decided as 512-  
12631 byte units.

12632 Historical file systems provided no way to obtain exact figures for the space allocation given to  
12633 files. There are two known areas of inaccuracies in historical file systems: cases of *indirect blocks*  
12634 being used by the file system or *sparse* files yielding incorrectly high values. An indirect block is  
12635 space used by the file system in the storage of the file, but that need not be counted in the space  
12636 allocated to the file. A *sparse* file is one in which an *lseek()* call has been made to a position  
12637 beyond the end of the file and data has subsequently been written at that point. A file system  
12638 need not allocate all the intervening zero-filled blocks to such a file. It is up to the  
12639 implementation to define exactly how accurate its methods are.

12640 The **-a** and **-s** options were mutually-exclusive in the original version of *du*. The POSIX Shell  
12641 and Utilities description is implied by the language in the SVID where **-s** is described as causing  
12642 “only the grand total” to be reported. Some systems may produce output for **-sa**, but a Strictly  
12643 Conforming POSIX Shell and Utilities Application cannot use that combination.

12644 The **-a** and **-s** options were adopted from the SVID except that the System V behavior of not  
12645 listing non-directories explicitly given as operands, unless the **-a** option is specified, was  
12646 considered a bug; the BSD-based behavior (report for all operands) is mandated. The default  
12647 behavior of *du* in the SVID with regard to reporting the failure to read files (it produces no  
12648 messages) was considered counter-intuitive, and thus it was specified that the POSIX Shell and  
12649 Utilities default behavior shall be to produce such messages. These messages can be turned off  
12650 with shell redirection to achieve the System V behavior.

12651 The **-x** option is historical practice on recent BSD systems. It has been adopted by this volume of  
12652 IEEE Std 1003.1-200x because there was no other historical method of limiting the *du* search to a  
12653 single file hierarchy. This limitation of the search is necessary to make it possible to obtain file  
12654 space usage information about a file system on which other file systems are mounted, without  
12655 having to resort to a lengthy *find* and *awk* script.

12656 **FUTURE DIRECTIONS**

12657 None.

12658 **SEE ALSO**

12659           *ls*

12660 **CHANGE HISTORY**

12661           First released in Issue 2.

12662 **Issue 6**

12663           This utility is now marked as part of the User Portability Utilities option.

12664           The APPLICATION USAGE section is added.

12665           This utility is reinstated, as the LEGACY marking was incorrect in Issue 5.

12666           The obsolescent `-r` option has been removed.

12667           The Open Group Corrigendum U025/3 is applied. The *du* utility had incorrectly been marked  
12668           LEGACY.

12669           The `-H` and `-L` options for symbolic links are added as described in the IEEE P1003.2b draft  
12670           standard.

12671 **NAME**

12672 echo — write arguments to standard output

12673 **SYNOPSIS**12674 echo [*string* ...]12675 **DESCRIPTION**12676 The *echo* utility writes its arguments to standard output, followed by a <newline>. If there are  
12677 no arguments, only the <newline> is written.12678 **OPTIONS**12679 The *echo* utility shall not recognize the "--" argument in the manner specified by Guideline 10  
12680 of the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2, Utility Syntax Guidelines;  
12681 "--" shall be recognized as a string operand.

12682 Implementations shall not support any options.

12683 **OPERANDS**

12684 The following operands shall be supported:

12685 *string* A string to be written to standard output. If any operand is *-n*, it shall be treated as  
12686 a string, not an option. The following character sequences shall be recognized  
12687 within any of the arguments:

12688 \a Write an &lt;alert&gt;.

12689 \b Write a &lt;backspace&gt;.

12690 \c Suppress the <newline> that otherwise follows the final argument in the  
12691 output. All characters following the '\c' in the arguments shall be  
12692 ignored.

12693 \f Write a &lt;form-feed&gt;.

12694 \n Write a &lt;newline&gt;.

12695 \r Write a &lt;carriage-return&gt;.

12696 \t Write a &lt;tab&gt;.

12697 \v Write a &lt;vertical-tab&gt;.

12698 \\ Write a backslash character.

12699 \0*num* Write an 8-bit value that is the zero, one, two, or three-digit octal number  
12700 *num*.12701 **STDIN**

12702 Not used.

12703 **INPUT FILES**

12704 None.

12705 **ENVIRONMENT VARIABLES**12706 The following environment variables shall affect the execution of *echo*:12707 *LANG* Provide a default value for the internationalization variables that are unset or null.  
12708 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
12709 Internationalization Variables for the precedence of internationalization variables  
12710 used to determine the values of locale categories.)12711 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
12712 internationalization variables.

12713            *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 12714 characters (for example, single-byte as opposed to multi-byte characters in  
 12715 arguments).

12716            *LC\_MESSAGES*  
 12717                    Determine the locale that should be used to affect the format and contents of  
 12718 diagnostic messages written to standard error.

12719 XSI        *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

12720 **ASYNCHRONOUS EVENTS**

12721            Default.

12722 **STDOUT**

12723            The *echo* utility arguments shall be separated by single <space>s and a <newline> shall follow |  
 12724 the last argument. Output transformations shall occur based on the escape sequences in the |  
 12725 input. See the OPERANDS section. |

12726 **STDERR**

12727            The standard error shall be used only for diagnostic messages. |

12728 **OUTPUT FILES**

12729            None.

12730 **EXTENDED DESCRIPTION**

12731            None.

12732 **EXIT STATUS**

12733            The following exit values shall be returned:

12734            0 Successful completion.

12735            >0 An error occurred.

12736 **CONSEQUENCES OF ERRORS**

12737            Default.

12738 **APPLICATION USAGE**

12739            In the ISO/IEC 9945-2:1993 standard, it was not possible to use *echo* portably across all systems  
 12740 that were not XSI-conformant unless both *-n* (as the first argument) and escape sequences were  
 12741 omitted.

12742            The *printf* utility can be used portably to emulate any of the traditional behaviors of the *echo*  
 12743 utility as follows:

12744            • The historic System V *echo* and the current requirements in this volume of  
 12745 IEEE Std 1003.1-200x are equivalent to:

12746            `printf "%b\n" "$*"`

12747            • The BSD *echo* is equivalent to:

12748            `if [ "X$1" = "X-n" ]`  
 12749            `then`  
 12750                 `shift`  
 12751                 `printf "%s" "$*"`  
 12752            `else`  
 12753                 `printf "%s\n" "$*"`  
 12754            `fi`

12755 New applications are encouraged to use *printf* instead of *echo*.

12756 **EXAMPLES**

12757 None.

12758 **RATIONALE**

12759 The *echo* utility has not been made obsolescent because of its extremely widespread use in |  
12760 historical applications. Conforming applications that wish to do prompting without <newline>s |  
12761 or that could possibly be expecting to echo a *-n*, should use the *printf* utility derived from the |  
12762 Ninth Edition system.

12763 As specified, *echo* writes its arguments in the simplest of ways. The two different historical  
12764 versions of *echo* vary in fatally incompatible ways.

12765 The BSD *echo* checks the first argument for the string *-n* which causes it to suppress the  
12766 <newline> that would otherwise follow the final argument in the output.

12767 The System V *echo* does not support any options, but allows escape sequences within its  
12768 operands, as described in the OPERANDS section.

12769 The *echo* utility does not support Utility Syntax Guideline 10 because historical applications  
12770 depend on *echo* to echo *all* of its arguments, except for the *-n* option in the BSD version.

12771 **FUTURE DIRECTIONS**

12772 None.

12773 **SEE ALSO**

12774 *printf*

12775 **CHANGE HISTORY**

12776 First released in Issue 2.

12777 **Issue 5**

12778 In the OPTIONS section, the last sentence is changed to indicate that implementations “do not”  
12779 support any options; in the previous issue this said “need not”.

12780 **Issue 6**

12781 The following new requirements on POSIX implementations derive from alignment with the  
12782 Single UNIX Specification:

- 12783 • A set of character sequences is defined as *string* operands.
- 12784 • *LC\_CTYPE* is added to the list of environment variables affecting *echo*.
- 12785 • In the OPTIONS section, implementations shall not support any options.

12786 **NAME**

12787           ed — edit text

12788 **SYNOPSIS**12789           ed [-p *string*][-s][*file*]12790 **DESCRIPTION**

12791           The *ed* utility is a line-oriented text editor that uses two modes: *command mode* and *input mode*.  
 12792           In command mode the input characters shall be interpreted as commands, and in input mode  
 12793           they shall be interpreted as text. See the EXTENDED DESCRIPTION section.

12794 **OPTIONS**

12795           The *ed* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2,  
 12796           Utility Syntax Guidelines.

12797           The following options shall be supported:

12798           -p *string*    Use *string* as the prompt string when in command mode. By default, there shall be  
 12799                           no prompt string.

12800           -s            Suppress the writing of byte counts by **e**, **E**, **r**, and **w** commands and of the '!'  
 12801                           prompt after a *!command*.

12802 **OPERANDS**

12803           The following operand shall be supported:

12804           *file*         If the *file* argument is given, *ed* shall simulate an **e** command on the file named by  
 12805                           the pathname, *file*, before accepting commands from the standard input. If the *file*  
 12806                           operand is '-', the results are unspecified.

12807 **STDIN**

12808           The standard input shall be a text file consisting of commands, as described in the EXTENDED  
 12809           DESCRIPTION section.

12810 **INPUT FILES**

12811           The input files shall be text files.

12812 **ENVIRONMENT VARIABLES**

12813           The following environment variables shall affect the execution of *ed*:

12814           HOME         Determine the pathname of the user's home directory.

12815           LANG         Provide a default value for the internationalization variables that are unset or null.  
 12816                           (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
 12817                           Internationalization Variables for the precedence of internationalization variables  
 12818                           used to determine the values of locale categories.)

12819           LC\_ALL        If set to a non-empty string value, override the values of all the other  
 12820                           internationalization variables.

12821           LC\_COLLATE    Determine the locale for the behavior of ranges, equivalence classes, and multi-  
 12822                           character collating elements within regular expressions.  
 12823

12824           LC\_CTYPE     Determine the locale for the interpretation of sequences of bytes of text data as  
 12825                           characters (for example, single-byte as opposed to multi-byte characters in  
 12826                           arguments and input files) and the behavior of character classes within regular  
 12827                           expressions.

12828           LC\_MESSAGES   Determine the locale that should be used to affect the format and contents of  
 12829

- 12830 diagnostic messages written to standard error and informative messages written to  
12831 standard output.
- 12832 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 12833 **ASYNCHRONOUS EVENTS**
- 12834 The *ed* utility shall take the standard action for all signals (see the ASYNCHRONOUS EVENTS  
12835 section in Section 1.11 (on page 2221)) with the following exceptions:
- 12836 **SIGINT** The *ed* utility shall interrupt its current activity, write the string "?\n" to standard  
12837 output, and return to command mode (see the EXTENDED DESCRIPTION  
12838 section).
- 12839 **SIGHUP** If the buffer is not empty and has changed since the last write, the *ed* utility shall  
12840 attempt to write a copy of the buffer in a file. First, the file named **ed.hup** in the  
12841 current directory shall be used; if that fails, the file named **ed.hup** in the directory  
12842 named by the *HOME* environment variable shall be used. In any case, the *ed* utility  
12843 shall exit without returning to command mode.
- 12844 **SIGQUIT** The *ed* utility shall ignore this event.
- 12845 **STDOUT**
- 12846 Various editing commands and the prompting feature (see **-p**) write to standard output, as  
12847 described in the EXTENDED DESCRIPTION section.
- 12848 **STDERR**
- 12849 The standard error shall be used only for diagnostic messages.
- 12850 **OUTPUT FILES**
- 12851 The output files shall be text files whose formats are dependent on the editing commands given.
- 12852 **EXTENDED DESCRIPTION**
- 12853 The *ed* utility shall operate on a copy of the file it is editing; changes made to the copy shall have  
12854 no effect on the file until a **w** (write) command is given. The copy of the text is called the *buffer*.
- 12855 Commands to *ed* have a simple and regular structure: zero, one, or two *addresses* followed by a  
12856 single-character *command*, possibly followed by parameters to that command. These addresses  
12857 specify one or more lines in the buffer. Every command that requires addresses has default  
12858 addresses, so that the addresses very often can be omitted. If the **-p** option is specified, the  
12859 prompt string shall be written to standard output before each command is read.
- 12860 In general, only one command can appear on a line. Certain commands allow text to be input.  
12861 This text is placed in the appropriate place in the buffer. While *ed* is accepting text, it is said to be  
12862 in *input mode*. In this mode, no commands shall be recognized; all input is merely collected.  
12863 Input mode is terminated by entering a line consisting of two characters: a period ( '.' )  
12864 followed by a <newline>. This line is not considered part of the input text.
- 12865 **Regular Expressions in ed**
- 12866 The *ed* utility shall support basic regular expressions, as described in the Base Definitions  
12867 volume of IEEE Std 1003.1-200x, Section 9.3, Basic Regular Expressions. Since regular  
12868 expressions in *ed* are always matched against single lines (excluding the terminating  
12869 <newline>s), never against any larger section of text, there is no way for a regular expression to  
12870 match a <newline>.
- 12871 A null RE shall be equivalent to the last RE encountered.
- 12872 Regular expressions are used in addresses to specify lines, and in some commands (for example,  
12873 the **s** substitute command) to specify portions of a line to be substituted.

12874

**Addresses in ed**

12875

12876

12877

Addressing in *ed* relates to the current line. Generally, the current line is the last line affected by a command. The current line number is the address of the current line. If the edit buffer is not empty, the initial value for the current line shall be the last line in the edit buffer; otherwise, zero.

12878

Addresses shall be constructed as follows:

12879

1. The period character ( `' . '` ) shall address the current line.

12880

2. The dollar sign character ( `' $ '` ) shall address the last line of the edit buffer.

12881

3. The positive decimal number *n* shall address the *n*th line of the edit buffer.

12882

12883

12884

12885

4. The apostrophe-x character pair ( `" ' x "` ) shall address the line marked with the mark name character *x*, which shall be a lowercase letter from the portable character set. It shall be an error if the character has not been set to mark a line or if the line that was marked is not currently present in the edit buffer.

12886

12887

12888

12889

12890

12891

12892

12893

12894

12895

5. A BRE enclosed by slash characters ( `' / '` ) shall address the first line found by searching forwards from the line following the current line toward the end of the edit buffer and stopping at the first line for which the line excluding the terminating `<newline>` matches the BRE. The BRE consisting of a null BRE delimited by a pair of slash characters shall address the next line for which the line excluding the terminating `<newline>` matches the last BRE encountered. In addition, the second slash can be omitted at the end of a command line. Within the BRE, a backslash-slash pair ( `" \ / "` ) shall represent a literal slash instead of the BRE delimiter. If necessary, the search shall wrap around to the beginning of the buffer and continue up to and including the current line, so that the entire buffer is searched.

12896

12897

12898

12899

12900

12901

12902

12903

12904

12905

6. A BRE enclosed by question-mark characters ( `' ? '` ) shall address the first line found by searching backwards from the line preceding the current line toward the beginning of the edit buffer and stopping at the first line for which the line excluding the terminating `<newline>` matches the BRE. The BRE consisting of a null BRE delimited by a pair of question-mark characters ( `" ? ? "` ) shall address the previous line for which the line excluding the terminating `<newline>` matches the last BRE encountered. In addition, the second question-mark can be omitted at the end of a command line. Within the BRE, a backslash-question-mark pair ( `" \ ? "` ) shall represent a literal question mark instead of the BRE delimiter. If necessary, the search shall wrap around to the end of the buffer and continue up to and including the current line, so that the entire buffer is searched.

12906

12907

12908

7. A plus-sign ( `' + '` ) or hyphen character ( `' - '` ) followed by a decimal number shall address the current line plus or minus the number. A plus-sign or hyphen character not followed by a decimal number shall address the current line plus or minus 1.

12909

12910

Addresses can be followed by zero or more address offsets, optionally `<blank>`-separated. Address offsets are constructed as follows:

12911

12912

12913

- A plus-sign or hyphen character followed by a decimal number shall add or subtract, respectively, the indicated number of lines to or from the address. A plus-sign or hyphen character not followed by a decimal number shall add or subtract 1 to or from the address.

12914

- A decimal number shall add the indicated number of lines to the address.

12915

12916

12917

12918

It shall not be an error for an intermediate address value to be less than zero or greater than the last line in the edit buffer. It shall be an error for the final address value to be less than zero or greater than the last line in the edit buffer. It shall be an error if a search for a BRE fails to find a matching line.

12919 Commands accept zero, one, or two addresses. If more than the required number of addresses  
 12920 are provided to a command that requires zero addresses, it shall be an error. Otherwise, if more  
 12921 than the required number of addresses are provided to a command, the addresses specified first  
 12922 shall be evaluated and then discarded until the maximum number of valid addresses remain, for  
 12923 the specified command.

12924 Addresses shall be separated from each other by a comma (',') or semicolon character (';').  
 12925 In the case of a semicolon separator, the current line ('.') shall be set to the first address, and  
 12926 only then will the second address be calculated. This feature can be used to determine the  
 12927 starting line for forwards and backwards searches; see rules 5. and 6.

12928 Addresses can be omitted on either side of the comma or semicolon separator, in which case the  
 12929 resulting address pairs shall be as follows:

12930

| Specified | Resulting   |
|-----------|-------------|
| ,         | 1 , \$      |
| , addr    | 1 ,a ddr    |
| addr ,    | addr , addr |
| ;         | . ; \$      |
| ; addr    | . ; addr    |
| addr ;    | addr ; addr |

12931

12932

12933

12934

12935

12936

12937 Any <blank>s included between addresses, address separators, or address offsets shall be  
 12938 ignored.

12939

### Commands in ed

12940 In the following list of *ed* commands, the default addresses are shown in parentheses. The  
 12941 number of addresses shown in the default shall be the number expected by the command. The  
 12942 parentheses are not part of the address; they show that the given addresses are the default.

12943 It is generally invalid for more than one command to appear on a line. However, any command  
 12944 (except **e**, **E**, **f**, **q**, **Q**, **r**, **w**, and **!**) can be suffixed by the letter **l**, **n**, or **p**; in which case, except for  
 12945 the **l**, **n**, and **p** commands, the command shall be executed and then the new current line shall be  
 12946 written as described below under the **l**, **n**, and **p** commands. When an **l**, **n**, or **p** suffix is used  
 12947 with an **l**, **n**, or **p** command, the command shall write to standard output as described below, but  
 12948 it is unspecified whether the suffix writes the current line again in the requested format or  
 12949 whether the suffix has no effect. For example, the **pl** command (base **p** command with an **l**  
 12950 suffix) shall either write just the current line or write it twice—once as specified for **p** and once  
 12951 as specified for **l**. Also, the **g**, **G**, **v**, and **V** commands shall take a command as a parameter.

12952 Each address component can be preceded by zero or more <blank>s. The command letter can be  
 12953 preceded by zero or more <blank>s. If a suffix letter (**l**, **n**, or **p**) is given, the application shall  
 12954 ensure that it immediately follows the command.

12955 The **e**, **E**, **f**, **r**, and **w** commands shall take an optional *file* parameter, separated from the  
 12956 command letter by one or more <blank>s.

12957 If changes have been made in the buffer since the last **w** command that wrote the entire buffer,  
 12958 *ed* shall warn the user if an attempt is made to destroy the editor buffer via the **e** or **q** commands.  
 12959 The *ed* utility shall write the string:

12960 "?\n"

12961 (followed by an explanatory message if *help mode* has been enabled via the **H** command) to  
 12962 standard output and shall continue in command mode with the current line number unchanged.  
 12963 If the **e** or **q** command is repeated with no intervening command, it shall take effect.

12964 If a terminal disconnect is detected:

- 12965 • If the buffer is not empty and has changed since the last write, the *ed* utility shall attempt to
- 12966 write a copy of the buffer to a file named **ed.hup** in the current directory. If this write fails, *ed*
- 12967 shall attempt to write a copy of the buffer to a filename **ed.hup** in the directory named by the
- 12968 *HOME* environment variable. If both these attempts fail, *ed* shall exit without saving the
- 12969 buffer.
- 12970 • The *ed* utility shall not write the file to the currently remembered pathname or return to
- 12971 command mode, and shall terminate with a non-zero exit status.

12972 If an end-of-file is detected on standard input:

- 12973 • If the *ed* utility is in input mode, *ed* shall terminate input mode and return to command mode.
- 12974 It is unspecified if any partially entered lines (that is, input text without a terminating
- 12975 <newline>) are discarded from the input text.
- 12976 • If the *ed* utility is in command mode, it shall act as if a **q** command had been entered.

12977 If the closing delimiter of an RE or of a replacement string (for example, ' / ') in a **g**, **G**, **s**, **v**, or **V**

12978 command would be the last character before a <newline>, that delimiter can be omitted, in

12979 which case the addressed line shall be written. For example, the following pairs of commands

12980 are equivalent:

```
12981 s/s1/s2      s/s1/s2/p
12982 g/s1         g/s1/p
12983 ?s1         ?s1?
```

12984 If an invalid command is entered, *ed* shall write the string:

```
12985 "?\n"
```

12986 (followed by an explanatory message if *help mode* has been enabled via the **H** command) to

12987 standard output and shall continue in command mode with the current line number unchanged.

## 12988 Append Command

```
12989 Synopsis:      (.)a
12990                <text>
12991                .
```

12992 The **a** command shall read the given text and append it after the addressed line; the current line

12993 number shall become the address of the last inserted line or, if there were none, the addressed

12994 line. Address 0 shall be valid for this command; it shall cause the appended text to be placed at

12995 the beginning of the buffer.

## 12996 Change Command

```
12997 Synopsis:      (.,.)c
12998                <text>
12999                .
```

13000 The **c** command shall delete the addressed lines, then accept input text that replaces these lines;

13001 the current line shall be set to the address of the last line input; or, if there were none, at the line

13002 after the last line deleted; if the lines deleted were originally at the end of the buffer, the current

13003 line number shall be set to the address of the new last line; if no lines remain in the buffer, the

13004 current line number shall be set to zero. Address 0 shall be valid for this command; it shall be

13005 interpreted as if address 1 were specified.

13006 **Delete Command**13007 *Synopsis:* (.,.)d

13008 The **d** command shall delete the addressed lines from the buffer. The address of the line after the  
 13009 last line deleted shall become the current line number; if the lines deleted were originally at the  
 13010 end of the buffer, the current line number shall be set to the address of the new last line; if no  
 13011 lines remain in the buffer, the current line number shall be set to zero.

13012 **Edit Command**13013 *Synopsis:* e [*file*]

13014 The **e** command shall delete the entire contents of the buffer and then read in the file named by  
 13015 the pathname *file*. The current line number shall be set to the address of the last line of the  
 13016 buffer. If no pathname is given, the currently remembered pathname, if any, shall be used (see  
 13017 the **f** command). The number of bytes read shall be written to standard output, unless the **-s**  
 13018 option was specified, in the following format:

13019 "%d\n", &lt;number of bytes read&gt;

13020 The name *file* shall be remembered for possible use as a default pathname in subsequent **e**, **E**, **r**,  
 13021 and **w** commands. If *file* is replaced by **'!'**, the rest of the line shall be taken to be a shell  
 13022 command line whose output is to be read. Such a shell command line shall not be remembered  
 13023 as the current *file*. All marks shall be discarded upon the completion of a successful **e** command.  
 13024 If the buffer has changed since the last time the entire buffer was written, the user shall be  
 13025 warned, as described previously.

13026 **Edit Without Checking Command**13027 *Synopsis:* E [*file*]

13028 The **E** command shall possess all properties and restrictions of the **e** command except that the  
 13029 editor shall not check to see whether any changes have been made to the buffer since the last **w**  
 13030 command.

13031 **Filename Command**13032 *Synopsis:* f [*file*]

13033 If *file* is given, the **f** command shall change the currently remembered pathname to *file*; whether  
 13034 the name is changed or not, it shall then write the (possibly new) currently remembered  
 13035 pathname to the standard output in the following format:

13036 "%s\n", &lt;pathname&gt;

13037 The current line number shall be unchanged.

13038 **Global Command**13039 *Synopsis:* (1,\$)g/RE/command list

13040 In the **g** command, the first step shall be to mark every line for which the line excluding the  
 13041 terminating <newline> matches the given *RE*. Then, going sequentially from the beginning of  
 13042 the file to the end of the file, the given *command list* shall be executed for each marked line, with  
 13043 the current line number set to the address of that line. Any line modified by the *command list*  
 13044 shall be unmarked. When the **g** command completes, the current line number shall have the  
 13045 value assigned by the last command in the *command list*. If there were no matching lines, the  
 13046 current line number shall not be changed. A single command or the first of a list of commands

13047 shall appear on the same line as the global command. All lines of a multi-line list except the last  
 13048 line shall be ended with a backslash preceding the terminating <newline>; the **a**, **i**, and **c**  
 13049 commands and associated input are permitted. The **' . '** terminating input mode can be omitted  
 13050 if it would be the last line of the *command list*. An empty *command list* shall be equivalent to the **p**  
 13051 command. The use of the **g**, **G**, **v**, **V**, and **!** commands in the *command list* produces undefined  
 13052 results. Any character other than <space> or <newline> can be used instead of a slash to delimit  
 13053 the *RE*. Within the *RE*, the *RE* delimiter itself can be used as a literal character if it is preceded  
 13054 by a backslash.

### 13055 **Interactive Global Command**

13056 *Synopsis:* (1, \$)G/*RE*/

13057 In the **G** command, the first step shall be to mark every line for which the line excluding the  
 13058 terminating <newline> matches the given *RE*. Then, for every such line, that line shall be  
 13059 written, the current line number shall be set to the address of that line, and any one command  
 13060 (other than one of the **a**, **c**, **i**, **g**, **G**, **v**, and **V** commands) shall be read and executed. A <newline>  
 13061 shall act as a null command (causing no action to be taken on the current line); an **' & '** shall  
 13062 cause the re-execution of the most recent non-null command executed within the current  
 13063 invocation of **G**. Note that the commands input as part of the execution of the **G** command can  
 13064 address and affect any lines in the buffer. The final value of the current line number shall be the  
 13065 value set by the last command successfully executed. (Note that the last command successfully  
 13066 executed shall be the **G** command itself if a command fails or the null command is specified.) If  
 13067 there were no matching lines, the current line number shall not be changed. The **G** command can  
 13068 be terminated by a SIGINT signal. Any character other than <space> or <newline> can be used  
 13069 instead of a slash to delimit the *RE* and the replacement. Within the *RE*, the *RE* delimiter itself  
 13070 can be used as a literal character if it is preceded by a backslash.

### 13071 **Help Command**

13072 *Synopsis:* h

13073 The **h** command shall write a short message to standard output that explains the reason for the  
 13074 most recent **' ? '** notification. The current line number shall be unchanged.

### 13075 **Help-Mode Command**

13076 *Synopsis:* H

13077 The **H** command shall cause *ed* to enter a mode in which help messages (see the **h** command)  
 13078 shall be written to standard output for all subsequent **' ? '** notifications. The **H** command  
 13079 alternately shall turn this mode on and off; it is initially off. If the help-mode is being turned on,  
 13080 the **H** command also explains the previous **' ? '** notification, if there was one. The current line  
 13081 number shall be unchanged.

### 13082 **Insert Command**

13083 *Synopsis:* (.)i  
 13084 <text>  
 13085 .

13086 The **i** command shall insert the given text before the addressed line; the current line is set to the  
 13087 last inserted line or, if there was none, to the addressed line. This command differs from the **a**  
 13088 command only in the placement of the input text. Address 0 shall be valid for this command; it  
 13089 shall be interpreted as if address 1 were specified.

13090 **Join Command**13091 *Synopsis:* ( . , .+1 ) j

13092 The **j** command shall join contiguous lines by removing the appropriate <newline>s. If exactly  
 13093 one address is given, this command shall do nothing. If lines are joined, the current line number  
 13094 shall be set to the address of the joined line; otherwise, the current line number shall be  
 13095 unchanged.

13096 **Mark Command**13097 *Synopsis:* ( . ) k x

13098 The **k** command shall mark the addressed line with name *x*, which the application shall ensure is  
 13099 a lowercase letter from the portable character set. The address " 'x'" shall then refer to this line;  
 13100 the current line number shall be unchanged.

13101 **List Command**13102 *Synopsis:* ( . , . ) l

13103 The **l** command shall write to standard output the addressed lines in a visually unambiguous  
 13104 form. The characters listed in the Base Definitions volume of IEEE Std 1003.1-200x, Table 5-1,  
 13105 Escape Sequences and Associated Actions ( '\', '\a', '\b', '\f', '\r', '\t', '\v' ) shall  
 13106 be written as the corresponding escape sequence; the '\n' in that table is not applicable. Non-  
 13107 printable characters not in the table shall be written as one three-digit octal number (with a  
 13108 preceding backslash character) for each byte in the character (most significant byte first). If the  
 13109 size of a byte on the system is greater than nine bits, the format used for non-printable characters  
 13110 is implementation-defined.

13111 Long lines shall be folded, with the point of folding indicated by <newline> preceded by a |  
 13112 backslash; the length at which folding occurs is unspecified, but should be appropriate for the |  
 13113 output device. The end of each line shall be marked with a '\$', and '\$' characters within the |  
 13114 text shall be written with a preceding backslash. An **l** command can be appended to any other |  
 13115 command other than **e**, **E**, **f**, **q**, **Q**, **r**, **w**, or **!**. The current line number shall be set to the address of  
 13116 the last line written.

13117 **Move Command**13118 *Synopsis:* ( . , . ) *address*

13119 The **m** command shall reposition the addressed lines after the line addressed by *address*.  
 13120 Address 0 shall be valid for *address* and cause the addressed lines to be moved to the beginning  
 13121 of the buffer. It shall be an error if *address* falls within the range of moved lines. The  
 13122 current line number shall be set to the address of the last line moved.

13123 **Number Command**13124 *Synopsis:* ( . , . ) n

13125 The **n** command shall write to standard output the addressed lines, preceding each line by its  
 13126 line number and a <tab>; the current line number shall be set to the address of the last line  
 13127 written. The **n** command can be appended to any command other than **e**, **E**, **f**, **q**, **Q**, **r**, **w**, or **!**.

13128 **Print Command**13129 *Synopsis:* (.,.)p

13130 The **p** command shall write to standard output the addressed lines; the current line number shall  
 13131 be set to the address of the last line written. The **p** command can be appended to any command  
 13132 other than **e**, **E**, **f**, **q**, **Q**, **r**, **w**, or **!**.

13133 **Prompt Command**13134 *Synopsis:* P

13135 The **P** command shall cause *ed* to prompt with an asterisk ( '\*' ) (or *string*, if **-p** is specified) for  
 13136 all subsequent commands. The **P** command alternatively shall turn this mode on and off; it shall  
 13137 be initially on if the **-p** option is specified; otherwise, off. The current line number shall be  
 13138 unchanged.

13139 **Quit Command**13140 *Synopsis:* q

13141 The **q** command shall cause *ed* to exit. If the buffer has changed since the last time the entire  
 13142 buffer was written, the user shall be warned, as described previously.

13143 **Quit Without Checking Command**13144 *Synopsis:* Q

13145 The **Q** command shall cause *ed* to exit without checking whether changes have been made in the  
 13146 buffer since the last **w** command.

13147 **Read Command**13148 *Synopsis:* (\$)r [*file*]

13149 The **r** command shall read in the file named by the pathname *file* and append it after the  
 13150 addressed line. If no *file* argument is given, the currently remembered pathname, if any, shall be  
 13151 used (see the **e** and **f** commands). The currently remembered pathname shall not be changed  
 13152 unless there is no remembered pathname. Address 0 shall be valid for **r** and shall cause the file to  
 13153 be read at the beginning of the buffer. If the read is successful, and **-s** was not specified, the  
 13154 number of bytes read shall be written to standard output in the following format:

13155 "%d\n", &lt;number of bytes read&gt;

13156 The current line number shall be set to the address of the last line read in. If *file* is replaced by  
 13157 '!', the rest of the line shall be taken to be a shell command line whose output is to be read.  
 13158 Such a shell command line shall not be remembered as the current pathname.

13159 **Substitute Command**13160 *Synopsis:* (.,.)s/RE/replacement/flags

13161 The **s** command shall search each addressed line for an occurrence of the specified *RE* and  
 13162 replace either the first or all (non-overlapped) matched strings with the *replacement*; see the  
 13163 following description of the **g** suffix. It is an error if the substitution fails on every addressed  
 13164 line. Any character other than <space> or <newline> can be used instead of a slash to delimit the  
 13165 *RE* and the replacement. Within the *RE*, the *RE* delimiter itself can be used as a literal character if  
 13166 it is preceded by a backslash. The current line shall be set to the address of the last line on which  
 13167 a substitution occurred.

13168 An ampersand ('&') appearing in the *replacement* shall be replaced by the string matching the  
 13169 RE on the current line. The special meaning of '&' in this context can be suppressed by  
 13170 preceding it by backslash. As a more general feature, the characters '\n', where *n* is a digit,  
 13171 shall be replaced by the text matched by the corresponding back-reference expression. When the  
 13172 character '%' is the only character in the *replacement*, the *replacement* used in the most recent  
 13173 substitute command shall be used as the *replacement* in the current substitute command; if there  
 13174 was no previous substitute command, the use of '%' in this manner shall be an error. The '%'  
 13175 shall lose its special meaning when it is in a replacement string of more than one character or is  
 13176 preceded by a backslash. For each backslash ('\') encountered in scanning *replacement* from  
 13177 beginning to end, the following character shall lose its special meaning (if any). It is unspecified  
 13178 what special meaning is given to any character other than '&', '\', '%', or digits.

13179 A line can be split by substituting a <newline> into it. The application shall ensure it escapes the  
 13180 <newline> in the *replacement* by preceding it by backslash. Such substitution cannot be done as  
 13181 part of a **g** or **v** *command list*. The current line number shall be set to the address of the last line  
 13182 on which a substitution is performed. If no substitution is performed, the current line number  
 13183 shall be unchanged. If a line is split, a substitution shall be considered to have been performed  
 13184 on each of the new lines for the purpose of determining the new current line number. A  
 13185 substitution shall be considered to have been performed even if the replacement string is  
 13186 identical to the string that it replaces.

13187 The application shall ensure that the value of *flags* is zero or more of:

13188 *count* Substitute for the *count*th occurrence only of the *RE* found on each addressed line.

13189 **g** Globally substitute for all non-overlapping instances of the *RE* rather than just the first  
 13190 one. If both **g** and *count* are specified, the results are unspecified.

13191 **l** Write to standard output the final line in which a substitution was made. The line shall  
 13192 be written in the format specified for the **l** command.

13193 **n** Write to standard output the final line in which a substitution was made. The line shall  
 13194 be written in the format specified for the **n** command.

13195 **p** Write to standard output the final line in which a substitution was made. The line shall  
 13196 be written in the format specified for the **p** command.

### 13197 **Copy Command**

13198 *Synopsis:* (.,.)*taddress*

13199 The **t** command shall be equivalent to the **m** command, except that a copy of the addressed lines  
 13200 shall be placed after address *address* (which can be 0); the current line number shall be set to the  
 13201 address of the last line added.

### 13202 **Undo Command**

13203 *Synopsis:* u

13204 The **u** command shall nullify the effect of the most recent command that modified anything in  
 13205 the buffer, namely the most recent **a**, **c**, **d**, **g**, **i**, **j**, **m**, **r**, **s**, **t**, **u**, **v**, **G**, or **V** command. All changes  
 13206 made to the buffer by a **g**, **G**, **v**, or **V** global command shall be undone as a single change; if no  
 13207 changes were made by the global command (such as with **g/RE/p**), the **u** command shall have  
 13208 no effect. The current line number shall be set to the value it had immediately before the  
 13209 command being undone started.

13210 **Global Non-Matched Command**13211 *Synopsis:* (1,\$)v/RE/command list13212 This command shall be equivalent to the global command **g** except that the lines that are marked  
13213 during the first step shall be those for which the line excluding the terminating <newline> does  
13214 not match the *RE*.13215 **Interactive Global Not-Matched Command**13216 *Synopsis:* (1,\$)V/RE/13217 This command shall be equivalent to the interactive global command **G** except that the lines that  
13218 are marked during the first step shall be those for which the line excluding the terminating  
13219 <newline> does not match the *RE*.13220 **Write Command**13221 *Synopsis:* (1,\$)w [*file*]13222 The **w** command shall write the addressed lines into the file named by the pathname *file*. The  
13223 command shall create the file, if it does not exist, or shall replace the contents of the existing file.  
13224 The currently remembered pathname shall not be changed unless there is no remembered  
13225 pathname. If no pathname is given, the currently remembered pathname, if any, shall be used  
13226 (see the **e** and **f** commands); the current line number shall be unchanged. If the command is  
13227 successful, the number of bytes written shall be written to standard output, unless the **-s** option  
13228 was specified, in the following format:

13229 "%d\n", &lt;number of bytes written&gt;

13230 If *file* begins with '!', the rest of the line shall be taken to be a shell command line whose  
13231 standard input shall be the addressed lines. Such a shell command line shall not be remembered  
13232 as the current pathname. This usage of the write command with '!' shall not be considered as a  
13233 "last **w** command that wrote the entire buffer", as described previously; thus, this alone shall not  
13234 prevent the warning to the user if an attempt is made to destroy the editor buffer via the **e** or **q**  
13235 commands.13236 **Line Number Command**13237 *Synopsis:* (\$)=13238 The line number of the addressed line shall be written to standard output in the following  
13239 format:

13240 "%d\n", &lt;line number&gt;

13241 The current line number shall be unchanged by this command.

13242 **Shell Escape Command**13243 *Synopsis:* !command13244 The remainder of the line after the '!' shall be sent to the command interpreter to be  
13245 interpreted as a shell command line. Within the text of that shell command line, the unescaped  
13246 character '%' shall be replaced with the remembered pathname; if a '!' appears as the first  
13247 character of the command, it shall be replaced with the text of the previous shell command  
13248 executed via '!'. Thus, "!!" shall repeat the previous !command. If any replacements of '%' or  
13249 '!' are performed, the modified line shall be written to the standard output before *command* is  
13250 executed. The '!' command shall write:

- 13251 "!\n"
- 13252 to standard output upon completion, unless the `-s` option is specified. The current line number  
13253 shall be unchanged.
- 13254 **Null Command**
- 13255 *Synopsis:* ( .+1 )
- 13256 An address alone on a line shall cause the addressed line to be written. A <newline> alone shall  
13257 be equivalent to "+1p". The current line number shall be set to the address of the written line.
- 13258 **EXIT STATUS**
- 13259 The following exit values shall be returned:
- 13260 0 Successful completion without any file or command errors.
- 13261 >0 An error occurred.
- 13262 **CONSEQUENCES OF ERRORS**
- 13263 When an error in the input script is encountered, or when an error is detected that is a  
13264 consequence of the data (not) present in the file or due to an external condition such as a read or  
13265 write error:
- 13266 • If the standard input is a terminal device file, all input shall be flushed, and a new command  
13267 read.
  - 13268 • If the standard input is a regular file, *ed* shall terminate with a non-zero exit status.
- 13269 **APPLICATION USAGE**
- 13270 Because of the extremely terse nature of the default error messages, the prudent script writer  
13271 begins the *ed* input commands with an **H** command, so that if any errors do occur at least some  
13272 clue as to the cause is made available.
- 13273 In previous versions, an obsolescent `-` option was described. This is no longer specified.  
13274 Applications should use the `-s` option. Using `-` as a *file* operand now produces unspecified  
13275 results. This allows implementations to continue to support the former required behavior.
- 13276 **EXAMPLES**
- 13277 None.
- 13278 **RATIONALE**
- 13279 The initial description of this utility was adapted from the SVID. It contains some features not  
13280 found in Version 7 or BSD-derived systems. Some of the differences between the POSIX and  
13281 BSD *ed* utilities include, but need not be limited to:
- 13282 • The BSD `-` option does not suppress the `'!'` prompt after a `!` command.
  - 13283 • BSD does not support the special meanings of the `'%'` and `'!'` characters within a `!`  
13284 command.
  - 13285 • BSD does not support the *addresses* `' ; '` and `' , '`.
  - 13286 • BSD allows the command/suffix pairs **pp**, **ll**, and so on, which are unspecified in this volume  
13287 of IEEE Std 1003.1-200x.
  - 13288 • BSD does not support the `'!'` character part of the **e**, **r**, or **w** commands.
  - 13289 • A failed **g** command in BSD sets the line number to the last line searched if there are no  
13290 matches.

- 13291           • BSD does not default the *command list* to the **p** command.
- 13292           • BSD does not support the **G**, **h**, **H**, **n**, or **V** commands.
- 13293           • On BSD, if there is no inserted text, the insert command changes the current line to the  
13294           referenced line -1; that is, the line before the specified line.
- 13295           • On BSD, the *join* command with only a single address changes the current line to that  
13296           address.
- 13297           • BSD does not support the **P** command; moreover, in BSD it is synonymous with the **p**  
13298           command.
- 13299           • BSD does not support the *undo* of the commands **j**, **m**, **r**, **s**, or **t**.
- 13300           • The Version 7 *ed* command **W**, and the BSD *ed* commands **W**, **wq**, and **z** are not present in this  
13301           volume of IEEE Std 1003.1-200x.
- 13302           The **-s** option was added to allow the functionality of the now withdrawn **-** option in a manner  
13303           compatible with the Utility Syntax Guidelines.
- 13304           In early proposals there was a limit, {ED\_FILE\_MAX}, that described the historical limitations of  
13305           some *ed* utilities in their handling of large files; some of these have had problems with files larger  
13306           than 100 000 bytes. It was this limitation that prompted much of the desire to include a *split*  
13307           command in this volume of IEEE Std 1003.1-200x. Since this limit was removed, this volume of  
13308           IEEE Std 1003.1-200x requires that implementations document the file size limits imposed by *ed*  
13309           in the conformance document. The limit {ED\_LINE\_MAX} was also removed; therefore, the  
13310           global limit {LINE\_MAX} is used for input and output lines.
- 13311           The manner in which the **l** command writes non-printable characters was changed to avoid the  
13312           historical backspace-overstrike method. On video display terminals, the overstrike is ambiguous  
13313           because most terminals simply replace overstruck characters, making the **l** format not useful for  
13314           its intended purpose of unambiguously understanding the content of the line. The historical  
13315           backslash escapes were also ambiguous. (The string "a\0011" could represent a line containing  
13316           those six characters or a line containing the three characters 'a', a byte with a binary value of 1,  
13317           and a 1.) In the format required here, a backslash appearing in the line is written as '\\ ' so that  
13318           the output is truly unambiguous. The method of marking the ends of lines was adopted from the  
13319           *ex* editor and is required for any line ending in <space>s; the '\$ ' is placed on all lines so that a  
13320           real '\$ ' at the end of a line cannot be misinterpreted.
- 13321           Systems with bytes too large to fit into three octal digits must devise other means of displaying  
13322           non-printable characters. Consideration was given to requiring that the number of octal digits be  
13323           large enough to hold a byte, but this seemed to be too confusing for applications on the vast  
13324           majority of systems where three digits are adequate. It would be theoretically possible for the  
13325           application to use the *getconf* utility to find out the CHAR\_BIT value and deal with such an  
13326           algorithm; however, there is really no portable way that an application can use the octal values  
13327           of the bytes across various coded character sets, so the additional specification was not  
13328           worthwhile.
- 13329           The description of how a NUL is written was removed. The NUL character cannot be in text  
13330           files, and this volume of IEEE Std 1003.1-200x should not dictate behavior in the case of  
13331           undefined, erroneous input.
- 13332           Unlike some of the other editing utilities, the filenames accepted by the **E**, **e**, **R**, and **r** commands  
13333           are not patterns.
- 13334           Early proposals stated that the **-p** option worked only when standard input was associated with  
13335           a terminal device. This has been changed to conform to historical implementations, thereby  
13336           allowing applications to interpose themselves between a user and the *ed* utility.

13337 The form of the substitute command that uses the **n** suffix was limited in some historical  
 13338 documentation (where this was described incorrectly as “backreferencing”). This limit has been  
 13339 omitted because there is no reason an editor processing lines of {LINE\_MAX} length should have  
 13340 this restriction. The command **s/x/X/2047** should be able to substitute the 2047th occurrence of **x**  
 13341 on a line.

13342 The use of printing commands with printing suffixes (such as **pn**, **lp**, and so on) was made  
 13343 unspecified because BSD-based systems allow this, whereas System V does not.

13344 Some BSD-based systems exit immediately upon receipt of end-of-file if all of the lines in the file  
 13345 have been deleted. Since this volume of IEEE Std 1003.1-200x refers to the **q** command in this  
 13346 instance, such behavior is not allowed.

13347 Some historical implementations returned exit status zero even if command errors had occurred;  
 13348 this is not allowed by this volume of IEEE Std 1003.1-200x.

13349 Some historical implementations contained a bug that allowed a single period to be entered in  
 13350 input mode as <backslash> <period> <newline>. This is not allowed by the *ed* because there is  
 13351 no description of escaping any of the characters in input mode; backslashes are entered into the  
 13352 buffer exactly as typed. The typical method of entering a single period has been to precede it  
 13353 with another character and then use the substitute command to delete that character.

13354 It is difficult under some modes of some versions of historical operating system terminal drivers  
 13355 to distinguish between an end-of-file condition and terminal disconnect. IEEE Std 1003.1-200x  
 13356 does not require implementations to distinguish between the two situations, which permits  
 13357 historical implementations of the *ed* utility on historical platforms to conform. Implementations  
 13358 are encouraged to distinguish between the two, if possible, and take appropriate action on  
 13359 terminal disconnect.

13360 Historically, *ed* accepted a zero address for the **a** and **r** commands in order to insert text at the  
 13361 start of the edit buffer. When the buffer was empty the command **.=** returned zero.  
 13362 IEEE Std 1003.1-200x requires conformance to historical practice.

13363 For consistency with the **a** and **r** commands and better user functionality, the **i** and **c** commands  
 13364 must also accept an address of 0, in which case **0i** is treated as **1i** and likewise for the **c**  
 13365 command.

13366 All of the following are valid addresses:

13367 **+++** Three lines after the current line.

13368 **/pattern/-** One line before the next occurrence of pattern.

13369 **-2** Two lines before the current line.

13370 **3 ---- 2** Line one (note the intermediate negative address).

13371 **1 2 3** Line six.

13372 Any number of addresses can be provided to commands taking addresses; for example,  
 13373 "**1,2,3,4,5p**" prints lines 4 and 5, because two is the greatest valid number of addresses  
 13374 accepted by the **print** command. This, in combination with the semicolon delimiter, permits  
 13375 users to create commands based on ordered patterns in the file. For example, the command  
 13376 "**3;/foo/;+2p**" will display the first line after line 3 that contains the pattern *foo*, plus the next  
 13377 two lines. Note that the address "**3;**" must still be evaluated before being discarded, because  
 13378 the search origin for the **/foo/** command depends on this.

13379 Historically, *ed* disallowed address chains, as discussed above, consisting solely of comma or  
 13380 semicolon separators; for example, "**,, ,**" or "**;; ;**" were considered an error. For consistency of  
 13381 address specification, this restriction is removed. The following table lists some of the address

13382 forms now possible:

|       | Address | Addr1 | Addr2 | Status     | Comment               |
|-------|---------|-------|-------|------------|-----------------------|
| 13384 | 7,      | 7     | 7     | Historical |                       |
| 13385 | 7,5,    | 5     | 5     | Historical |                       |
| 13386 | 7,5,9   | 5     | 9     | Historical |                       |
| 13387 | 7,9     | 7     | 9     | Historical |                       |
| 13388 | 7,+     | 7     | 8     | Historical |                       |
| 13389 | ,       | 1     | \$    | Historical |                       |
| 13390 | ,7      | 1     | 7     | Extension  |                       |
| 13391 | ,,      | \$    | \$    | Extension  |                       |
| 13392 | ,;      | \$    | \$    | Extension  |                       |
| 13393 | 7;      | 7     | 7     | Historical |                       |
| 13394 | 7;5;    | 5     | 5     | Historical |                       |
| 13395 | 7;5;9   | 5     | 9     | Historical |                       |
| 13396 | 7;5,9   | 5     | 9     | Historical |                       |
| 13397 | 7;\$;4  | \$    | 4     | Historical | Valid, but erroneous. |
| 13398 | 7;9     | 7     | 9     | Historical |                       |
| 13399 | 7;+     | 7     | 8     | Historical |                       |
| 13400 | ;       | .     | \$    | Historical |                       |
| 13401 | ;7      | .     | 7     | Extension  |                       |
| 13402 | ;;      | \$    | \$    | Extension  |                       |
| 13403 | ;,      | \$    | \$    | Extension  |                       |

13404 Historically, values could be added to addresses by including them after one or more <blank>s;  
 13405 for example, "3 - 5p" wrote the seventh line of the file, and "/foo/ 5" was the same as  
 13406 "5 /foo/". However, only absolute values could be added; for example, "5 /foo/" was an  
 13407 error. IEEE Std 1003.1-200x requires conformance to historical practice.

13408 Historically, *ed* accepted the '^' character as an address, in which case it was identical to the  
 13409 hyphen character. IEEE Std 1003.1-200x does not require or prohibit this behavior.

#### 13410 FUTURE DIRECTIONS

13411 None.

#### 13412 SEE ALSO

13413 *ex, sed, sh, vi*

#### 13414 CHANGE HISTORY

13415 First released in Issue 2.

#### 13416 Issue 5

13417 In the OPTIONS section, the meaning of -s and - is clarified.

13418 Second FUTURE DIRECTION added.

#### 13419 Issue 6

13420 The obsolescent single-minus form has been removed.

13421 A second APPLICATION USAGE note has been added.

13422 The Open Group Corrigendum U025/2 is applied, correcting the description of the Edit section.

13423 The *ed* utility is updated to align with the IEEE P1003.2b draft standard. This includes addition of  
 13424 the treatment of the SIGQUIT signal, changes to *ed* addressing, changes to processing when  
 13425 end-of-file is detected and when terminal disconnect is detected.

13426

The normative text is reworded to avoid use of the term “must” for application requirements.

13427 **NAME**

13428           env — set the environment for command invocation

13429 **SYNOPSIS**13430           env [-i][name=value]... [*utility* [*argument*...]]13431 **DESCRIPTION**13432           The *env* utility shall obtain the current environment, modify it according to its arguments, then  
13433           invoke the utility named by the *utility* operand with the modified environment.13434           Optional arguments shall be passed to *utility*.13435           If no *utility* operand is specified, the resulting environment shall be written to the standard  
13436           output, with one *name=value* pair per line.13437 **OPTIONS**13438           The *env* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section  
13439           12.2, Utility Syntax Guidelines.

13440           The following options shall be supported:

13441           -i           Invoke *utility* with exactly the environment specified by the arguments; the  
13442           inherited environment shall be ignored completely.13443 **OPERANDS**

13444           The following operands shall be supported:

13445           *name=value*   Arguments of the form *name=value* shall modify the execution environment, and  
13446           shall be placed into the inherited environment before the *utility* is invoked.13447           *utility*       The name of the utility to be invoked. If the *utility* operand names any of the  
13448           special built-in utilities in Section 2.14 (on page 2266), the results are undefined.13449           *argument*    A string to pass as an argument for the invoked utility.13450 **STDIN**

13451           Not used.

13452 **INPUT FILES**

13453           None.

13454 **ENVIRONMENT VARIABLES**13455           The following environment variables shall affect the execution of *env*:13456           *LANG*         Provide a default value for the internationalization variables that are unset or null.  
13457           (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
13458           Internationalization Variables for the precedence of internationalization variables  
13459           used to determine the values of locale categories.)13460           *LC\_ALL*        If set to a non-empty string value, override the values of all the other  
13461           internationalization variables.13462           *LC\_CTYPE*    Determine the locale for the interpretation of sequences of bytes of text data as  
13463           characters (for example, single-byte as opposed to multi-byte characters in  
13464           arguments).13465           *LC\_MESSAGES*13466           Determine the locale that should be used to affect the format and contents of  
13467           diagnostic messages written to standard error.13468 *XSI*           *NLS\_PATH*    Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

13469            *PATH*            Determine the location of the *utility*, as described in the Base Definitions volume of  
13470                            IEEE Std 1003.1-200x, Chapter 8, Environment Variables. If *PATH* is specified as a  
13471                            *name=value* operand to *env*, the *value* given shall be used in the search for *utility*.

#### 13472 ASYNCHRONOUS EVENTS

13473            Default.

#### 13474 STDOUT

13475            If no *utility* operand is specified, each *name=value* pair in the resulting environment shall be  
13476            written in the form:

13477            "%s=%s\n", <name>, <value>

13478            If the *utility* operand is specified, the *env* utility shall not write to standard output.

#### 13479 STDERR

13480            The standard error shall be used only for diagnostic messages.

#### 13481 OUTPUT FILES

13482            None.

#### 13483 EXTENDED DESCRIPTION

13484            None.

#### 13485 EXIT STATUS

13486            If the *utility* utility is invoked, the exit status of *env* shall be the exit status of *utility*; otherwise,  
13487            the *env* utility shall exit with one of the following values:

13488            0        The *env* utility completed successfully.

13489            1–125    An error occurred in the *env* utility.

13490            126    The utility specified by *utility* was found but could not be invoked.

13491            127    The utility specified by *utility* could not be found.

#### 13492 CONSEQUENCES OF ERRORS

13493            Default.

#### 13494 APPLICATION USAGE

13495            The *command*, *env*, *nice*, *nohup*, *time*, and *xargs* utilities have been specified to use exit code 127 if  
13496            an error occurs so that applications can distinguish “failure to find a utility” from “invoked  
13497            utility exited with an error indication”. The value 127 was chosen because it is not commonly  
13498            used for other meanings; most utilities use small values for “normal error conditions” and the  
13499            values above 128 can be confused with termination due to receipt of a signal. The value 126 was  
13500            chosen in a similar manner to indicate that the utility could be found, but not invoked. Some  
13501            scripts produce meaningful error messages differentiating the 126 and 127 cases. The distinction  
13502            between exit codes 126 and 127 is based on KornShell practice that uses 127 when all attempts to  
13503            *exec* the utility fail with [ENOENT], and uses 126 when any attempt to *exec* the utility fails for  
13504            any other reason.

13505            Historical implementations of the *env* utility use the *execvp()* or *execlp()* functions defined in the  
13506            System Interfaces volume of IEEE Std 1003.1-200x to invoke the specified utility; this provides  
13507            better performance and keeps users from having to escape characters with special meaning to  
13508            the shell. Therefore, shell functions, special built-ins, and built-ins that are only provided by the  
13509            shell are not found.

13510 **EXAMPLES**

13511           The following command:

```
13512           env -i PATH=/mybin mygrep xyz myfile
```

13513           invokes the command *mygrep* with a new *PATH* value as the only entry in its environment. In  
13514           this case, *PATH* is used to locate *mygrep*, which then must reside in **/mybin**.

13515 **RATIONALE**

13516           As with all other utilities that invoke other utilities, this volume of IEEE Std 1003.1-200x only  
13517           specifies what *env* does with standard input, standard output, standard error, input files, and  
13518           output files. If a utility is executed, it is not constrained by the specification of input and output  
13519           by *env*.

13520           The **-i** option was added to allow the functionality of the withdrawn **-** option in a manner  
13521           compatible with the Utility Syntax Guidelines.

13522           Some have suggested that *env* is redundant since the same effect is achieved by:

```
13523           name=value ... utility [ argument ... ]
```

13524           The example is equivalent to *env* when an environment variable is being added to the  
13525           environment of the command, but not when the environment is being set to the given value.

13526           The *env* utility also writes out the current environment if invoked without arguments. There is  
13527           sufficient functionality beyond what the example provides to justify inclusion of *env*.

13528 **FUTURE DIRECTIONS**

13529           None.

13530 **SEE ALSO**

13531           Section 2.5 (on page 2235)

13532 **CHANGE HISTORY**

13533           First released in Issue 2.

## 13534 NAME

13535 ex — text editor

## 13536 SYNOPSIS

13537 UP `ex [-rR][-l][-s | -v][-c command][-t tagstring][-w size][file ...]`

13538

## 13539 DESCRIPTION

13540 The *ex* utility is a line-oriented text editor. There are two other modes of the editor—open and  
 13541 visual—in which screen-oriented editing is available. This is described more fully by the *ex* **open**  
 13542 and **visual** commands and in *vi*.

13543 This section uses the term *edit buffer* to describe the current working text. No specific  
 13544 implementation is implied by this term. All editing changes are performed on the edit buffer,  
 13545 and no changes to it shall affect any file until an editor command writes the file.

13546 Certain terminals do not have all the capabilities necessary to support the complete *ex* definition,  
 13547 such as the full-screen editing commands (*visual mode* or *open mode*). When these commands  
 13548 cannot be supported on such terminals, this condition shall not produce an error message such  
 13549 as “not an editor command” or report a syntax error. The implementation may either accept the  
 13550 commands and produce results on the screen that are the result of an unsuccessful attempt to  
 13551 meet the requirements of this volume of IEEE Std 1003.1-200x or report an error describing the  
 13552 terminal-related deficiency.

## 13553 OPTIONS

13554 The *ex* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2,  
 13555 Utility Syntax Guidelines.

13556 The following options shall be supported:

13557 **-c *command*** Specify an initial command to be executed in the first edit buffer loaded from an  
 13558 existing file (see the EXTENDED DESCRIPTION section). Implementations may  
 13559 support more than a single **-c** option. In such implementations, the specified  
 13560 commands shall be executed in the order specified on the command line.

13561 **-r** Recover the named files (see the EXTENDED DESCRIPTION section). Recovery  
 13562 information for a file shall be saved during an editor or system crash (for example,  
 13563 when the editor is terminated by a signal which the editor can catch), or after the  
 13564 use of an *ex* **preserve** command.

13565 A *crash* in this context is an unexpected failure of the system or utility that requires  
 13566 restarting the failed system or utility. A system crash implies that any utilities  
 13567 running at the time also crash. In the case of an editor or system crash, the number  
 13568 of changes to the edit buffer (since the most recent **preserve** command) that will be  
 13569 recovered is unspecified.

13570 If no *file* operands are given and the **-t** option is not specified, all other options, the  
 13571 *EXINIT* variable, and any *.exrc* files shall be ignored; a list of all recoverable files  
 13572 available to the invoking user shall be written, and the editor shall exit normally  
 13573 without further action.

13574 **-R** Set **readonly** edit option.

13575 **-s** Prepare *ex* for batch use by taking the following actions:

- 13576 • Suppress writing prompts and informational (but not diagnostic) messages.
- 13577 • Ignore the value of *TERM* and any implementation default terminal type and
- 13578 assume the terminal is a type incapable of supporting open or visual modes;

- 13579                   see the **visual** command and the description of *vi*.
- 13580                   • Suppress the use of the *EXINIT* environment variable and the reading of any  
13581                   *.exrc* file; see the EXTENDED DESCRIPTION section.
- 13582                   • Suppress autoindentation, ignoring the value of the **autoindent** edit option.
- 13583           **-t tagstring** Edit the file containing the specified *tagstring*; see *ctags*. The tags feature  
13584                   represented by **-t tagstring** and the **tag** command is optional. It shall be provided  
13585                   on any system that also provides a conforming implementation of *ctags*; otherwise,  
13586                   the use of **-t** produces undefined results. On any system, it shall be an error to  
13587                   specify more than a single **-t** option.
- 13588           **-v**           Begin in visual mode (see *vi*).
- 13589           **-w size**       Set the value of the *window* editor option to *size*.
- 13590 **OPERANDS**
- 13591           The following operand shall be supported:
- 13592           **file**           A pathname of a file to be edited.
- 13593 **STDIN**
- 13594           The standard input consists of a series of commands and input text, as described in the  
13595           EXTENDED DESCRIPTION section. The implementation may limit each line of standard input  
13596           to a length of {LINE\_MAX}.
- 13597           If the standard input is not a terminal device, it shall be as if the **-s** option had been specified.
- 13598           If a read from the standard input returns an error, or if the editor detects an end-of-file condition  
13599           from the standard input, it shall be equivalent to a SIGHUP asynchronous event.
- 13600 **INPUT FILES**
- 13601           Input files shall be text files or files that would be text files except for an incomplete last line that  
13602           is not longer than {LINE\_MAX}-1 bytes in length and contains no NUL characters. By default,  
13603           any incomplete last line shall be treated as if it had a trailing <newline>. The editing of other  
13604           forms of files may optionally be allowed by *ex* implementations.
- 13605           The *.exrc* files and source files shall be text files consisting of *ex* commands; see the EXTENDED  
13606           DESCRIPTION section.
- 13607           By default, the editor shall read lines from the files to be edited without interpreting any of those  
13608           lines as any form of editor command.
- 13609 **ENVIRONMENT VARIABLES**
- 13610           The following environment variables shall affect the execution of *ex*:
- 13611           **COLUMNS**   Override the system-selected horizontal screen size. See the Base Definitions  
13612                   volume of IEEE Std 1003.1-200x, Chapter 8, Environment Variables for valid values  
13613                   and results when it is unset or null.
- 13614           **EXINIT**       Determine a list of *ex* commands that are executed on editor start-up. See the  
13615                   EXTENDED DESCRIPTION section for more details of the initialization phase.
- 13616           **HOME**         Determine a pathname of a directory that shall be searched for an editor start-up  
13617                   file named *.exrc*; see the EXTENDED DESCRIPTION section.
- 13618           **LANG**         Provide a default value for the internationalization variables that are unset or null.  
13619                   (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
13620                   Internationalization Variables for the precedence of internationalization variables  
13621                   used to determine the values of locale categories.)

- 13622 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
13623 internationalization variables.
- 13624 *LC\_COLLATE*  
13625 Determine the locale for the behavior of ranges, equivalence classes, and multi-  
13626 character collating elements within regular expressions.
- 13627 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
13628 characters (for example, single-byte as opposed to multi-byte characters in  
13629 arguments and input files), the behavior of character classes within regular  
13630 expressions, the classification of characters as uppercase or lowercase letters, the  
13631 case conversion of letters, and the detection of word boundaries.
- 13632 *LC\_MESSAGES*  
13633 Determine the locale that should be used to affect the format and contents of  
13634 diagnostic messages written to standard error.
- 13635 *LINES* Override the system-selected vertical screen size, used as the number of lines in a  
13636 screenful and the vertical screen size in visual mode. See the Base Definitions  
13637 volume of IEEE Std 1003.1-200x, Chapter 8, Environment Variables for valid values  
13638 and results when it is unset or null.
- 13639 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 13640 *PATH* Determine the search path for the shell command specified in the *ex* editor  
13641 commands **!**, **shell**, **read**, and **write**, and the open and visual mode command **!**; see  
13642 the description of command search and execution in Section 2.9.1.1 (on page 2249).
- 13643 *SHELL* Determine the preferred command line interpreter for use as the default value of  
13644 the **shell** edit option.
- 13645 *TERM* Determine the name of the terminal type. If this variable is unset or null, an  
13646 unspecified default terminal type shall be used.
- 13647 **ASYNCHRONOUS EVENTS**  
13648 The following term is used in this and following sections to specify command and asynchronous  
13649 event actions:
- 13650 *complete write*  
13651 A complete write is a write of the entire contents of the edit buffer to a file of a type  
13652 other than a terminal device, or the saving of the edit buffer caused by the user  
13653 executing the *ex* **preserve** command. Writing the contents of the edit buffer to a  
13654 temporary file that will be removed when the editor exits shall not be considered a  
13655 complete write.
- 13656 The following actions shall be taken upon receipt of signals:
- 13657 **SIGINT** If the standard input is not a terminal device, *ex* shall not write the file or return to  
13658 command or text input mode, and shall exit with a non-zero exit status.
- 13659 Otherwise, if executing an open or visual text input mode command, *ex* in receipt  
13660 of **SIGINT** shall behave identically to its receipt of the <ESC> character.
- 13661 Otherwise:
- 13662 1. If executing an *ex* text input mode command, all input lines that have been  
13663 completely entered shall be resolved into the edit buffer, and any partially  
13664 entered line shall be discarded.

- 13665 2. If there is a currently executing command, it shall be aborted and a message  
13666 displayed. Unless otherwise specified by the *ex* or *vi* command descriptions,  
13667 it is unspecified whether any lines modified by the executing command  
13668 appear modified, or as they were before being modified by the executing  
13669 command, in the buffer.
- 13670 If the currently executing command was a motion command, its associated  
13671 command shall be discarded.
- 13672 3. If in open or visual command mode, the terminal shall be alerted.
- 13673 4. The editor shall then return to command mode.
- 13674 SIGCONT The screen shall be refreshed if in open or visual mode.
- 13675 SIGHUP If the edit buffer has been modified since the last complete write, *ex* shall attempt  
13676 to save the edit buffer so that it can be recovered later using the *-r* option or the *ex*  
13677 **recover** command. The editor shall not write the file or return to command or text  
13678 input mode, and shall terminate with a non-zero exit status.
- 13679 SIGTERM Refer to SIGHUP.
- 13680 The action taken for all other signals is unspecified.
- 13681 **STDOUT**
- 13682 The standard output shall be used only for writing prompts to the user, for informational  
13683 messages, and for writing lines from the file.
- 13684 **STDERR**
- 13685 The standard error shall be used only for diagnostic messages.
- 13686 **OUTPUT FILES**
- 13687 The output from *ex* shall be text files.
- 13688 **EXTENDED DESCRIPTION**
- 13689 Only the *ex* mode of the editor is described in this section. See *vi* for additional editing  
13690 capabilities available in *ex*.
- 13691 When an error occurs, *ex* shall write a message. If the terminal supports a standout mode (such  
13692 as inverse video), the message shall be written in standout mode. If the terminal does not  
13693 support a standout mode, and the edit option **errorbells** is set, an alert action shall precede the  
13694 error message.
- 13695 By default, *ex* shall start in command mode, which shall be indicated by a **:** prompt; see the  
13696 **prompt** command. Text input mode can be entered by the **append**, **insert**, or **change** commands;  
13697 it can be exited (and command mode re-entered) by typing a period ( **.** ) alone at the beginning  
13698 of a line.
- 13699 **Initialization in *ex* and *vi***
- 13700 The following symbols are used in this and following sections to specify locations in the edit  
13701 buffer:
- 13702 *alternate and current path names*
- 13703 Two pathnames, named *current* and *alternate*, are maintained by the editor. Any *ex*  
13704 commands that take filenames as arguments shall set them as follows:
- 13705 1. If a *file* argument is specified to the *ex* **edit**, **ex**, or **recover** commands, or if an *ex* **tag**  
13706 command replaces the contents of the edit buffer.

- 13707                   a. If the command replaces the contents of the edit buffer, the current pathname  
13708 shall be set to the *file* argument or the file indicated by the tag, and the alternate  
13709 pathname shall be set to the previous value of the current pathname.
- 13710                   b. Otherwise, the alternate pathname shall be set to the *file* argument.
- 13711           2. If a *file* argument is specified to the **ex next** command:
- 13712                   a. If the command replaces the contents of the edit buffer, the current pathname  
13713 shall be set to the first *file* argument, and the alternate pathname shall be set to  
13714 the previous value of the current pathname.
- 13715           3. If a *file* argument is specified to the **ex file** command, the current pathname shall be set  
13716 to the *file* argument, and the alternate pathname shall be set to the previous value of  
13717 the current pathname.
- 13718           4. If a *file* argument is specified to the **ex read** and **write** commands (that is, when  
13719 reading or writing a file, and not to the program named by the **shell** edit option), or a  
13720 *file* argument is specified to the **ex xit** command:
- 13721                   a. If the current pathname has no value, the current pathname shall be set to the *file*  
13722 argument.
- 13723                   b. Otherwise, the alternate pathname shall be set to the *file* argument.

13724           If the alternate pathname is set to the previous value of the current pathname when the  
13725 current pathname had no previous value, then the alternate pathname shall have no value  
13726 as a result.

13727           *current line*

13728           The line of the edit buffer referenced by the cursor. Each command description specifies the  
13729 current line after the command has been executed, as the *current line value*. When the edit  
13730 buffer contains no lines, the current line shall be zero; see **Addressing in ex** (on page 2562).

13731           *current column*

13732           The current display line column occupied by the cursor. (The columns shall be numbered  
13733 beginning at 1.) Each command description specifies the current column after the command  
13734 has been executed, as the *current column value*. This column is an *ideal* column that is  
13735 remembered over the lifetime of the editor. The actual display line column upon which the  
13736 cursor rests may be different from the current column; see the cursor positioning discussion  
13737 in **Command Descriptions in vi** (on page 3186).

13738           *set to non-<blank>*

13739           A description for a current column value, meaning that the current column shall be set to  
13740 the last display line column on which is displayed any part of the first non-<blank> of the  
13741 line. If the line has no non-<blank> non- <newline>s, the current column shall be set to the  
13742 last display line column on which is displayed any part of the last non-<newline> in the  
13743 line. If the line is empty, the current column shall be set to column position 1.

13744           The length of lines in the edit buffer may be limited to {LINE\_MAX} bytes. In open and visual  
13745 mode, the length of lines in the edit buffer may be limited to the number of characters that will  
13746 fit in the display. If either limit is exceeded during editing, an error message shall be written. If  
13747 either limit is exceeded by a line read in from a file, an error message shall be written and the  
13748 edit session may be terminated.

13749           If the editor stops running due to any reason other than a user command, and the edit buffer has  
13750 been modified since the last complete write, it shall be equivalent to a SIGHUP asynchronous  
13751 event. If the system crashes, it shall be equivalent to a SIGHUP asynchronous event.

13752 During initialization (before the first file is copied into the edit buffer or any user commands  
13753 from the terminal are processed) the following shall occur:

- 13754 1. If the environment variable *EXINIT* is set, the editor shall execute the *ex* commands  
13755 contained in that variable.
- 13756 2. If the *EXINIT* variable is not set, and all of the following are true:
  - 13757 a. The *HOME* environment variable is not null and not empty.
  - 13758 b. The file *.exrc* in the directory referred to by the *HOME* environment variable:
    - 13759 1. Exists
    - 13760 2. Is owned by the same user ID as the real user ID of the process or the process  
13761 has appropriate privileges
    - 13762 3. Is not writeable by anyone other than the owner

13763 the editor shall execute the *ex* commands contained in that file.

- 13764 3. If and only if all the following are true:
  - 13765 a. The current directory is not referred to by the *HOME* environment variable.
  - 13766 b. A command in the *EXINIT* environment variable or a command in the *.exrc* file in the  
13767 directory referred to by the *HOME* environment variable sets the editor option *exrc*.
  - 13768 c. The *.exrc* file in the current directory:
    - 13769 1. Exists
    - 13770 2. Is owned by the same user ID as the real user ID of the process, or by one of a  
13771 set of implementation-defined user IDs
    - 13772 3. Is not writeable by anyone other than the owner

13773 the editor shall attempt to execute the *ex* commands contained in that file.

13774 Lines in any *.exrc* file that are blank lines shall be ignored. If any *.exrc* file exists, but is not read  
13775 for ownership or permission reasons, it shall be an error.

13776 After the *EXINIT* variable and any *.exrc* files are processed, the first file specified by the user  
13777 shall be edited, as follows:

- 13778 1. If the user specified the *-t* option, the effect shall be as if the *ex tag* command was entered  
13779 with the specified argument, with the exception that if tag processing does not result in a  
13780 file to edit, the effect shall be as described in step 3. below.
- 13781 2. Otherwise, if the user specified any command line *file* arguments, the effect shall be as if  
13782 the *ex edit* command was entered with the first of those arguments as its *file* argument.
- 13783 3. Otherwise, the effect shall be as if the *ex edit* command was entered with a nonexistent  
13784 filename as its *file* argument. It is unspecified whether this action shall set the current  
13785 pathname. In an implementation where this action does not set the current pathname, any  
13786 editor command using the current pathname shall fail until an editor command sets the  
13787 current pathname.

13788 If the *-r* option was specified, the first time a file in the initial argument list or a file specified by  
13789 the *-t* option is edited, if recovery information has previously been saved about it, that  
13790 information shall be recovered and the editor shall behave as if the contents of the edit buffer  
13791 have already been modified. If there are multiple instances of the file to be recovered, the one  
13792 most recently saved shall be recovered, and an informational message that there are previous

13793 versions of the file that can be recovered shall be written. If no recovery information about a file  
 13794 is available, an informational message to this effect shall be written, and the edit shall proceed as  
 13795 usual.

13796 If the `-c` option was specified, the first time a file that already exists (including a file that might  
 13797 not exist but for which recovery information is available, when the `-r` option is specified)  
 13798 replaces or initializes the contents of the edit buffer, the current line shall be set to the last line of  
 13799 the edit buffer, the current column shall be set to non-`<blank>`, and the `ex` commands specified  
 13800 with the `-c` option shall be executed. In this case, the current line and current column shall not be  
 13801 set as described for the command associated with the replacement or initialization of the edit  
 13802 buffer contents. However, if the `-t` option or a **tag** command is associated with this action, the `-c`  
 13803 option commands shall be executed and then the movement to the tag shall be performed.

13804 The current argument list shall initially be set to the filenames specified by the user on the  
 13805 command line. If no filenames are specified by the user, the current argument list shall be empty.  
 13806 If the `-t` option was specified, it is unspecified whether any filename resulting from tag  
 13807 processing shall be prepended to the current argument list. In the case where the filename is  
 13808 added as a prefix to the current argument list, the current argument list reference shall be set to  
 13809 that filename. In the case where the filename is not added as a prefix to the current argument  
 13810 list, the current argument list reference shall logically be located before the first of the filenames  
 13811 specified on the command line (for example, a subsequent `ex next` command shall edit the first  
 13812 filename from the command line). If the `-t` option was not specified, the current argument list  
 13813 reference shall be to the first of the filenames on the command line.

#### 13814 Addressing in ex

13815 Addressing in `ex` relates to the current line and the current column; the address of a line is its 1-  
 13816 based line number, the address of a column is its 1-based count from the beginning of the line.  
 13817 Generally, the current line is the last line affected by a command. The current line number is the  
 13818 address of the current line. In each command description, the effect of the command on the  
 13819 current line number and the current column is described.

13820 Addresses are constructed as follows:

- 13821 1. The character `'.'` (period) shall address the current line.
- 13822 2. The character `'$'` shall address the last line of the edit buffer.
- 13823 3. The positive decimal number *n* shall address the *n*th line of the edit buffer.
- 13824 4. The address `"'x"` refers to the line marked with the mark name character `'x'`, which shall  
 13825 be a lowercase letter from the portable character set or one of the characters `'\'` or `'\'`. It  
 13826 shall be an error if the line that was marked is not currently present in the edit buffer or the  
 13827 mark has not been set. Lines can be marked with the `ex mark` or `k` commands, or the `vi m`  
 13828 command.
- 13829 5. A regular expression (RE) enclosed by slashes (`'/'`) shall address the first line found by  
 13830 searching forwards from the line following the current line toward the end of the edit  
 13831 buffer and stopping at the first line for which the line excluding the terminating `<newline>`  
 13832 matches the regular expression. As stated in **Regular Expressions in ex** (on page 2592), an  
 13833 address consisting of a null regular expression delimited by slashes `"/"` shall address the  
 13834 next line for which the line excluding the terminating `<newline>` matches the last regular  
 13835 expression encountered. In addition, the second slash can be omitted at the end of a  
 13836 command line. If the **wrapscan** edit option is set, the search shall wrap around to the  
 13837 beginning of the edit buffer and continue up to and including the current line, so that the  
 13838 entire edit buffer is searched. Within the regular expression, the sequence `"\"` shall  
 13839 represent a literal slash instead of the regular expression delimiter.

13840 6. A regular expression enclosed in question marks ('?') shall address the first line found by  
 13841 searching backwards from the line preceding the current line toward the beginning of the  
 13842 edit buffer and stopping at the first line for which the line excluding the terminating  
 13843 <newline> matches the regular expression. An address consisting of a null regular  
 13844 expression delimited by question marks "??" shall address the previous line for which the  
 13845 line excluding the terminating <newline> matches the last regular expression encountered.  
 13846 In addition, the second question mark can be omitted at the end of a command line. If the  
 13847 **wrapsan** edit option is set, the search shall wrap around from the beginning of the edit  
 13848 buffer to the end of the edit buffer and continue up to and including the current line, so  
 13849 that the entire edit buffer is searched. Within the regular expression, the sequence "\?"  
 13850 shall represent a literal question mark instead of the RE delimiter.

13851 7. A plus sign ('+') or a minus sign ('-') followed by a decimal number shall address the  
 13852 current line plus or minus the number. A '+' or '-' not followed by a decimal number  
 13853 shall address the current line plus or minus 1.

13854 Addresses can be followed by zero or more address offsets, optionally <blank>-separated.  
 13855 Address offsets are constructed as follows:

13856 1. A '+' or '-' immediately followed by a decimal number shall add (subtract) the  
 13857 indicated number of lines to (from) the address. A '+' or '-' not followed by a decimal  
 13858 number shall add (subtract) 1 to (from) the address.

13859 2. A decimal number shall add the indicated number of lines to the address.

13860 It shall not be an error for an intermediate address value to be less than zero or greater than the  
 13861 last line in the edit buffer. It shall be an error for the final address value to be less than zero or  
 13862 greater than the last line in the edit buffer.

13863 Commands take zero, one, or two addresses; see the descriptions of *1addr* and *2addr* in  
 13864 **Command Descriptions in ex** (on page 2569). If more than the required number of addresses  
 13865 are provided to a command that requires zero addresses, it shall be an error. Otherwise, if more  
 13866 than the required number of addresses are provided to a command, the addresses specified first  
 13867 shall be evaluated and then discarded until the maximum number of valid addresses remain.

13868 Addresses shall be separated from each other by a comma (',') or a semicolon (;). If no  
 13869 address is specified before or after a comma or semicolon separator, it shall be as if the address  
 13870 of the current line was specified before or after the separator. In the case of a semicolon  
 13871 separator, the current line ('.') shall be set to the first address, and only then will the next  
 13872 address be calculated. This feature can be used to determine the starting line for forwards and  
 13873 backwards searches (see rules 5. and 6.).

13874 A percent sign ('%') shall be equivalent to entering the two addresses "1, \$".

13875 Any delimiting <blank>s between addresses, address separators, or address offsets shall be  
 13876 discarded.

### 13877 **Command Line Parsing in ex**

13878 The following symbol is used in this and following sections to describe parsing behavior:

13879 *escape* If a character is referred to as "backslash escaped" or "<control>-V escaped," it  
 13880 shall mean that the character acquired or lost a special meaning by virtue of being  
 13881 preceded, respectively, by a backslash or <control>-V character. Unless otherwise  
 13882 specified, the escaping character shall be discarded at that time and shall not be  
 13883 further considered for any purpose.

- 13884 Command-line parsing shall be done in the following steps. For each step, characters already  
 13885 evaluated shall be ignored; that is, the phrase "leading character" refers to the next character  
 13886 that has not yet been evaluated.
- 13887 1. Leading colon characters shall be skipped.
  - 13888 2. Leading <blank>s shall be skipped.
  - 13889 3. If the leading character is a double-quote character, the characters up to and including the  
 13890 next non-backslash-escaped <newline> shall be discarded, and any subsequent characters  
 13891 shall be parsed as a separate command.
  - 13892 4. Leading characters that can be interpreted as addresses shall be evaluated; see **Addressing**  
 13893 **in ex** (on page 2562).
  - 13894 5. Leading <blank>s shall be skipped.
  - 13895 6. If the next character is a vertical-line character or a <newline>:  
 13896 a. If the next character is a <newline>:  
 13897 1. If *ex* is in open or visual mode, the current line shall be set to the last address  
 13898 specified, if any.  
 13899 2. Otherwise, if the last command was terminated by a vertical-line character, no  
 13900 action shall be taken; for example, the command "||<newline>" shall  
 13901 execute two implied commands, not three.  
 13902 3. Otherwise, step 6.b. shall apply.  
 13903 b. Otherwise, the implied command shall be the **print** command. The last #, **p**, and **l**  
 13904 flags specified to any *ex* command shall be remembered and shall apply to this  
 13905 implied command. Executing the *ex* **number**, **print**, or **list** command shall set the  
 13906 remembered flags to #, nothing, and **l**, respectively, plus any other flags specified for  
 13907 that execution of the **number**, **print**, or **list** command.  
 13908 If *ex* is not currently performing a **global** or **v** command, and no address or count is  
 13909 specified, the current line shall be incremented by 1 before the command is executed.  
 13910 If incrementing the current line would result in an address past the last line in the  
 13911 edit buffer, the command shall fail, and the increment shall not happen.  
 13912 c. The <newline> or vertical-line character shall be discarded and any subsequent  
 13913 characters shall be parsed as a separate command.
  - 13914 7. The command name shall be comprised of the next character (if the character is not  
 13915 alphabetic), or the next character and any subsequent alphabetic characters (if the  
 13916 character is alphabetic), with the following exceptions:  
 13917 a. Commands that consist of any prefix of the characters in the command name **delete**,  
 13918 followed immediately by any of the characters 'l', 'p', '+', '-', or '#' shall be  
 13919 interpreted as a **delete** command, followed by a <blank>, followed by the characters  
 13920 that were not part of the prefix of the **delete** command. The maximum number of  
 13921 characters shall be matched to the command name **delete**; for example, "de1" shall  
 13922 not be treated as "de" followed by the flag **l**.  
 13923 b. Commands that consist of the character 'k', followed by a character that can be  
 13924 used as the name of a mark, shall be equivalent to the mark command followed by a  
 13925 <blank>, followed by the character that followed the 'k'.  
 13926 c. Commands that consist of the character 's', followed by characters that could be  
 13927 interpreted as valid options to the **s** command, shall be the equivalent of the **s**

13928 command, without any pattern or replacement values, followed by a <blank>,  
13929 followed by the characters after the 's'.

13930 8. The command name shall be matched against the possible command names, and a  
13931 command name that contains a prefix matching the characters specified by the user shall  
13932 be the executed command. In the case of commands where the characters specified by the  
13933 user could be ambiguous, the executed command shall be as follows:

|       |           |               |           |              |           |              |
|-------|-----------|---------------|-----------|--------------|-----------|--------------|
| 13934 | <b>a</b>  | <b>append</b> | <b>n</b>  | <b>next</b>  | <b>t</b>  | <b>t</b>     |
| 13935 | <b>c</b>  | <b>change</b> | <b>p</b>  | <b>print</b> | <b>u</b>  | <b>undo</b>  |
| 13936 | <b>ch</b> | <b>change</b> | <b>pr</b> | <b>print</b> | <b>un</b> | <b>undo</b>  |
| 13937 | <b>e</b>  | <b>edit</b>   | <b>r</b>  | <b>read</b>  | <b>v</b>  | <b>v</b>     |
| 13938 | <b>m</b>  | <b>move</b>   | <b>re</b> | <b>read</b>  | <b>w</b>  | <b>write</b> |
| 13939 | <b>ma</b> | <b>mark</b>   | <b>s</b>  | <b>s</b>     |           |              |

13940 Implementation extensions with names causing similar ambiguities shall not be checked  
13941 for a match until all possible matches for commands specified by IEEE Std 1003.1-200x  
13942 have been checked.

13943 9. If the command is a **!** command, or if the command is a **read** command followed by zero  
13944 or more <blank>s and a **!**, or if the command is a **write** command followed by one or more  
13945 <blank>s and a **!**, the rest of the command shall include all characters up to a non-  
13946 backslash-escaped <newline>. The <newline> shall be discarded and any subsequent  
13947 characters shall be parsed as a separate *ex* command.

13948 10. Otherwise, if the command is an **edit**, **ex**, or **next** command, or a **visual** command while in  
13949 open or visual mode, the next part of the command shall be parsed as follows:

13950 a. Any '**!**' character immediately following the command shall be skipped and be part  
13951 of the command.

13952 b. Any leading <blank>s shall be skipped and be part of the command.

13953 c. If the next character is a '+', characters up to the first non-backslash-escaped  
13954 <newline> or non-backslash-escaped <blank> shall be skipped and be part of the  
13955 command.

13956 d. The rest of the command shall be determined by the steps specified in paragraph 12.

13957 11. Otherwise, if the command is a **global**, **open**, **s**, or **v** command, the next part of the  
13958 command shall be parsed as follows:

13959 a. Any leading <blank>s shall be skipped and be part of the command.

13960 b. If the next character is not an alphanumeric, double-quote, <newline>, backslash, or  
13961 vertical-line character:

13962 1. The next character shall be used as a command delimiter.

13963 2. If the command is a **global**, **open**, or **v** command, characters up to the first  
13964 non-backslash-escaped <newline>, or first non-backslash-escaped delimiter  
13965 character, shall be skipped and be part of the command.

13966 3. If the command is an **s** command, characters up to the first non-backslash-  
13967 escaped <newline>, or second non-backslash-escaped delimiter character, shall  
13968 be skipped and be part of the command.

13969 c. If the command is a **global** or **v** command, characters up to the first non-backslash-  
13970 escaped <newline> shall be skipped and be part of the command.

- 13971 d. Otherwise, the rest of the command shall be determined by the steps specified in  
13972 paragraph 12.
- 13973 12. Otherwise:
- 13974 a. If the command was a **map**, **unmap**, **abbreviate**, or **unabbreviate** command,  
13975 characters up to the first non-<control>-V-escaped <newline>, vertical-line, or  
13976 double-quote character shall be skipped and be part of the command.
- 13977 b. Otherwise, characters up to the first non-backslash-escaped <newline>, vertical-line,  
13978 or double-quote character shall be skipped and be part of the command.
- 13979 c. If the command was an **append**, **change**, or **insert** command, and the step 12.b.  
13980 ended at a vertical-line character, any subsequent characters, up to the next non-  
13981 backslash-escaped <newline> shall be used as input text to the command.
- 13982 d. If the command was ended by a double-quote character, all subsequent characters,  
13983 up to the next non-backslash-escaped <newline>, shall be discarded.
- 13984 e. The terminating <newline> or vertical-line character shall be discarded and any  
13985 subsequent characters shall be parsed as a separate *ex* command.

13986 Command arguments shall be parsed as described by the Synopsis and Description of each  
13987 individual *ex* command. This parsing shall not be <blank>-sensitive, except for the **!** argument,  
13988 which must follow the command name without intervening <blank>s, and where it would  
13989 otherwise be ambiguous. For example, *count* and *flag* arguments need not be <blank>-separated  
13990 because "*d22p*" is not ambiguous, but *file* arguments to the *ex next* command must be  
13991 separated by one or more <blank>s. Any <blank> in command arguments for the **abbreviate**,  
13992 **unabbreviate**, **map**, and **unmap** commands can be <control>-V-escaped, in which case the  
13993 <blank> shall not be used as an argument delimiter. Any <blank> in the command argument for  
13994 any other command can be backslash-escaped, in which case that <blank> shall not be used as  
13995 an argument delimiter.

13996 Within command arguments for the **abbreviate**, **unabbreviate**, **map**, and **unmap** commands,  
13997 any character can be <control>-V-escaped. All such escaped characters shall be treated literally  
13998 and shall have no special meaning. Within command arguments for all other *ex* commands that  
13999 are not regular expressions or replacement strings, any character that would otherwise have a  
14000 special meaning can be backslash-escaped. Escaped characters shall be treated literally, without  
14001 special meaning as shell expansion characters or **'!'**, **'%'**, and **'#'** expansion characters. See  
14002 **Regular Expressions in ex** (on page 2592) and **Replacement Strings in ex** (on page 2592) for  
14003 descriptions of command arguments that are regular expressions or replacement strings.

14004 Non-backslash-escaped **'%'** characters appearing in *file* arguments to any *ex* command shall be  
14005 replaced by the current pathname; unescaped **'#'** characters shall be replaced by the alternate  
14006 pathname. It shall be an error if **'%'** or **'#'** characters appear unescaped in an argument and  
14007 their corresponding values are not set.

14008 Non-backslash-escaped **'!'** characters in the arguments to either the *ex!* command or the open  
14009 and visual mode **!** command, or in the arguments to the *ex read* command, where the first non-  
14010 <blank> after the command name is a **'!'** character, or in the arguments to the *ex write*  
14011 command where the command name is followed by one or more <blank>s and the first non-  
14012 <blank> after the command name is a **'!'** character, shall be replaced with the arguments to the  
14013 last of those three commands as they appeared after all unescaped **'%'**, **'#'**, and **'!'** characters  
14014 were replaced. It shall be an error if **'!'** characters appear unescaped in one of these commands  
14015 and there has been no previous execution of one of these commands.

14016 If an error occurs during the parsing or execution of an *ex* command:

- 14017 • An informational message to this effect shall be written. Execution of the **ex** command shall
- 14018 stop, and the cursor (for example, the current line and column) shall not be further modified.
- 14019 • If the **ex** command resulted from a map expansion, all characters from that map expansion
- 14020 shall be discarded, except as otherwise specified by the **map** command.
- 14021 • Otherwise, if the **ex** command resulted from the processing of an *EXINIT* environment
- 14022 variable, a **.exrc** file, a **:source** command, a **-c** option, or a **+command** specified to an **ex edit**,
- 14023 **ex**, **next**, or **visual** command, no further commands from the source of the commands shall
- 14024 be executed.
- 14025 • Otherwise, if the **ex** command resulted from the execution of a buffer or a **global** or **v**
- 14026 command, no further commands caused by the execution of the buffer or the **global** or **v**
- 14027 command shall be executed.
- 14028 • Otherwise, if the **ex** command was not terminated by a <newline>, all characters up to and
- 14029 including the next non-backslash-escaped <newline> shall be discarded.

### 14030 **Input Editing in ex**

14031 The following symbols are used in this and following sections to specify command actions.

14032 *word* In the POSIX locale, a word consists of a maximal sequence of letters, digits, and  
 14033 underscores, delimited at both ends by characters other than letters, digits, or  
 14034 underscores, or by the beginning or end of a line or the edit buffer.

14035 When accepting input characters from the user, in either **ex** command mode or **ex** text input  
 14036 mode, **ex** shall enable canonical mode input processing, as defined in the System Interfaces  
 14037 volume of IEEE Std 1003.1-200x.

14038 If in **ex** text input mode:

- 14039 1. If the **number** edit option is set, **ex** shall prompt for input using the line number that would
- 14040 be assigned to the line if it is entered, in the format specified for the **ex number** command.
- 14041 2. If the **autoindent** edit option is set, **ex** shall prompt for input using **autoindent** characters,
- 14042 as described by the **autoindent** edit option. **autoindent** characters shall follow the line
- 14043 number, if any.

14044 If in **ex** command mode:

- 14045 1. If the **prompt** edit option is set, input shall be prompted for using a single ' : ' character;
- 14046 otherwise, there shall be no prompt.

14047 The input characters in the following sections shall have the following effects on the input line.

### 14048 **Scroll**

14049 *Synopsis:* eof

14050 See the description of the *stty eof* character in *stty*.

14051 If in **ex** command mode:

14052 If the *eof* character is the first character entered on the line, the line shall be evaluated as if it |  
 14053 contained two characters: a <control>-D and a <newline>. |

14054 Otherwise, the *eof* character shall have no special meaning. |

14055 If in **ex** text input mode:

14056 If the cursor follows an **autoindent** character, the **autoindent** characters in the line shall be |  
 14057 modified so that a part of the next text input character will be displayed on the first column |  
 14058 in the line after the previous **shiftwidth** edit option column boundary, and the user shall be |  
 14059 prompted again for input for the same line. |

14060 Otherwise, if the cursor follows a '0', which follows an **autoindent** character, and the '0' |  
 14061 was the previous text input character, the '0' and all **autoindent** characters in the line shall |  
 14062 be discarded, and the user shall be prompted again for input for the same line. |

14063 Otherwise, if the cursor follows a '^', which follows an **autoindent** character, and the '^' |  
 14064 was the previous text input character, the '^' and all **autoindent** characters in the line shall |  
 14065 be discarded, and the user shall be prompted again for input for the same line. In addition, |  
 14066 the **autoindent** level for the next input line shall be derived from the same line from which |  
 14067 the **autoindent** level for the current input line was derived. |

14068 Otherwise, if there are no **autoindent** or text input characters in the line, the *eof* character |  
 14069 shall be discarded. |

14070 Otherwise, the *eof* character shall have no special meaning. |

14071 **<newline>**

14072 *Synopsis:* <newline>  
 14073 <control>-J

14074 If in *ex* command mode:

14075 Cause the command line to be parsed; <control>-J shall be mapped to the <newline> for this |  
 14076 purpose. |

14077 If in *ex* text input mode:

14078 Terminate the current line. If there are no characters other than **autoindent** characters on the |  
 14079 line, all characters on the line shall be discarded. |

14080 Prompt for text input on a new line after the current line. If the **autoindent** edit option is set, |  
 14081 an appropriate number of **autoindent** characters shall be added as a prefix to the line as |  
 14082 described by the *ex* **autoindent** edit option. |

14083 **<backslash>**

14084 *Synopsis:* <backslash>

14085 Allow the entry of a subsequent <newline> or <control>-J as a literal character, removing any |  
 14086 special meaning that it may have to the editor during text input mode. The backslash character |  
 14087 shall be retained and evaluated when the command line is parsed, or retained and included |  
 14088 when the input text becomes part of the edit buffer.

14089 **<control>-V**

14090 *Synopsis:* <control>-V

14091 Allow the entry of any subsequent character as a literal character, removing any special meaning |  
 14092 that it may have to the editor during text input mode. The <control>-V character shall be |  
 14093 discarded before the command line is parsed or the input text becomes part of the edit buffer.

14094 If the “literal next” functionality is performed by the underlying system, it is implementation- |  
 14095 defined whether a character other than <control>-V performs this function. |

14096 <control>-W

14097 *Synopsis:* <control>-W

14098 Discard the <control>-W, and the word previous to it in the input line, including any <blank>s  
 14099 following the word and preceding the <control>-W. If the “word erase” functionality is  
 14100 performed by the underlying system, it is implementation-defined whether a character other  
 14101 than <control>-W performs this function.

14102 **Command Descriptions in ex**

14103 The following symbols are used in this section to represent command modifiers. Some of these  
 14104 modifiers can be omitted, in which case the specified defaults shall be used.

14105 *laddr* A single line address, given in any of the forms described in **Addressing in ex** (on  
 14106 page 2562); the default shall be the current line (‘.’), unless otherwise specified.

14107 If the line address is zero, it shall be an error, unless otherwise specified in the  
 14108 following command descriptions.

14109 If the edit buffer is empty, and the address is specified with a command other than  
 14110 =, **append**, **insert**, **open**, **put**, **read**, or **visual**, or the address is not zero, it shall be  
 14111 an error.

14112 *2addr* Two addresses specifying an inclusive range of lines. If no addresses are specified,  
 14113 the default for *2addr* shall be the current line only (“.”), unless otherwise  
 14114 specified in the following command descriptions. If one address is specified, *2addr*  
 14115 shall specify that line only, unless otherwise specified in the following command  
 14116 descriptions.

14117 It shall be an error if the first address is greater than the second address.

14118 If the edit buffer is empty, and the two addresses are specified with a command  
 14119 other than the **!**, **write**, **wq**, or **xit** commands, or either address is not zero, it shall  
 14120 be an error.

14121 *count* A positive decimal number. If *count* is specified, it shall be equivalent to specifying  
 14122 an additional address to the command, unless otherwise specified by the following  
 14123 command descriptions. The additional address shall be equal to the last address  
 14124 specified to the command (either explicitly or by default) plus *count*−1.

14125 If this would result in an address greater than the last line of the edit buffer, it shall  
 14126 be corrected to equal the last line of the edit buffer.

14127 *flags* One or more of the characters ‘+’, ‘-’, ‘#’, ‘p’, or ‘l’ (ell). The flag characters  
 14128 can be <blank>-separated, and in any order or combination. The characters ‘#’,  
 14129 ‘p’, and ‘l’ shall cause lines to be written in the format specified by the **print**  
 14130 command with the specified *flags*.

14131 The lines to be written are as follows:

- 14132 1. All edit buffer lines written during the execution of the **ex** **&**, **~**, **list**, **number**,  
 14133 **open**, **print**, **s**, **visual**, and **z** commands shall be written as specified by *flags*.
- 14134 2. After the completion of an **ex** command with a flag as an argument, the  
 14135 current line shall be written as specified by *flags*, unless the current line was  
 14136 the last line written by the command.

14137 The characters ‘+’ and ‘-’ cause the value of the current line after the execution  
 14138 of the **ex** command to be adjusted by the offset address as described in **Addressing**

14139                   **in ex** (on page 2562). This adjustment shall occur before the current line is written  
 14140 as described in 2. above.

14141                   The default for *flags* shall be none.

14142           *buffer*           One of a number of named areas for holding text. The named buffers are specified  
 14143 by the alphanumeric characters of the POSIX locale. There shall also be one  
 14144 “unnamed” buffer. When no buffer is specified for editor commands that use a  
 14145 buffer, the unnamed buffer shall be used. Commands that store text into buffers  
 14146 shall store the text as it was before the command took effect, and shall store text  
 14147 occurring earlier in the file before text occurring later in the file, regardless of how  
 14148 the text region was specified. Commands that store text into buffers shall store the  
 14149 text into the unnamed buffer as well as any specified buffer.

14150                   In *ex* commands, buffer names are specified as the name by itself. In open or visual  
 14151 mode commands the name is preceded by a double quote ( ' " ' ) character.

14152                   If the specified buffer name is an uppercase character, and the buffer contents are  
 14153 to be modified, the buffer shall be appended to rather than being overwritten. If  
 14154 the buffer is not being modified, specifying the buffer name in lowercase and  
 14155 uppercase shall have identical results.

14156                   There shall also be buffers named by the numbers 1 through 9. In open and visual  
 14157 mode, if a region of text including characters from more than a single line is being  
 14158 modified by the *vi c* or *d* commands, the motion character associated with the *c* or  
 14159 *d* commands specifies that the buffer text shall be in line mode, or the commands  
 14160 %, /, ?, (, ), N, n, {, or } are used to define a region of text for the *c* or *d* commands,  
 14161 the contents of buffers 1 through 8 shall be moved into the buffer named by the  
 14162 next numerically greater value, the contents of buffer 9 shall be discarded, and the  
 14163 region of text shall be copied into buffer 1. This shall be in addition to copying the  
 14164 text into a user-specified buffer or unnamed buffer, or both. Numeric buffers can  
 14165 be specified as a source buffer for open and visual mode commands; however,  
 14166 specifying a numeric buffer as the write target of an open or visual mode  
 14167 command shall have unspecified results.

14168                   The text of each buffer shall have the characteristic of being in either line or  
 14169 character mode. Appending text to a non-empty buffer shall set the mode to match  
 14170 the characteristic of the text being appended. Appending text to a buffer shall  
 14171 cause the creation of at least one additional line in the buffer. All text stored into  
 14172 buffers by *ex* commands shall be in line mode. The *ex* commands that use buffers  
 14173 as the source of text specify individually how buffers of different modes are  
 14174 handled. Each open or visual mode command that uses buffers for any purpose  
 14175 specifies individually the mode of the text stored into the buffer and how buffers  
 14176 of different modes are handled.

14177           *file*           Command text used to derive a pathname. The default shall be the current  
 14178 pathname, as defined previously, in which case, if no current pathname has yet  
 14179 been established it shall be an error, except where specifically noted in the  
 14180 individual command descriptions that follow. If the command text contains any of  
 14181 the characters '~', '{', '[', '\*', '?', '\$', '\', ' ', '\ ', and '\ ', it shall be  
 14182 subjected to the process of “shell expansions”, as described below; if more than a  
 14183 single pathname results and the command expects only one, it shall be an error.

14184                   The process of shell expansions in the editor shall be done as follows. The *ex* utility  
 14185 shall pass two arguments to the program named by the shell edit option; the first  
 14186 shall be *-c*, and the second shall be the string "echo" and the command text as a

14187 single argument. The standard output and standard error of that command shall  
14188 replace the command text.

14189 **!** A character that can be appended to the command name to modify its operation,  
14190 as detailed in the individual command descriptions. With the exception of the *ex*  
14191 **read**, **write**, and **!** commands, the '**!**' character shall only act as a modifier if there  
14192 are no <blank>s between it and the command name.

14193 *remembered search direction*

14194 The *vi* commands **N** and **n** begin searching in a forwards or backwards direction in  
14195 the edit buffer based on a remembered search direction, which is initially unset,  
14196 and is set by the *ex* **global**, **v**, **s**, and **tag** commands, and the *vi* / and **?** commands.

14197 **Abbreviate**

14198 *Synopsis:* `ab[breviate][lhs rhs]`

14199 If *lhs* and *rhs* are not specified, write the current list of abbreviations and do nothing more.

14200 Implementations may restrict the set of characters accepted in *lhs* or *rh*, except that printable  
14201 characters and <blank>s shall not be restricted. Additional restrictions shall be implementation-  
14202 defined.

14203 In both *lhs* and *rhs*, any character may be escaped with a <control>-V, in which case the  
14204 character shall not be used to delimit *lhs* from *rhs*, and the escaping <control>-V shall be  
14205 discarded.

14206 In open and visual text input mode, if a non-word or <ESC> character that is not escaped by a  
14207 <control>-V character is entered after a word character, a check shall be made for a set of  
14208 characters matching *lhs*, in the text input entered during this command. If it is found, the effect  
14209 shall be as if *rhs* was entered instead of *lhs*.

14210 The set of characters that are checked is defined as follows:

- 14211 1. If there are no characters inserted before the word and non-word or <ESC> characters that  
14212 triggered the check, the set of characters shall consist of the word character.
- 14213 2. If the character inserted before the word and non-word or <ESC> characters that triggered  
14214 the check is a word character, the set of characters shall consist of the characters inserted  
14215 immediately before the triggering characters that are word characters, plus the triggering  
14216 word character.
- 14217 3. If the character inserted before the word and non-word or <ESC> characters that triggered  
14218 the check is not a word character, the set of characters shall consist of the characters that  
14219 were inserted before the triggering characters that are neither <blank>s nor word  
14220 characters, plus the triggering word character.

14221 It is unspecified whether the *lhs* argument entered for the *ex* **abbreviate** and **unabbreviate**  
14222 commands is replaced in this fashion. Regardless of whether or not the replacement occurs, the  
14223 effect of the command shall be as if the replacement had not occurred.

14224 *Current line:* Unchanged.

14225 *Current column:* Unchanged.

14226 **Append**14227 *Synopsis:* `[1addr] a[ppend][!]`14228 Enter text input mode; the input text shall be placed after the specified line. If line zero is  
14229 specified, the text shall be placed at the beginning of the edit buffer.14230 This command shall be affected by the **number** and **autoindent** edit options; following the  
14231 command name with '!' shall cause the **autoindent** edit option setting to be toggled for the  
14232 duration of this command only.14233 *Current line:* Set to the last input line; if no lines were input, set to the specified line, or to the  
14234 first line of the edit buffer if a line of zero was specified, or zero if the edit buffer is empty.14235 *Current column:* Set to non-<blank>.14236 **Arguments**14237 *Synopsis:* `ar[gs]`14238 Write the current argument list, with the current argument-list entry, if any, between '[' and  
14239 ']' characters.14240 *Current line:* Unchanged.14241 *Current column:* Unchanged.14242 **Change**14243 *Synopsis:* `[2addr] c[hange][!][count]`14244 Enter *ex* text input mode; the input text shall replace the specified lines. The specified lines shall  
14245 be copied into the unnamed buffer, which shall become a line mode buffer.14246 This command shall be affected by the **number** and **autoindent** edit options; following the  
14247 command name with '!' shall cause the **autoindent** edit option setting to be toggled for the  
14248 duration of this command only.14249 *Current line:* Set to the last input line; if no lines were input, set to the line before the first  
14250 address, or to the first line of the edit buffer if there are no lines preceding the first address, or to  
14251 zero if the edit buffer is empty.14252 *Current column:* Set to non-<blank>.14253 **Change Directory**14254 *Synopsis:* `chd[ir][!][directory]`14255 `cd[!][directory]`14256 Change the current working directory to *directory*.14257 If no *directory* argument is specified, and the *HOME* environment variable is set to a non-null  
14258 and non-empty value, *directory* shall default to the value named in the *HOME* environment  
14259 variable. If the *HOME* environment variable is empty or is undefined, the default value of  
14260 *directory* is implementation-defined.14261 If no '!' is appended to the command name, and the edit buffer has been modified since the  
14262 last complete write, and the current pathname does not begin with a '/', it shall be an error.14263 *Current line:* Unchanged.

14264 *Current column*: Unchanged.

### 14265 **Copy**

14266 *Synopsis:*    [2addr] co[py] laddr [flags]  
14267            [2addr] t laddr [flags]

14268 Copy the specified lines after the specified destination line; line zero specifies that the lines shall  
14269 be placed at the beginning of the edit buffer.

14270 *Current line*: Set to the last line copied.

14271 *Current column*: Set to non-<blank>.

### 14272 **Delete**

14273 *Synopsis:*    [2addr] d[ele]te[buffer][count][flags]

14274 Delete the specified lines into a buffer (defaulting to the unnamed buffer), which shall become a  
14275 line-mode buffer.

14276 Flags can immediately follow the command name; see **Command Line Parsing in ex** (on page  
14277 2563).

14278 *Current line*: Set to the line following the deleted lines, or to the last line in the edit buffer if that  
14279 line is past the end of the edit buffer, or to zero if the edit buffer is empty.

14280 *Current column*: Set to non-<blank>.

### 14281 **Edit**

14282 *Synopsis:*    e[dit][!][+command][file]  
14283            ex[!][+command][file]

14284 If no '!' is appended to the command name, and the edit buffer has been modified since the  
14285 last complete write, it shall be an error.

14286 If *file* is specified, replace the current contents of the edit buffer with the current contents of *file*,  
14287 and set the current pathname to *file*. If *file* is not specified, replace the current contents of the  
14288 edit buffer with the current contents of the file named by the current pathname. If for any reason  
14289 the current contents of the file cannot be accessed, the edit buffer shall be empty.

14290 The *+command* option shall be <blank>-delimited; <blank>s within *+command* can be escaped by  
14291 preceding them with a backslash character. The *+command* shall be interpreted as an *ex*  
14292 command immediately after the contents of the edit buffer have been replaced and the current  
14293 line and column have been set.

14294 If the edit buffer is empty:

14295 *Current line*: Set to 0.

14296 *Current column*: Set to 1.

14297 Otherwise, if executed while in *ex* command mode or if the *+command* argument is specified:

14298 *Current line*: Set to the last line of the edit buffer.

14299 *Current column*: Set to non-<blank>.

14300 Otherwise, if *file* is omitted or results in the current pathname:

14301 *Current line*: Set to the first line of the edit buffer.

- 14302 *Current column*: Set to non-<blank>.
- 14303 Otherwise, if *file* is the same as the last file edited, the line and column shall be set as follows; if  
14304 the file was previously edited, the line and column may be set as follows:
- 14305 *Current line*: Set to the last value held when that file was last edited. If this value is not a valid  
14306 line in the new edit buffer, set to the first line of the edit buffer.
- 14307 *Current column*: If the current line was set to the last value held when the file was last edited, set  
14308 to the last value held when the file was last edited. Otherwise, or if the last value is not a valid  
14309 column in the new edit buffer, set to non-<blank>.
- 14310 Otherwise:
- 14311 *Current line*: Set to the first line of the edit buffer.
- 14312 *Current column*: Set to non-<blank>.
- 14313 **File**
- 14314 *Synopsis*:     f[ile][file]
- 14315 If a *file* argument is specified, the alternate pathname shall be set to the current pathname, and  
14316 the current pathname shall be set to *file*.
- 14317 Write an informational message. If the file has a current pathname, it shall be included in this  
14318 message; otherwise, the message shall indicate that there is no current pathname. If the edit  
14319 buffer contains lines, the current line number and the number of lines in the edit buffer shall be  
14320 included in this message; otherwise, the message shall indicate that the edit buffer is empty. If  
14321 the edit buffer has been modified since the last complete write, this fact shall be included in this  
14322 message. If the **readonly** edit option is set, this fact shall be included in this message. The  
14323 message may contain other unspecified information.
- 14324 *Current line*: Unchanged.
- 14325 *Current column*: Unchanged.
- 14326 **Global**
- 14327 *Synopsis*:     [2addr] g[lobal] /pattern/ [commands]  
14328                 [2addr] v /pattern/ [commands]
- 14329 The optional '!' character after the **global** command shall be the same as executing the **v**  
14330 command.
- 14331 If *pattern* is empty (for example, "//") or not specified, the last regular expression used in the  
14332 editor command shall be used as the *pattern*. The *pattern* can be delimited by slashes (shown in  
14333 the Synopsis), as well as any non-alphanumeric or non-<blank> other than backslash, vertical  
14334 line, double quote, or <newline>.
- 14335 If no lines are specified, the lines shall default to the entire file.
- 14336 The **global** and **v** commands are logically two-pass operations. First, mark the lines within the  
14337 specified lines for which the line excluding the terminating <newline> matches (**global**) or does  
14338 not match (**v** or **global!**) the specified pattern. Second, execute the *ex* commands given by  
14339 *commands*, with the current line ('.') set to each marked line. If an error occurs during this  
14340 process, or the contents of the edit buffer are replaced (for example, by the **ex:edit** command) an  
14341 error message shall be written and no more commands resulting from the execution of this  
14342 command shall be processed.

14343 Multiple **ex** commands can be specified by entering multiple commands on a single line using a  
 14344 vertical line to delimit them, or one per line, by escaping each <newline> with a backslash.

14345 If no commands are specified:

- 14346 1. If in **ex** command mode, it shall be as if the **print** command were specified.
- 14347 2. Otherwise, no command shall be executed.

14348 For the **append**, **change**, and **insert** commands, the input text shall be included as part of the  
 14349 command, and the terminating period can be omitted if the command ends the list of  
 14350 commands. The **open** and **visual** commands can be specified as one of the commands, in which  
 14351 case each marked line shall cause the editor to enter open or visual mode. If open or visual mode  
 14352 is exited using the **vi Q** command, the current line shall be set to the next marked line, and open  
 14353 or visual mode reentered, until the list of marked lines is exhausted.

14354 The **global**, **v**, and **undo** commands cannot be used in *commands*. Marked lines may be deleted  
 14355 by commands executed for lines occurring earlier in the file than the marked lines. In this case,  
 14356 no commands shall be executed for the deleted lines.

14357 If the remembered search direction is not set, the **global** and **v** commands shall set it to forward.

14358 The **autoprint** and **autoindent** edit options shall be inhibited for the duration of the **g** or **v**  
 14359 command.

14360 *Current line*: If no commands executed, set to the last marked line. Otherwise, as specified for  
 14361 the executed **ex** commands.

14362 *Current column*: If no commands are executed, set to non-<blank>; otherwise, as specified for the  
 14363 individual **ex** commands.

## 14364 **Insert**

14365 *Synopsis*: `[1addr] i[nsert][!]`

14366 Enter **ex** text input mode; the input text shall be placed before the specified line. If the line is zero  
 14367 or 1, the text shall be placed at the beginning of the edit buffer.

14368 This command shall be affected by the **number** and **autoindent** edit options; following the  
 14369 command name with `'!'` shall cause the **autoindent** edit option setting to be toggled for the  
 14370 duration of this command only.

14371 *Current line*: Set to the last input line; if no lines were input, set to the line before the specified  
 14372 line, or to the first line of the edit buffer if there are no lines preceding the specified line, or zero  
 14373 if the edit buffer is empty.

14374 *Current column*: Set to non-<blank>.

## 14375 **Join**

14376 *Synopsis*: `[2addr] j[oin][!][count][flags]`

14377 If *count* is specified:

14378 If no address was specified, the **join** command shall behave as if *2addr* were the current line |  
14379 and the current line plus *count* (*.,. + count*). |

14380 If one address was specified, the **join** command shall behave as if *2addr* were the specified |  
14381 address and the specified address plus *count* (*addr,addr + count*). |

14382 If two addresses were specified, the **join** command shall behave as if an additional address, |  
14383 equal to the last address plus *count* - 1 (*addr1,addr2,addr2 + count - 1*), was specified. |

14384 If this would result in a second address greater than the last line of the edit buffer, it shall be |  
14385 corrected to be equal to the last line of the edit buffer. |

14386 If no *count* is specified:

14387 If no address was specified, the **join** command shall behave as if *2addr* were the current line |  
14388 and the next line (*.,. + 1*). |

14389 If one address was specified, the **join** command shall behave as if *2addr* were the specified |  
14390 address and the next line (*addr,addr + 1*). |

14391 Join the text from the specified lines together into a single line, which shall replace the specified |  
14392 lines.

14393 If a '!' character is appended to the command name, the **join** shall be without modification of |  
14394 any line, independent of the current locale.

14395 Otherwise, in the POSIX locale, set the current line to the first of the specified lines, and then, for |  
14396 each subsequent line, proceed as follows:

- 14397 1. Discard leading <space>s from the line to be joined.
- 14398 2. If the line to be joined is now empty, delete it, and skip steps 3 through 5.
- 14399 3. If the current line ends in a <blank>, or the first character of the line to be joined is a ' ) ' |  
14400 character, join the lines without further modification.
- 14401 4. If the last character of the current line is a ' . ' , join the lines with two <space>s between |  
14402 them.
- 14403 5. Otherwise, join the lines with a single <space> between them.

14404 *Current line*: Set to the first line specified.

14405 *Current column*: Set to non-<blank>.

14406 **List**

14407 *Synopsis*: [2*addr*] l[*ist*][*count*][*flags*]

14408 This command shall be equivalent to the **ex** command:

14409 [2*addr*] p[*rint*][*count*] l[*flags*]

14410 See **Print** (on page 2580).

14411 **Map**14412 *Synopsis:* `map[!][lhs rhs]`14413 If *lhs* and *rhs* are not specified:

- 14414 1. If '!' is specified, write the current list of text input mode maps.
- 14415 2. Otherwise, write the current list of command mode maps.
- 14416 3. Do nothing more.

14417 Implementations may restrict the set of characters accepted in *lhs* or *rhs*, except that printable  
 14418 characters and <blank>s shall not be restricted. Additional restrictions shall be implementation-  
 14419 defined. In both *lhs* and *rhs*, any character can be escaped with a <control>-V, in which case the  
 14420 character shall not be used to delimit *lhs* from *rhs*, and the escaping <control>-V shall be  
 14421 discarded.

14422 If the character '!' is appended to the **map** command name, the mapping shall be effective  
 14423 during open or visual text input mode rather than **open** or **visual** command mode. This allows  
 14424 *lhs* to have two different **map** definitions at the same time: one for command mode and one for  
 14425 text input mode.

14426 For command mode mappings:

14427 When the *lhs* is entered as any part of a *vi* command in open or visual mode (but not as part  
 14428 of the arguments to the command), the action shall be as if the corresponding *rhs* had been  
 14429 entered.

14430 If any character in the command, other than the first, is escaped using a <control>-V  
 14431 character, that character shall not be part of a match to an *lhs*.

14432 It is unspecified whether implementations shall support **map** commands where the *lhs* is  
 14433 more than a single character in length, where the first character of the *lhs* is printable.

14434 If *lhs* contains more than one character and the first character is '#', followed by a sequence  
 14435 of digits corresponding to a numbered function key, then when this function key is typed it  
 14436 shall be mapped to *rhs*. Characters other than digits following a '#' character also  
 14437 represent the function key named by the characters in the *lhs* following the '#' and may be  
 14438 mapped to *rhs*. It is unspecified how function keys are named or what function keys are  
 14439 supported.

14440 For text input mode mappings:

14441 When the *lhs* is entered as any part of text entered in open or visual text input modes, the  
 14442 action shall be as if the corresponding *rhs* had been entered.

14443 If any character in the input text is escaped using a <control>-V character, that character shall  
 14444 not be part of a match to an *lhs*.

14445 It is unspecified whether the *lhs* argument entered for the **map** or **unmap** commands is  
 14446 replaced in this fashion. Regardless of whether or not the replacement occurs, the effect of  
 14447 the command shall be as if the replacement had not occurred.

14448 If only part of the *lhs* is entered, it is unspecified how long the editor will wait for additional,  
 14449 possibly matching characters before treating the already entered characters as not matching the  
 14450 *lhs*.

14451 The *rhs* characters shall themselves be subject to remapping, unless otherwise specified by the  
 14452 **remap** edit option, except that if the characters in *lhs* occur as prefix characters in *rhs*, those  
 14453 characters shall not be remapped.

14454 On block-mode terminals, the mapping need not occur immediately (for example, it may occur  
14455 after the terminal transmits a group of characters to the system), but it shall achieve the same  
14456 results as if it occurred immediately.

14457 *Current line*: Unchanged.

14458 *Current column*: Unchanged.

## 14459 **Mark**

14460 *Synopsis:* `[laddr] ma[rk] character`

14461 `[laddr] k character`

14462 Implementations shall support *character* values of a single lowercase letter of the POSIX locale  
14463 and the characters `'` and `'`; support of other characters is implementation-defined.

14464 If executing the `vi m` command, set the specified mark to the current line and 1-based numbered  
14465 character referenced by the current column, if any; otherwise, column position 1.

14466 Otherwise, set the specified mark to the specified line and 1-based numbered first non-<blank>  
14467 non- <newline> in the line, if any; otherwise, the last non-<newline> in the line, if any;  
14468 otherwise, column position 1.

14469 The mark shall remain associated with the line until the mark is reset or the line is deleted. If a  
14470 deleted line is restored by a subsequent **undo** command, any marks previously associated with  
14471 the line, which have not been reset, shall be restored as well. Any use of a mark not associated  
14472 with a current line in the edit buffer shall be an error.

14473 The marks `'` and `'` shall be set as described previously, immediately before the following events  
14474 occur in the editor:

- 14475 1. The use of `' $ '` as an *ex* address
- 14476 2. The use of a positive decimal number as an *ex* address
- 14477 3. The use of a search command as an *ex* address
- 14478 4. The use of a mark reference as an *ex* address
- 14479 5. The use of the following open and visual mode commands: <control>-], %, (, ), [, ], {, }.
- 14480 6. The use of the following open and visual mode commands: `'`, **G**, **H**, **L**, **M**, **z** if the current  
14481 line will change as a result of the command
- 14482 7. The use of the open and visual mode commands: `/`, `?`, **N**, `'`, **n** if the current line or column  
14483 will change as a result of the command
- 14484 8. The use of the *ex* mode commands: **z**, **undo**, **global**, **v**

14485 For rules 1., 2., 3., and 4., the `'` and `'` marks shall not be set if the *ex* command is parsed as  
14486 specified by rule 6.a. in **Command Line Parsing in ex** (on page 2563).

14487 For rules 5., 6., and 7., the `'` and `'` marks shall not be set if the commands are used as motion  
14488 commands in open and visual mode.

14489 For rules 1., 2., 3., 4., 5., 6., 7., and 8., the `'` and `'` marks shall not be set if the command fails.

14490 The `'` and `'` marks shall be set as described previously, each time the contents of the edit buffer  
14491 are replaced (including the editing of the initial buffer), if in open or visual mode, or if in **ex**  
14492 mode and the edit buffer is not empty, before any commands or movements (including  
14493 commands or movements specified by the `-c` or `-t` options or the `+command` argument) are  
14494 executed on the edit buffer. If in open or visual mode, the marks shall be set as if executing the `vi`

- 14495 **m** command; otherwise, as if executing the **ex mark** command.
- 14496 When changing from **ex** mode to open or visual mode, if the ‘ and ’ marks are not already set,  
14497 the ‘ and ’ marks shall be set as described previously.
- 14498 *Current line*: Unchanged.
- 14499 *Current column*: Unchanged.
- 14500 **Move**
- 14501 *Synopsis*: `[2addr] m[ove] 1addr [flags]`
- 14502 Move the specified lines after the specified destination line. A destination of line zero specifies  
14503 that the lines shall be placed at the beginning of the edit buffer. It shall be an error if the  
14504 destination line is within the range of lines to be moved.
- 14505 *Current line*: Set to the last of the moved lines.
- 14506 *Current column*: Set to non-<blank>.
- 14507 **Next**
- 14508 *Synopsis*: `n[ext][!][+command][file ...]`
- 14509 If no ‘!’ is appended to the command name, and the edit buffer has been modified since the  
14510 last complete write, it shall be an error, unless the file is successfully written as specified by the  
14511 **autowrite** option.
- 14512 If one or more files is specified:
- 14513 1. Set the argument list to the specified filenames.
  - 14514 2. Set the current argument list reference to be the first entry in the argument list.
  - 14515 3. Set the current pathname to the first filename specified.
- 14516 Otherwise:
- 14517 1. It shall be an error if there are no more filenames in the argument list after the filename  
14518 currently referenced.
  - 14519 2. Set the current pathname and the current argument list reference to the filename after the  
14520 filename currently referenced in the argument list.
- 14521 Replace the contents of the edit buffer with the contents of the file named by the current  
14522 pathname. If for any reason the contents of the file cannot be accessed, the edit buffer shall be  
14523 empty.
- 14524 This command shall be affected by the **autowrite** and **writeany** edit options.
- 14525 The *+command* option shall be <blank>-delimited; <blank>s can be escaped by preceding them  
14526 with a backslash character. The *+command* shall be interpreted as an **ex** command immediately  
14527 after the contents of the edit buffer have been replaced and the current line and column have  
14528 been set.
- 14529 *Current line*: Set as described for the **edit** command.
- 14530 *Current column*: Set as described for the **edit** command.

14531 **Number**

14532 *Synopsis:* [2addr] nu[mber][count][flags]  
 14533 [2addr] #[count][flags]

14534 These commands shall be equivalent to the *ex* command:

14535 [2addr] p[rint][count] #[flags]

14536 See **Print**.

14537 **Open**

14538 *Synopsis:* [1addr] o[pen] /pattern/ [flags]

14539 This command need not be supported on block-mode terminals or terminals with insufficient  
 14540 capabilities. If standard input, standard output, or standard error are not terminal devices, the  
 14541 results are unspecified.

14542 Enter open mode.

14543 The trailing delimiter can be omitted from pattern at the end of the command line. If pattern is  
 14544 empty (for example, "//") or not specified, the last regular expression used in the editor shall be  
 14545 used as the pattern. The pattern can be delimited by slashes (shown in the Synopsis), as well as  
 14546 any alphanumeric, or non-*<blank>* other than backslash, vertical line, double quote, or  
 14547 *<newline>*.

14548 *Current line:* Set to the specified line.

14549 *Current column:* Set to non-*<blank>*.

14550 **Preserve**

14551 *Synopsis:* pre[serve]

14552 Save the edit buffer in a form that can later be recovered by using the *-r* option or by using the *ex*  
 14553 **recover** command. After the file has been preserved, a mail message shall be sent to the user.  
 14554 This message shall be readable by invoking the *mailx* utility. The message shall contain the name  
 14555 of the file, the time of preservation, and an *ex* command that could be used to recover the file.  
 14556 Additional information may be included in the mail message.

14557 *Current line:* Unchanged.

14558 *Current column:* Unchanged.

14559 **Print**

14560 *Synopsis:* [2addr] p[rint][count][flags]

14561 Write the addressed lines. The behavior is unspecified if the number of columns on the display is  
 14562 less than the number of columns required to write any single character in the lines being written.

14563 Non-printable characters, except for the *<tab>*, shall be written as implementation-defined  
 14564 multi-character sequences.

14565 If the # flag is specified or the **number** edit option is set, each line shall be preceded by its line  
 14566 number in the following format:

14567 "%6dΔΔ", *<line number>*

14568 If the **l** flag is specified or the **list** edit option is set:

- 14569 1. The characters listed in the Base Definitions volume of IEEE Std 1003.1-200x, Table 5-1,  
 14570 Escape Sequences and Associated Actions shall be written as the corresponding escape  
 14571 sequence.
- 14572 2. Non-printable characters not in the Base Definitions volume of IEEE Std 1003.1-200x, Table  
 14573 5-1, Escape Sequences and Associated Actions shall be written as one three-digit octal  
 14574 number (with a preceding backslash) for each byte in the character (most significant byte  
 14575 first). If the size of a byte on the system is greater than 9 bits, the format used for non-  
 14576 printable characters is implementation-defined.
- 14577 3. The end of each line shall be marked with a '\$', and literal '\$' characters within the line  
 14578 shall be written with a preceding backslash.

14579 Long lines shall be folded; the length at which folding occurs is unspecified, but should be  
 14580 appropriate for the output terminal, considering the number of columns of the terminal.

14581 If a line is folded, and the **l** flag is not specified and the **list** edit option is not set, it is unspecified  
 14582 whether a multi-column character at the folding position is separated; it shall not be discarded.

14583 *Current line:* Set to the last written line.

14584 *Current column:* Unchanged if the current line is unchanged; otherwise, set to non-<blank>.

### 14585 **Put**

14586 *Synopsis:* [ *laddr* ] pu[t][*buffer*]

14587 Append text from the specified buffer (by default, the unnamed buffer) to the specified line; line  
 14588 zero specifies that the text shall be placed at the beginning of the edit buffer. Each portion of a  
 14589 line in the buffer shall become a new line in the edit buffer, regardless of the mode of the buffer.

14590 *Current line:* Set to the last line entered into the edit buffer.

14591 *Current column:* Set to non-<blank>.

### 14592 **Quit**

14593 *Synopsis:* q[uit][!]

14594 If no '!' is appended to the command name:

- 14595 1. If the edit buffer has been modified since the last complete write, it shall be an error.
- 14596 2. If there are filenames in the argument list after the filename currently referenced, and the  
 14597 last command was not a **quit**, **wq**, **xit**, or **ZZ** (see **Exit** (on page 3220)) command, it shall be  
 14598 an error.

14599 Otherwise, terminate the editing session.

### 14600 **Read**

14601 *Synopsis:* [ *laddr* ] r[ead][!][*file*]

14602 If '!' is not the first non-<blank> to follow the command name, a copy of the specified file shall  
 14603 be appended into the edit buffer after the specified line; line zero specifies that the copy shall be  
 14604 placed at the beginning of the edit buffer. The number of lines and bytes read shall be written. If  
 14605 no *file* is named, the current pathname shall be the default. If there is no current pathname, then  
 14606 *file* shall become the current pathname. If there is no current pathname or *file* operand, it shall be  
 14607 an error. Specifying a *file* that is not of type regular shall have unspecified results.

14608 Otherwise, if *file* is preceded by '!', the rest of the line after the '!' shall have '%', '#', and  
14609 '!' characters expanded as described in **Command Line Parsing in ex** (on page 2563).

14610 The *ex* utility shall then pass two arguments to the program named by the *shell* edit option; the  
14611 first shall be `-c` and the second shall be the expanded arguments to the **read** command as a  
14612 single argument. The standard input of the program shall be set to the standard input of the *ex*  
14613 program when it was invoked. The standard error and standard output of the program shall be  
14614 appended into the edit buffer after the specified line.

14615 Each line in the copied file or program output (as delimited by <newline>s or the end of the file  
14616 or output if it is not immediately preceded by a <newline>), shall be a separate line in the edit  
14617 buffer. Any occurrences of <carriage-return> and <newline> pairs in the output shall be treated  
14618 as single <newline>s.

14619 The special meaning of the '!' following the **read** command can be overridden by escaping it  
14620 with a backslash character.

14621 *Current line*: If no lines are added to the edit buffer, unchanged. Otherwise, if in open or visual  
14622 mode, set to the first line entered into the edit buffer. Otherwise, set to the last line entered into  
14623 the edit buffer.

14624 *Current column*: Set to non-<blank>.

## 14625 **Recover**

14626 *Synopsis*: `rec[over][!] file`

14627 If no '!' is appended to the command name, and the edit buffer has been modified since the  
14628 last complete write, it shall be an error.

14629 If no *file* operand is specified, then the current pathname shall be used. If there is no current  
14630 pathname or *file* operand, it shall be an error.

14631 If no recovery information has previously been saved about *file*, the **recover** command shall  
14632 behave identically to the **edit** command, and an informational message to this effect shall be  
14633 written.

14634 Otherwise, set the current pathname to *file*, and replace the current contents of the edit buffer  
14635 with the recovered contents of *file*. If there are multiple instances of the file to be recovered, the  
14636 one most recently saved shall be recovered, and an informational message that there are  
14637 previous versions of the file that can be recovered shall be written. The editor shall behave as if  
14638 the contents of the edit buffer have already been modified.

14639 *Current file*: Set as described for the **edit** command.

14640 *Current column*: Set as described for the **edit** command.

## 14641 **Rewind**

14642 *Synopsis*: `rew[ind][!]`

14643 If no '!' is appended to the command name, and the edit buffer has been modified since the  
14644 last complete write, it shall be an error, unless the file is successfully written as specified by the  
14645 **autowrite** option.

14646 If the argument list is empty, it shall be an error.

14647 The current argument list reference and the current pathname shall be set to the first filename in  
14648 the argument list.

14649 Replace the contents of the edit buffer with the contents of the file named by the current  
 14650 pathname. If for any reason the contents of the file cannot be accessed, the edit buffer shall be  
 14651 empty.

14652 This command shall be affected by the **autowrite** and **writeany** edit options.

14653 *Current line:* Set as described for the **edit** command.

14654 *Current column:* Set as described for the **edit** command.

## 14655 **Set**

14656 *Synopsis:*    `se[t][option]=[value] ...][nooption ...][option? ...][all]`

14657 When no arguments are specified, write the value of the **term** edit option and those options  
 14658 whose values have been changed from the default settings; when the argument *all* is specified,  
 14659 write all of the option values.

14660 Giving an option name followed by the character '?' shall cause the current value of that  
 14661 option to be written. The '?' can be separated from the option name by zero or more <blank>s.  
 14662 The '?' shall be necessary only for Boolean valued options. Boolean options can be given values  
 14663 by the form **set option** to turn them on or **set nooption** to turn them off; string and numeric  
 14664 options can be assigned by the form **set option=value**. Any <blank>s in strings can be included  
 14665 as is by preceding each <blank> with an escaping backslash. More than one option can be set or  
 14666 listed by a single set command by specifying multiple arguments, each separated from the next  
 14667 by one or more <blank>s.

14668 See **Edit Options in ex** (on page 2593) for details about specific options.

14669 *Current line:* Unchanged.

14670 *Current column:* Unchanged.

## 14671 **Shell**

14672 *Synopsis:*    `sh[ell]`

14673 Invoke the program named in the **shell** edit option with the single argument **-i** (interactive  
 14674 mode). Editing shall be resumed when the program exits.

14675 *Current line:* Unchanged.

14676 *Current column:* Unchanged.

## 14677 **Source**

14678 *Synopsis:*    `so[urce] file`

14679 Read and execute *ex* commands from *file*. Lines in the file that are blank lines shall be ignored.

14680 *Current line:* As specified for the individual *ex* commands.

14681 *Current column:* As specified for the individual *ex* commands.

14682 **Substitute**

14683 *Synopsis:* [2addr] s[substitute][/*pattern/repl*][*options*][*count*][*flags*]  
 14684 [2addr] &[*options*][*count*][*flags*]  
 14685 [2addr] ~[*options*][*count*][*flags*]

14686 Replace the first instance of the pattern *pattern* by the string *repl* on each specified line. (See  
 14687 **Regular Expressions in ex** (on page 2592) and **Replacement Strings in ex** (on page 2592).) Any  
 14688 non-alphabetic, non-<blank> delimiter other than '\\', '|', double quote, or <newline> can be  
 14689 used instead of '/'. Backslash characters can be used to escape delimiters, backslash  
 14690 characters, and other special characters.

14691 The trailing delimiter can be omitted from *pattern* or from *repl* at the end of the command line. If  
 14692 both *pattern* and *repl* are not specified or are empty (for example, "//"), the last **s** command  
 14693 shall be repeated. If only *pattern* is not specified or is empty, the last regular expression used in  
 14694 the editor shall be used as the pattern. If only *repl* is not specified or is empty, the pattern shall be  
 14695 replaced by nothing. If the entire replacement pattern is '%', the last replacement pattern to an  
 14696 **s** command shall be used.

14697 Entering a <carriage-return> in *repl* (which requires an escaping backslash in *ex* mode and an  
 14698 escaping <control>-V in open or *vi* mode) shall split the line at that point, creating a new line in  
 14699 the edit buffer. The <carriage-return> shall be discarded.

14700 If options include the letter 'g' (**global**), all non-overlapping instances of the pattern in the line  
 14701 shall be replaced.

14702 If options includes the letter 'c' (**confirm**), then before each substitution the line shall be  
 14703 written; the written line shall reflect all previous substitutions. On the following line, <space>s  
 14704 shall be written beneath the characters from the line that are before the *pattern* to be replaced,  
 14705 and '^' characters written beneath the characters included in the *pattern* to be replaced. The *ex*  
 14706 utility shall then wait for a response from the user. An affirmative response shall cause the  
 14707 substitution to be done, while any other input shall not make the substitution. An affirmative  
 14708 response shall consist of a line with the affirmative response (as defined by the current locale) at  
 14709 the beginning of the line. This line shall be subject to editing in the same way as the *ex* command  
 14710 line.

14711 If interrupted (see the ASYNCHRONOUS EVENTS section), any modifications confirmed by the  
 14712 user shall be preserved in the edit buffer after the interrupt.

14713 If the remembered search direction is not set, the **s** command shall set it to forward.

14714 In the second Synopsis, the **&** command shall repeat the previous substitution, as if the **&**  
 14715 command were replaced by:

14716 *s/pattern/repl/*

14717 where *pattern* and *repl* are as specified in the previous **s**, **&**, or **~** command.

14718 In the third Synopsis, the **~** command shall repeat the previous substitution, as if the **'~'** were  
 14719 replaced by:

14720 *s/pattern/repl/*

14721 where *pattern* shall be the last regular expression specified to the editor, and *repl* shall be from  
 14722 the previous substitution (including **&** and **~**) command.

14723 These commands shall be affected by the *LC\_MESSAGES* environment variable.

14724 *Current line:* Set to the last line in which a substitution occurred, or, unchanged if no  
 14725 substitution occurred.

14726 *Current column:* Set to non-<blank>.

## 14727 **Suspend**

14728 *Synopsis:*    su[suspend][!]  
14729            st[op][!]

14730 Allow control to return to the invoking process; *ex* shall suspend itself as if it had received the  
14731 SIGTSTP signal. The suspension shall occur only if job control is enabled in the invoking shell  
14732 (see the description of *set -m*).

14733 These commands shall be affected by the **autowrite** and **writeany** edit options.

14734 The current **susp** character (see *stty*) shall be equivalent to the **suspend** command. |

## 14735 **Tag**

14736 *Synopsis:*    ta[g][!] *tagstring*

14737 The results are unspecified if the format of a tags file is not as specified by the *ctags* utility (see  
14738 *ctags*) description.

14739 The **tag** command shall search for *tagstring* in the tag files referred to by the **tag** edit option, in  
14740 the order they are specified, until a reference to *tagstring* is found. Files shall be searched from  
14741 beginning to end. If no reference is found, it shall be an error and an error message to this effect  
14742 shall be written. If the reference is not found, or if an error occurs while processing a file referred  
14743 to in the **tag** edit option, it shall be an error, and an error message shall be written at the first  
14744 occurrence of such an error.

14745 Otherwise, if the tags file contained a pattern, the pattern shall be treated as a regular expression  
14746 used in the editor; for example, for the purposes of the **s** command.

14747 If the *tagstring* is in a file with a different name than the current pathname, set the current  
14748 pathname to the name of that file, and replace the contents of the edit buffer with the contents of  
14749 that file. In this case, if no '!' is appended to the command name, and the edit buffer has been  
14750 modified since the last complete write, it shall be an error, unless the file is successfully written  
14751 as specified by the **autowrite** option.

14752 This command shall be affected by the **autowrite**, **tag**, **taglength**, and **writeany** edit options.

14753 *Current line:* If the tags file contained a line number, set to that line number. If the line number is  
14754 larger than the last line in the edit buffer, an error message shall be written and the current line  
14755 shall be set as specified for the **edit** command.

14756 If the tags file contained a pattern, set to the first occurrence of the pattern in the file. If no  
14757 matching pattern is found, an error message shall be written and the current line shall be set as  
14758 specified for the **edit** command.

14759 *Current column:* If the tags file contained a line-number reference and that line-number was not  
14760 larger than the last line in the edit buffer, or if the tags file contained a pattern and that pattern  
14761 was found, set to non-<blank>. Otherwise, set as specified for the **edit** command.

14762 **Unabbreviate**14763 *Synopsis:* una[bbrev] lhs14764 If lhs is not an entry in the current list of abbreviations (see **Abbreviate** (on page 2571)), it shall  
14765 be an error. Otherwise, delete lhs from the list of abbreviations.14766 *Current line:* Unchanged.14767 *Current column:* Unchanged.14768 **Undo**14769 *Synopsis:* u[ndo]14770 Reverse the changes made by the last command that modified the contents of the edit buffer,  
14771 including **undo**. For this purpose, the **global**, **v**, **open**, and **visual** commands, and commands  
14772 resulting from buffer executions and mapped character expansions, are considered single  
14773 commands.14774 If no action that can be undone preceded the **undo** command, it shall be an error.14775 If the **undo** command restores lines that were marked, the mark shall also be restored unless it  
14776 was reset subsequent to the deletion of the lines.14777 *Current line:*

- 14778 1. If lines are added or changed in the file, set to the first line added or changed.
- 
- 14779 2. Set to the line before the first line deleted, if it exists.
- 
- 14780 3. Set to 1 if the edit buffer is not empty.
- 
- 14781 4. Set to zero.

14782 *Current column:* Set to non-<blank>.14783 **Unmap**14784 *Synopsis:* unm[ap][!] lhs14785 If '!' is appended to the command name, and if lhs is not an entry in the list of text input mode  
14786 map definitions, it shall be an error. Otherwise, delete lhs from the list of text input mode map  
14787 definitions.14788 If no '!' is appended to the command name, and if lhs is not an entry in the list of command  
14789 mode map definitions, it shall be an error. Otherwise, delete lhs from the list of command mode  
14790 map definitions.14791 *Current line:* Unchanged.14792 *Current column:* Unchanged.14793 **Version**14794 *Synopsis:* ve[rsion]14795 Write a message containing version information for the editor. The format of the message is  
14796 unspecified.14797 *Current line:* Unchanged.14798 *Current column:* Unchanged.

14799 **Visual**

14800 *Synopsis:* `[1addr] vi[sual][type][count][flags]`

14801 If *ex* is currently in open or visual mode, the Synopsis and behavior of the visual command shall  
14802 be the same as the **edit** command, as specified by **Edit** (on page 2573).

14803 Otherwise, this command need not be supported on block-mode terminals or terminals with  
14804 insufficient capabilities. If standard input, standard output, or standard error are not terminal  
14805 devices, the results are unspecified.

14806 If *count* is specified, the value of the **window** edit option shall be set to *count* (as described in  
14807 **window** (on page 2599)). If the '^' type character was also specified, the **window** edit option  
14808 shall be set before being used by the type character.

14809 Enter visual mode. If *type* is not specified, it shall be as if a *type* of '+' was specified. The *type*  
14810 shall cause the following effects:

- 14811 + Place the beginning of the specified line at the top of the display.
- 14812 - Place the end of the specified line at the bottom of the display.
- 14813 . Place the beginning of the specified line in the middle of the display.
- 14814 ^ If the specified line is less than or equal to the value of the **window** edit option, set the line  
14815 to 1; otherwise, decrement the line by the value of the **window** edit option minus 1. Place  
14816 the beginning of this line as close to the bottom of the displayed lines as possible, while still  
14817 displaying the value of the **window** edit option number of lines.

14818 *Current line:* Set to the specified line.

14819 *Current column:* Set to non-<blank>.

14820 **Write**

14821 *Synopsis:* `[2addr] w[rite][!][>>][file]`  
14822 `[2addr] w[rite][!][file]`  
14823 `[2addr] wq[!][>>][file]`

14824 If no lines are specified, the lines shall default to the entire file.

14825 The command **wq** shall be equivalent to a **write** command followed by a **quit** command; **wq!**  
14826 shall be equivalent to **write!** followed by **quit**. In both cases, if the **write** command fails, the  
14827 **quit** shall not be attempted.

14828 If the command name is not followed by one or more <blank>s, or *file* is not preceded by a '!'  
14829 character, the **write** shall be to a file.

- 14830 1. If the >> argument is specified, and the file already exists, the lines shall be appended to  
14831 the file instead of replacing its contents. If the >> argument is specified, and the file does  
14832 not already exist, it is unspecified whether the write shall proceed as if the >> argument  
14833 had not been specified or if the write shall fail.
- 14834 2. If the **readonly** edit option is set (see **readonly** (on page 2596)), the **write** shall fail.
- 14835 3. If *file* is specified, and is not the current pathname, and the file exists, the **write** shall fail.
- 14836 4. If *file* is not specified, the current pathname shall be used. If there is no current pathname,  
14837 the **write** command shall fail.
- 14838 5. If the current pathname is used, and the current pathname has been changed by the **file** or  
14839 **read** commands, and the file exists, the **write** shall fail. If the **write** is successful,

14840 subsequent **writes** shall not fail for this reason (unless the current pathname is changed  
14841 again).

14842 6. If the whole edit buffer is not being written, and the file to be written exists, the **write** shall  
14843 fail.

14844 For rules 1., 2., 4., and 5., the **write** can be forced by appending the character **'!'** to the  
14845 command name.

14846 For rules 2., 4., and 5., the **write** can be forced by setting the **writeany** edit option.

14847 Additional, implementation-defined tests may cause the **write** to fail.

14848 If the edit buffer is empty, a file without any contents shall be written.

14849 An informational message shall be written noting the number of lines and bytes written.

14850 Otherwise, if the command is followed by one or more **<blank>**s, and the file is preceded by  
14851 **'!'**, the rest of the line after the **'!'** shall have **'%'**, **'#'**, and **'!'** characters expanded as  
14852 described in **Command Line Parsing in ex** (on page 2563).

14853 The **ex** utility shall then pass two arguments to the program named by the **shell** edit option; the  
14854 first shall be **-c** and the second shall be the expanded arguments to the **write** command as a  
14855 single argument. The specified lines shall be written to the standard input of the command. The  
14856 standard error and standard output of the program, if any, shall be written as described for the  
14857 **print** command. If the last character in that output is not a **<newline>**, a **<newline>** shall be  
14858 written at the end of the output.

14859 The special meaning of the **'!'** following the **write** command can be overridden by escaping it  
14860 with a backslash character.

14861 *Current line:* Unchanged.

14862 *Current column:* Unchanged.

## 14863 **Write and Exit**

14864 *Synopsis:* `[2addr] x[it][!][file]`

14865 If the edit buffer has not been modified since the last complete **write**, **xit** shall be equivalent to  
14866 the **quit** command, or if a **'!'** is appended to the command name, to **quit!**.

14867 Otherwise, **xit** shall be equivalent to the **wq** command, or if a **'!'** is appended to the command  
14868 name, to **wq!**.

14869 *Current line:* Unchanged.

14870 *Current column:* Unchanged.

## 14871 **Yank**

14872 *Synopsis:* `[2addr] ya[nk][buffer][count]`

14873 Copy the specified lines to the specified buffer (by default, the unnamed buffer), which shall  
14874 become a line-mode buffer.

14875 *Current line:* Unchanged.

14876 *Current column:* Unchanged.

14877

**Adjust Window**

14878

*Synopsis:* `[laddr] z[!][type ...][count][flags]`

14879

If no line is specified, the current line shall be the default; if *type* is omitted as well, the current line value shall first be incremented by 1. If incrementing the current line would cause it to be greater than the last line in the edit buffer, it shall be an error.

14880

14881

14882

If there are <blank>s between the *type* argument and the preceding *z* command name or optional '!' character, it shall be an error.

14883

14884

If *count* is specified, the value of the **window** edit option shall be set to *count* (as described in **window** (on page 2599)). If *count* is omitted, it shall default to 2 times the value of the **scroll** edit option, or if ! was specified, the number of lines in the display minus 1.

14885

14886

14887

If *type* is omitted, then *count* lines starting with the specified line shall be written. Otherwise, *count* lines starting with the line specified by the *type* argument shall be written.

14888

14889

The *type* argument shall change the lines to be written. The possible values of *type* are as follows:

14890

– The specified line shall be decremented by the following value:

14891

$$((\text{number of ``-'' characters}) \times \text{count}) - 1$$

14892

If the calculation would result in a number less than 1, it shall be an error. Write lines from the edit buffer, starting at the new value of line, until *count* lines or the last line in the edit buffer has been written.

14893

14894

14895

+ The specified line shall be incremented by the following value:

14896

$$((\text{number of ``+'' characters}) - 1) \times \text{count} + 1$$

14897

If the calculation would result in a number greater than the last line in the edit buffer, it shall be an error. Write lines from the edit buffer, starting at the new value of line, until *count* lines or the last line in the edit buffer has been written.

14898

14899

14900

=, . If more than a single ' .' or '=' is specified, it shall be an error. The following steps shall be taken:

14901

14902

1. If *count* is zero, nothing shall be written.

14903

2. Write as many of the *N* lines before the current line in the edit buffer as exist. If *count* or '!' was specified, *N* shall be:

14904

$$(\text{count} - 1) / 2$$

14905

14906

Otherwise, *N* shall be:

14907

$$(\text{count} - 3) / 2$$

14908

If *N* is a number less than 3, no lines shall be written.

14909

3. If '=' was specified as the type character, write a line consisting of the smaller of the number of columns in the display divided by two, or 40 '-' characters.

14910

14911

4. Write the current line.

14912

5. Repeat step 3.

14913

6. Write as many of the *N* lines after the current line in the edit buffer as exist. *N* shall be defined as in step 2. If *N* is a number less than 3, no lines shall be written. current line in the edit buffer as exist. If *count* is less than 3, no lines shall be written.

14914

14915

- 14916        $\wedge$    The specified line shall be decremented by the following value:
- 14917            $((\text{number of ``\^`` characters}) + 1) \times \text{count} - 1$
- 14918           If the calculation would result in a number less than 1, it shall be an error. Write lines from  
14919           the edit buffer, starting at the new value of line, until *count* lines or the last line in the edit  
14920           buffer has been written.
- 14921       *Current line*: Set to the last line written, unless the type is =, in which case, set to the specified  
14922       line.
- 14923       *Current column*: Set to non-<blank>.
- 14924       **Escape**
- 14925       *Synopsis*:        ! *command*  
14926                    [*addr*]! *command*
- 14927       The contents of the line after the '!' shall have '%', '#', and '!' characters expanded as  
14928       described in **Command Line Parsing in ex** (on page 2563). If the expansion causes the text of the  
14929       line to change, it shall be redisplayed, preceded by a single '!' character.
- 14930       The *ex* utility shall execute the program named by the **shell** edit option. It shall pass two  
14931       arguments to the program; the first shall be **-c**, and the second shall be the expanded arguments  
14932       to the ! command as a single argument.
- 14933       If no lines are specified, the standard input, standard output, and standard error of the program  
14934       shall be set to the standard input, standard output, and standard error of the *ex* program when it  
14935       was invoked. In addition, a warning message shall be written if the edit buffer has been  
14936       modified since the last complete write, and the **warn** edit option is set.
- 14937       If lines are specified, they shall be passed to the program as standard input, and the standard  
14938       output and standard error of the program shall replace those lines in the edit buffer. Each line in  
14939       the program output (as delimited by <newline>s or the end of the output if it is not immediately  
14940       preceded by a <newline>), shall be a separate line in the edit buffer. Any occurrences of  
14941       <carriage-return> and <newline> pairs in the output shall be treated as single <newline>s. The  
14942       specified lines shall be copied into the unnamed buffer before they are replaced, and the  
14943       unnamed buffer shall become a line-mode buffer.
- 14944       If in *ex* mode, a single '!' character shall be written when the program completes.
- 14945       This command shall be affected by the **shell** and **warn** edit options. If no lines are specified, this  
14946       command shall be affected by the **autowrite** and **writeany** edit options. If lines are specified, this  
14947       command shall be affected by the **autoprint** edit option.
- 14948       *Current line*:
- 14949           1. If no lines are specified, unchanged.
  - 14950           2. Otherwise, set to the last line read in, if any lines are read in.
  - 14951           3. Otherwise, set to the line before the first line of the lines specified, if that line exists.
  - 14952           4. Otherwise, set to the first line of the edit buffer if the edit buffer is not empty.
  - 14953           5. Otherwise, set to zero.
- 14954       *Current column*: If no lines are specified, unchanged. Otherwise, set to non-<blank>.

14955       **Shift Left**14956       *Synopsis:*     [*2addr*] <[< ...][*count*][*flags*]14957       Shift the specified lines to the start of the line; the number of column positions to be shifted shall  
14958       be the number of command characters times the value of the **shiftwidth** edit option. Only  
14959       leading <blank>s shall be deleted or changed into other <blank>s in shifting; other characters  
14960       shall not be affected.14961       Lines to be shifted shall be copied into the unnamed buffer, which shall become a line-mode  
14962       buffer.14963       This command shall be affected by the **autoprint** edit option.14964       *Current line:* Set to the last line in the lines specified.14965       *Current column:* Set to non-<blank>.14966       **Shift Right**14967       *Synopsis:*     [*2addr*] >[> ...][*count*][*flags*]14968       Shift the specified lines away from the start of the line; the number of column positions to be  
14969       shifted shall be the number of command characters times the value of the **shiftwidth** edit option.  
14970       The shift shall be accomplished by adding <blank>s as a prefix to the line or changing leading  
14971       <blank>s into other <blank>s. Empty lines shall not be changed.14972       Lines to be shifted shall be copied into the unnamed buffer, which shall become a line-mode  
14973       buffer.14974       This command shall be affected by the **autoprint** edit option.14975       *Current line:* Set to the last line in the lines specified.14976       *Current column:* Set to non-<blank>.

## 14977       &lt;control&gt;-D

14978       *Synopsis:*     <control>-D14979       Write the next *n* lines, where *n* is the minimum of the values of the **scroll** edit option and the  
14980       number of lines after the current line in the edit buffer. If the current line is the last line of the  
14981       edit buffer it shall be an error.14982       *Current line:* Set to the last line written.14983       *Current column:* Set to non-<blank>.14984       **Write Line Number**14985       *Synopsis:*     [*laddr*] = [*flags*]14986       If *line* is not specified, it shall default to the last line in the edit buffer. Write the line number of  
14987       the specified line.14988       *Current line:* Unchanged.14989       *Current column:* Unchanged.

14990 **Execute**

14991 *Synopsis:* [2addr] @ *buffer*  
 14992 [2addr] \* *buffer*

14993 If no buffer is specified or is specified as '@' or '\*', the last buffer executed shall be used. If no  
 14994 previous buffer has been executed, it shall be an error.

14995 For each line specified by the addresses, set the current line ('.') to the specified line, and  
 14996 execute the contents of the named *buffer* (as they were at the time the @ command was executed)  
 14997 as *ex* commands. For each line of a line-mode buffer, and all but the last line of a character-mode  
 14998 buffer, the *ex* command parser shall behave as if the line was terminated by a <newline>.

14999 If an error occurs during this process, or a line specified by the addresses does not exist when the  
 15000 current line would be set to it, or more than a single line was specified by the addresses, and the  
 15001 contents of the edit buffer are replaced (for example, by the *ex:edit* command) an error message  
 15002 shall be written, and no more commands resulting from the execution of this command shall be  
 15003 processed.

15004 *Current line:* As specified for the individual *ex* commands.

15005 *Current column:* As specified for the individual *ex* commands.

15006 **Regular Expressions in ex**

15007 The *ex* utility shall support regular expressions that are a superset of the basic regular  
 15008 expressions described in the Base Definitions volume of IEEE Std 1003.1-200x, Section 9.3, Basic  
 15009 Regular Expressions. A null regular expression ("//") shall be equivalent to the last regular  
 15010 expression encountered.

15011 Regular expressions can be used in addresses to specify lines and, in some commands (for  
 15012 example, the **substitute** command), to specify portions of a line to be substituted.

15013 The following constructs can be used to enhance the basic regular expressions:

15014 \< < Match the beginning of a *word*. (See the definition of *word* at the beginning of **Command**  
 15015 **Descriptions in ex** (on page 2569).)

15016 \< > Match the end of a *word*.

15017 ~ Match the replacement part of the last **substitute** command. The tilde ('~') character can  
 15018 be escaped in a regular expression to become a normal character with no special meaning.  
 15019 The backslash shall be discarded.

15020 When the editor option **magic** is not set, the only characters with special meanings shall be '^'  
 15021 at the beginning of a pattern, '\$' at the end of a pattern, and '\'. The characters '.', '\*',  
 15022 '[', and '~' shall be treated as ordinary characters unless preceded by a '\'; when preceded  
 15023 by a '\' they shall regain their special meaning, or in the case of backslash, be handled as a  
 15024 single backslash. Backslashes used to escape other characters shall be discarded.

15025 **Replacement Strings in ex**

15026 The character '&' ('\&' if the editor option **magic** is not set) in the replacement string shall  
 15027 stand for the text matched by the pattern to be replaced. The character '~' ('\~' if **magic** is not  
 15028 set) shall be replaced by the replacement part of the previous **substitute** command. The  
 15029 sequence '\n', where *n* is an integer, shall be replaced by the text matched by the pattern  
 15030 enclosed in the *n*th set of parentheses '\(' and '\)'.  
 15031

15031 The strings '\l', '\u', '\L', and '\U' can be used to modify the case of elements in the  
 15032 replacement string (using the '\&' or "\"digit) notation. The string '\l' ('\u') shall cause

15033 the character that follows to be converted to lowercase (uppercase). The string '`\L`' ('`\U`')  
 15034 shall cause all characters subsequent to it to be converted to lowercase (uppercase) as they are  
 15035 inserted by the substitution until the string '`\e`' or '`\E`', or the end of the replacement string,  
 15036 is encountered.

15037 Otherwise, any character following a backslash shall be treated as that literal character, and the  
 15038 escaping backslash shall be discarded.

15039 An example of case conversion with the `s` command is as follows:

```
15040 :p
15041 The cat sat on the mat.
15042 :s/\<.at\>/\u&/gp
15043 The Cat Sat on the Mat.
15044 :s/S\(.*\)M/S\U\1\eM/p
15045 The Cat SAT ON THE Mat.
```

## 15046 Edit Options in `ex`

15047 The `ex` utility has a number of options that modify its behavior. These options have default  
 15048 settings, which can be changed using the `set` command.

15049 Options are Boolean unless otherwise specified.

### 15050 `autoindent`, `ai`

15051 [Default `unset`]

15052 If `autoindent` is set, each line in input mode shall be indented (using first as many `<tab>`s as  
 15053 possible, as determined by the editor option `tabstop`, and then using `<space>`s) to align with  
 15054 another line, as follows:

- 15055 1. If in open or visual mode and the text input is part of a line-oriented command (see the  
 15056 EXTENDED DESCRIPTION in `vi`), align to the first column. Otherwise, if in open or  
 15057 visual mode, indentation for each line shall be set as follows:
  - 15058 a. If a line was previously inserted as part of this command, it shall be set to the  
 15059 indentation of the last inserted line by default, or as otherwise specified for the  
 15060 `<control>-D` character in **Input Mode Commands in `vi`** (on page 3220).
  - 15061 b. Otherwise, it shall be set to the indentation of the previous current line, if any;  
 15062 otherwise, to the first column.
- 15063 2. For the `ex a`, `i`, and `c` commands, indentation for each line shall be set as follows:
  - 15064 a. If a line was previously inserted as part of this command, it shall be set to the  
 15065 indentation of the last inserted line by default, or as otherwise specified for the `eof`  
 15066 character in **Scroll** (on page 2567).
  - 15067 b. Otherwise, if the command is the `ex a` command, it shall be set to the line appended  
 15068 after, if any; otherwise to the first column.
  - 15069 c. Otherwise, if the command is the `ex i` command, it shall be set to the line inserted  
 15070 before, if any; otherwise to the first column.
  - 15071 d. Otherwise, if the command is the `ex c` command, it shall be set to the indentation of  
 15072 the line replaced.

15073 **autoprint, ap**15074 [Default *set*]

15075 If **autoprint** is set, the current line shall be written after each **ex** command that modifies the  
 15076 contents of the current edit buffer, and after each **tag** command for which the tag search pattern  
 15077 was found or tag line number was valid, unless:

- 15078 1. The command was executed while in open or visual mode.
- 15079 2. The command was executed as part of a **global** or **v** command or **@** buffer execution.
- 15080 3. The command was the form of the **read** command that reads a file into the edit buffer.
- 15081 4. The command was the **append**, **change**, or **insert** command.
- 15082 5. The command was not terminated by a <newline>.
- 15083 6. The current line shall be written by a flag specified to the command; for example, **delete #**  
 15084 shall write the current line as specified for the flag modifier to the **delete** command, and  
 15085 not as specified by the **autoprint** edit option.

15086 **autowrite, aw**15087 [Default *unset*]

15088 If **autowrite** is set, and the edit buffer has been modified since it was last completely written to  
 15089 any file, the contents of the edit buffer shall be written as if the **ex write** command had been  
 15090 specified without arguments, before each command affected by the **autowrite** edit option is  
 15091 executed. Appending the character **'!'** to the command name of any of the **ex** commands  
 15092 except **'!'** shall prevent the write. If the write fails, it shall be an error and the command shall  
 15093 not be executed.

15094 **beautify, bf**15095 XSI [Default *unset*]

15096 If **beautify** is set, all non-printable characters, other than <tab>s, <newline>s, and <form-feed>s,  
 15097 shall be discarded from text read in from files.

15098 **directory, dir**15099 [Default *implementation-defined*]

15100 The value of this option specifies the directory in which the editor buffer is to be placed. If this  
 15101 directory is not writable by the user, the editor shall quit.

15102 **edcompatible, ed**15103 [Default *unset*]

15104 Causes the presence of **g** and **c** suffixes on substitute commands to be remembered, and toggled  
 15105 by repeating the suffixes.

- 15106        **errorbells, eb**  
15107        [Default *unset*]  
15108        If the editor is in *ex* mode, and the terminal does not support a standout mode (such as inverse  
15109        video), and **errorbells** is set, error messages shall be preceded by alerting the terminal.
- 15110        **exrc**  
15111        [Default *unset*]  
15112        If **exrc** is set, *ex* shall access any **.exrc** file in the current directory, as described in **Initialization in**  
15113        **ex and vi** (on page 2559). If **exrc** is not set, *ex* shall ignore any **.exrc** file in the current directory  
15114        during initialization, unless the current directory is that named by the *HOME* environment  
15115        variable.
- 15116        **ignorecase, ic**  
15117        [Default *unset*]  
15118        If **ignorecase** is set, characters that have uppercase and lowercase representations shall have  
15119        those representations considered as equivalent for purposes of regular expression comparison.  
15120        The **ignorecase** edit option shall affect all remembered regular expressions; for example,  
15121        unsetting the **ignorecase** edit option shall cause a subsequent *vi n* command to search for the  
15122        last basic regular expression in a case-sensitive fashion.
- 15123        **list**  
15124        [Default *unset*]  
15125        If **list** is set, edit buffer lines written while in *ex* command mode shall be written as specified for  
15126        the **print** command with the **l** flag specified. In open or visual mode, each edit buffer line shall  
15127        be displayed as specified for the *ex print* command with the **l** flag specified. In open or visual  
15128        text input mode, when the cursor does not rest on any character in the line, it shall rest on the  
15129        ' \$ ' marking the end of the line.
- 15130        **magic**  
15131        [Default *set*]  
15132        If **magic** is set, modify the interpretation of characters in regular expressions and substitution  
15133        replacement strings (see **Regular Expressions in ex** (on page 2592) and **Replacement Strings in**  
15134        **ex** (on page 2592)).
- 15135        **mesg**  
15136        [Default *set*]  
15137        If **mesg** is set, the permission for others to use the **write** or **talk** commands to write to the  
15138        terminal shall be turned on while in open or visual mode. The shell-level command *mesg n* shall  
15139        take precedence over any setting of the *ex mesg* option; that is, if **mesg y** was issued before the  
15140        editor started (or in a shell escape), such as:  
15141        : !mesg y  
15142        the **mesg** option in *ex* shall suppress incoming messages, but the **mesg** option shall not enable  
15143        incoming messages if **mesg n** was issued.

15144 **number, nu**

15145 [Default *unset*]

15146 If **number** is set, edit buffer lines written while in *ex* command mode shall be written with line  
15147 numbers, in the format specified by the **print** command with the # flag specified. In *ex* text input  
15148 mode, each line shall be preceded by the line number it will have in the file.

15149 In open or visual mode, each edit buffer line shall be displayed with a preceding line number, in  
15150 the format specified by the *ex* **print** command with the # flag specified. This line number shall  
15151 not be considered part of the line for the purposes of evaluating the current column; that is,  
15152 column position 1 shall be the first column position after the format specified by the **print**  
15153 command.

15154 **paragraphs, para**

15155 [Default in the POSIX locale `IPLPPPQPP LIpplpipbp`]

15156 The **paragraphs** edit option shall define additional paragraph boundaries for the open and visual  
15157 mode commands. The **paragraphs** edit option can be set to a character string consisting of zero  
15158 or more character pairs. It shall be an error to set it to an odd number of characters.

15159 **prompt**

15160 [Default *set*]

15161 If **prompt** is set, *ex* command mode input shall be prompted for with a colon (':'); when unset,  
15162 no prompt shall be written.

15163 **readonly**

15164 [Default *see text*]

15165 If **readonly** edit option is set, read-only mode shall be enabled (see **Write** (on page 2587)). The  
15166 **readonly** edit option shall be initialized to set if either of the following conditions are true:

- 15167
- The command-line option `-R` was specified.
  - Performing actions equivalent to the `access()` function called with the following arguments  
15168 indicates that the file lacks write permission:  
15169
    1. The current pathname is used as the *path* argument.
    - 15170
    2. The constant `W_OK` is used as the *amode* argument.
    - 15171

15172 The **readonly** edit option may be initialized to set for other, implementation-defined reasons.  
15173 The **readonly** edit option shall not be initialized to unset based on any special privileges of the  
15174 user or process. The **readonly** edit option shall be reinitialized each time that the contents of the  
15175 edit buffer are replaced (for example, by an **edit** or **next** command) unless the user has explicitly  
15176 set it, in which case it shall remain set until the user explicitly unsets it. Once unset, it shall again  
15177 be reinitialized each time that the contents of the edit buffer are replaced.

15178 **redraw**15179 [Default *unset*]

15180 The editor simulates an intelligent terminal on a dumb terminal. (Since this is likely to require a  
 15181 large amount of output to the terminal, it is useful only at high transmission speeds.)

15182 **remap**15183 [Default *set*]

15184 If **remap** is set, map translation shall allow for maps defined in terms of other maps; translation  
 15185 shall continue until a final product is obtained. If *unset*, only a one-step translation shall be done.

15186 **report**

15187 [Default 5]

15188 The value of this **report** edit option specifies what number of lines being added, copied, deleted,  
 15189 or modified in the edit buffer will cause an informational message to be written to the user. The  
 15190 following conditions shall cause an informational message. The message shall contain the  
 15191 number of lines added, copied, deleted, or modified, but is otherwise unspecified.

- 15192 • An *ex* or *vi* editor command, other than **open**, **undo**, or **visual**, that modifies at least the value  
 15193 of the **report** edit option number of lines, and which is not part of an *ex* **global** or *v*  
 15194 command, or *ex* or *vi* buffer execution, shall cause an informational message to be written.

- 15195 • An *ex* **yank** or *vi* **y** or **Y** command, that copies at least the value of the **report** edit option plus  
 15196 1 number of lines, and which is not part of an *ex* **global** or *v* command, or *ex* or *vi* buffer  
 15197 execution, shall cause an informational message to be written.

- 15198 • An *ex* **global**, *v*, **open**, **undo**, or **visual** command or *ex* or *vi* buffer execution, that adds or  
 15199 deletes a total of at least the value of the **report** edit option number of lines, and which is not  
 15200 part of an *ex* **global** or *v* command, or *ex* or *vi* buffer execution, shall cause an informational  
 15201 message to be written. (For example, if 3 lines were added and 8 lines deleted during an *ex*  
 15202 **visual** command, 5 would be the number compared against the **report** edit option after the  
 15203 command completed.

15204 **scroll, scr**

15205 [Default (number of lines in the display -1)/2]

15206 The value of the **scroll** edit option shall determine the number of lines scrolled by the *ex* |  
 15207 <control>-D and **z** commands. For the *vi* <control>-D and <control>-U commands, it shall be the  
 15208 initial number of lines to scroll when no previous <control>-D or <control>-U command has  
 15209 been executed.

15210 **sections**15211 [Default in the POSIX locale `NHSHH HUnhsh`]

15212 The **sections** edit option shall define additional section boundaries for the open and visual mode  
 15213 commands. The **sections** edit option can be set to a character string consisting of zero or more  
 15214 character pairs; it shall be an error to set it to an odd number of characters.

- 15215        **shell, sh**  
15216        [Default from the environment variable *SHELL*]  
15217        The value of this option shall be a string. The default shall be taken from the *SHELL*  
15218        environment variable. If the *SHELL* environment variable is null or empty, the *sh* (see *sh*) utility  
15219        shall be the default.
- 15220        **shiftwidth, sw**  
15221        [Default 8]  
15222        The value of this option shall give the width in columns of an indentation level used during  
15223        autoindentation and by the shift commands (< and >).
- 15224        **showmatch, sm**  
15225        [Default *unset*]  
15226        The functionality described for the **showmatch** edit option need not be supported on block-  
15227        mode terminals or terminals with insufficient capabilities.  
15228        If **showmatch** is set, in open or visual mode, when a ' ) ' or ' } ' is typed, if the matching ' ( ' or  
15229        ' { ' is currently visible on the display, the matching ' ( ' or ' { ' shall be flagged moving the  
15230        cursor to its location for an unspecified amount of time.
- 15231        **showmode**  
15232        [Default *unset*]  
15233        If **showmode** is set, in open or visual mode, the current mode that the editor is in shall be  
15234        displayed on the last line of the display. Command mode and text input mode shall be  
15235        differentiated; other unspecified modes and implementation-defined information may be  
15236        displayed.
- 15237        **slowopen**  
15238        [Default *unset*]  
15239        If **slowopen** is set during open and visual text input modes, the editor shall not update portions  
15240        of the display other than those display line columns that display the characters entered by the  
15241        user (see **Input Mode Commands in vi** (on page 3220)).
- 15242        **tabstop, ts**  
15243        [Default 8]  
15244        The value of this edit option shall specify the column boundary used by a <tab> in the display  
15245        (see **autoprint, ap** (on page 2594) and **Input Mode Commands in vi** (on page 3220)).
- 15246        **taglength, tl**  
15247        [Default zero]  
15248        The value of this edit option shall specify the maximum number of characters that are  
15249        considered significant in the user-specified tag name and in the tag name from the tags file. If the  
15250        value is zero, all characters in both tag names shall be significant.

- 15251       **tags**  
15252       [Default *see text*]  
15253       The value of this edit option shall be a string of <blank>-delimited pathnames of files used by  
15254       the **tag** command. The default value is unspecified.
- 15255       **term**  
15256       [Default from the environment variable *TERM*]  
15257       The value of this edit option shall be a string. The default shall be taken from the *TERM* variable  
15258       in the environment. If the *TERM* environment variable is empty or null, the default is  
15259       unspecified. The editor shall use the value of this edit option to determine the type of the display  
15260       device.  
15261       The results are unspecified if the user changes the value of the term edit option after editor  
15262       initialization.
- 15263       **terse**  
15264       [Default *unset*]  
15265       If **terse** is set, error messages may be less verbose. However, except for this caveat, error  
15266       messages are unspecified. Furthermore, not all error messages need change for different settings  
15267       of this option.
- 15268       **warn**  
15269       [Default *set*]  
15270       If **warn** is set, and the contents of the edit buffer have been modified since they were last  
15271       completely written, the editor shall write a warning message before certain ! commands (see  
15272       **Escape** (on page 2590)).
- 15273       **window**  
15274       [Default *see text*]  
15275       A value used in open and visual mode, by the <control>-B and <control>-F commands, and, in  
15276       visual mode, to specify the number of lines displayed when the screen is repainted.  
15277       If the **-w** command-line option is not specified, the default value shall be set to the value of the  
15278       *LINES* environment variable. If the *LINES* environment variable is empty or null, the default  
15279       shall be the number of lines in the display minus 1.  
15280       Setting the **window** edit option to zero or to a value greater than the number of lines in the  
15281       display minus 1 (either explicitly or based on the **-w** option or the *LINES* environment variable)  
15282       shall cause the **window** edit option to be set to the number of lines in the display minus 1.  
15283       The baud rate of the terminal line may change the default in an implementation-defined manner.

- 15284       **wrapmargin, wm**
- 15285       [Default 0]
- 15286       If the value of this edit option is zero, it shall have no effect.
- 15287       If not in the POSIX locale, the effect of this edit option is implementation-defined.
- 15288       Otherwise, it shall specify a number of columns from the ending margin of the terminal.
- 15289       During open and visual text input modes, for each character for which any part of the character  
15290       is displayed in a column that is less than **wrapmargin** columns from the ending margin of the  
15291       display line, the editor shall behave as follows:
- 15292       1. If the character triggering this event is a <blank>, it, and all immediately preceding  
15293       <blank>s on the current line entered during the execution of the current text input  
15294       command, shall be discarded, and the editor shall behave as if the user had entered a single  
15295       <newline> instead. In addition, if the next user-entered character is a <space>, it shall be  
15296       discarded as well.
  - 15297       2. Otherwise, if there are one or more <blank>s on the current line immediately preceding the  
15298       last group of inserted non-<blank>s which was entered during the execution of the current  
15299       text input command, the <blank>s shall be replaced as if the user had entered a single  
15300       <newline> instead.
- 15301       If the **autoindent** edit option is set, and the events described in 1. or 2. are performed, any  
15302       <blank>s at or after the cursor in the current line shall be discarded.
- 15303       The ending margin shall be determined by the system or overridden by the user, as described for  
15304       **COLUMNS** in in the ENVIRONMENT VARIABLES section and the Base Definitions volume of  
15305       IEEE Std 1003.1-200x, Chapter 8, Environment Variables.
- 15306       **wrapscan, ws**
- 15307       [Default *set*]
- 15308       If **wrapscan** is set, searches (the *ex* / or ? addresses, or open and visual mode /, ?, N, and n  
15309       commands) shall wrap around the beginning or end of the edit buffer; when unset, searches  
15310       shall stop at the beginning or end of the edit buffer.
- 15311       **writeany, wa**
- 15312       [Default *unset*]
- 15313       If **writeany** is set, some of the checks performed when executing the *ex* **write** commands shall be  
15314       inhibited, as described in editor option **autowrite**.
- 15315       **EXIT STATUS**
- 15316       The following exit values shall be returned:
- 15317       0   Successful completion.
- 15318       >0  An error occurred.
- 15319       **CONSEQUENCES OF ERRORS**
- 15320       When any error is encountered and the standard input is not a terminal device file, *ex* shall not  
15321       write the file or return to command or text input mode, and shall terminate with a non-zero exit  
15322       status.
- 15323       Otherwise, when an unrecoverable error is encountered, it shall be equivalent to a SIGHUP  
15324       asynchronous event.

15325           Otherwise, when an error is encountered, the editor shall behave as specified in **Command Line**  
15326           **Parsing in ex** (on page 2563).

### 15327 APPLICATION USAGE

15328           If a SIGSEGV signal is received while *ex* is saving a file, the file might not be successfully saved.

15329           The **next** command can accept more than one file, so usage such as:

```
15330           next `ls [abc]*`
```

15331           is valid; it would not be valid for the **edit** or **read** commands, for example, because they expect  
15332           only one file and unspecified results occur.

### 15333 EXAMPLES

15334           None.

### 15335 RATIONALE

15336           The *ex/vi* specification is based on the historical practice found in the 4 BSD and System V  
15337           implementations of *ex* and *vi*. A freely redistributable implementation of *ex/vi*, which is  
15338           tracking IEEE Std 1003.1-200x fairly closely, and demonstrates the intended changes between  
15339           historical implementations and IEEE Std 1003.1-200x, may be obtained by anonymous FTP from:

```
15340           ftp://ftp.rdg.opengroup/pub/mirrors/nvi
```

15341           A *restricted editor* (both the historical *red* utility and modifications to *ex*) were considered and  
15342           rejected for inclusion. Neither option provided the level of security that users might expect.

15343           It is recognized that *ex* visual mode and related features would be difficult, if not impossible, to  
15344           implement satisfactorily on a block-mode terminal, or a terminal without any form of cursor  
15345           addressing; thus, it is not a mandatory requirement that such features should work on all  
15346           terminals. It is the intention, however, that an *ex* implementation should provide the full set of  
15347           capabilities on all terminals capable of supporting them.

### 15348 Options

15349           The **-c** replacement for **+command** was inspired by the **-e** option of *sed*. Historically, all such  
15350           commands (see **edit** and **next** as well) were executed from the last line of the edit buffer. This  
15351           meant, for example, that **"/pattern"** would fail unless the **wrapsan** option was set.  
15352           IEEE Std 1003.1-200x requires conformance to historical practice. Historically, some  
15353           implementations restricted the *ex* commands that could be listed as part of the command line  
15354           arguments. For consistency, IEEE Std 1003.1-200x does not permit these restrictions.

15355           In historical implementations of the editor, the **-R** option (and the **readonly** edit option) only  
15356           prevented overwriting of files; appending to files was still permitted, mapping loosely into the  
15357           *cs*h **noclobber** variable. Some implementations, however, have not followed this semantic, and  
15358           **readonly** does not permit appending either. IEEE Std 1003.1-200x follows the latter practice,  
15359           believing that it is a more obvious and intuitive meaning of **readonly**.

15360           The **-s** option suppresses all interactive user feedback and is useful for editing scripts in batch  
15361           jobs. The list of specific effects is historical practice. The terminal type “incapable of supporting  
15362           open and visual modes” has historically been named “dumb”.

15363           The **-t** option was required because the *ctags* utility appears in IEEE Std 1003.1-200x and the  
15364           option is available in all historical implementations of *ex*.

15365           Historically, the *ex* and *vi* utilities accepted a **-x** option, which did encryption based on the  
15366           algorithm found in the historical *crypt* utility. The **-x** option for encryption, and the associated  
15367           *crypt* utility, were omitted because the algorithm used was not specifiable and the export control  
15368           laws of some nations make it difficult to export cryptographic technology. In addition, it did not

15369 historically provide the level of security that users might expect.

### 15370 **Standard Input**

15371 An end-of-file condition is not equivalent to an end-of-file character. A common end-of-file  
15372 character, <control>-D, is historically an *ex* command.

15373 There was no maximum line length in historical implementations of *ex*. Specifically, as it was  
15374 parsed in chunks, the addresses had a different maximum length than the filenames. Further, the  
15375 maximum line buffer size was declared as BUFSIZ, which was different lengths on different  
15376 systems. This version selected the value of {LINE\_MAX} to impose a reasonable restriction on  
15377 portable usage of *ex* and to aid test suite writers in their development of realistic tests that  
15378 exercise this limit.

### 15379 **Input Files**

15380 It was an explicit decision by the standard developers that a <newline> be added to any file  
15381 lacking one. It was believed that this feature of *ex* and *vi* was relied on by users in order to make  
15382 text files lacking a trailing <newline> more portable. It is recognized that this will require a  
15383 user-specified option or extension for implementations that permit *ex* and *vi* to edit files of type  
15384 other than text if such files are not otherwise identified by the system. It was agreed that the  
15385 ability to edit files of arbitrary type can be useful, but it was not considered necessary to  
15386 mandate that an *ex* or *vi* implementation be required to handle files other than text files.

15387 The paragraph in the INPUT FILES section, “By default, ...”, is intended to close a long-standing  
15388 security problem in *ex* and *vi*, that of the “modeline” or “modelines” edit option. This feature  
15389 allows any line in the first or last five lines of the file containing the strings "ex:" or "vi:"  
15390 (and, apparently, "ei:" or "vx:") to be a line containing editor commands, and *ex* interprets all  
15391 the text up to the next ':' or <newline> as a command. Consider the consequences, for  
15392 example, of an unsuspecting user using *ex* or *vi* as the editor when replying to a mail message in  
15393 which a line such as:

```
15394 ex:! rm -rf :
```

15395 appeared in the signature lines. The standard developers believed strongly that an editor should  
15396 not by default interpret any lines of a file. Vendors are strongly urged to delete this feature from  
15397 their implementations of *ex* and *vi*.

### 15398 **Asynchronous Events**

15399 The intention of the phrase “complete write” is that the entire edit buffer be written to stable  
15400 storage. The note regarding temporary files is intended for implementations that use temporary  
15401 files to back edit buffers unnamed by the user.

15402 Historically, SIGQUIT was ignored by *ex*, but was the equivalent of the **Q** command in visual  
15403 mode; that is, it exited visual mode and entered *ex* mode. IEEE Std 1003.1-200x permits, but does  
15404 not require, this behavior. Historically, SIGINT was often used by *vi* users to terminate text  
15405 input mode (<control>-C is often easier to enter than <ESC>). Some implementations of *vi*  
15406 alerted the terminal on this event, and some did not. IEEE Std 1003.1-200x requires that SIGINT  
15407 behave identically to <ESC>, and that the terminal not be alerted.

15408 Historically, suspending the *ex* editor during text input mode was similar to SIGINT, as  
15409 completed lines were retained, but any partial line discarded, and the editor returned to  
15410 command mode. IEEE Std 1003.1-200x is silent on this issue; implementations are encouraged to  
15411 follow historical practice, where possible.

15412 Historically, the *vi* editor did not treat SIGTSTP as an asynchronous event, and it was therefore  
 15413 impossible to suspend the editor in visual text input mode. There are two major reasons for this.  
 15414 The first is that SIGTSTP is a broadcast signal on UNIX systems, and the chain of events where  
 15415 the shell *execs* an application that then *execs vi* usually caused confusion for the terminal state if  
 15416 SIGTSTP was delivered to the process group in the default manner. The second was that most  
 15417 implementations of the UNIX *curses* package are not reentrant, and the receipt of SIGTSTP at the  
 15418 wrong time will cause them to crash. IEEE Std 1003.1-200x is silent on this issue;  
 15419 implementations are encouraged to treat suspension as an asynchronous event if possible.

15420 Historically, modifications to the edit buffer made before SIGINT interrupted an operation were  
 15421 retained; that is, anywhere from zero to all of the lines to be modified might have been modified  
 15422 by the time the SIGINT arrived. These changes were not discarded by the arrival of SIGINT.  
 15423 IEEE Std 1003.1-200x permits this behavior, noting that the *undo* command is required to be able  
 15424 to undo these partially completed commands.

15425 The action taken for signals other than SIGINT, SIGCONT, SIGHUP, and SIGTERM is  
 15426 unspecified because some implementations attempt to save the edit buffer in a useful state when  
 15427 other signals are received.

#### 15428 **Standard Error**

15429 For *ex/vi*, diagnostic messages are those messages reported as a result of a failed attempt to  
 15430 invoke *ex* or *vi*, such as invalid options or insufficient resources, or an abnormal termination  
 15431 condition. Diagnostic messages should not be confused with the error messages generated by  
 15432 inappropriate or illegal user commands.

#### 15433 **Initialization in *ex* and *vi***

15434 If an *ex* command (other than **cd**, **chdir**, or **source**) has a filename argument, one or both of the  
 15435 alternate and current pathnames will be set. Informally, they are set as follows:

- 15436 1. If the *ex* command is one that replaces the contents of the edit buffer, and it succeeds, the  
 15437 current pathname will be set to the filename argument (the first filename argument in the  
 15438 case of the **next** command) and the alternate pathname will be set to the previous current  
 15439 pathname, if there was one.
- 15440 2. In the case of the file read/write forms of the **read** and **write** commands, if there is no  
 15441 current pathname, the current pathname will be set to the filename argument.
- 15442 3. Otherwise, the alternate pathname will be set to the filename argument.

15443 For example, **:edit foo** and **:recover foo**, when successful, set the current pathname, and, if there  
 15444 was a previous current pathname, the alternate pathname. The commands **:write**, **!command**,  
 15445 and **:edit** set neither the current or alternate pathnames. If the **:edit foo** command were to fail for  
 15446 some reason, the alternate pathname would be set. The **read** and **write** commands set the  
 15447 alternate pathname to their *file* argument, unless the current pathname is not set, in which case  
 15448 they set the current pathname to their *file* arguments. The alternate pathname was not  
 15449 historically set by the **:source** command. IEEE Std 1003.1-200x requires conformance to historical  
 15450 practice. Implementations adding commands that take filenames as arguments are encouraged  
 15451 to set the alternate pathname as described here.

15452 Historically, *ex* and *vi* read the **.exrc** file in the *\$HOME* directory twice, if the editor was executed  
 15453 in the *\$HOME* directory. IEEE Std 1003.1-200x prohibits this behavior.

15454 Historically, the 4 BSD *ex* and *vi* read the *\$HOME* and local **.exrc** files if they were owned by the  
 15455 real ID of the user, or the **sourceany** option was set, regardless of other considerations. This was  
 15456 a security problem because it is possible to put normal UNIX system commands inside a **.exrc**

15457 file. IEEE Std 1003.1-200x does not specify the **sourceany** option, and historical implementations  
15458 are encouraged to delete it.

15459 The **.exrc** files must be owned by the real ID of the user, and not writeable by anyone other than  
15460 the owner. The appropriate privileges exception is intended to permit users to acquire special  
15461 privileges, but continue to use the **.exrc** files in their home directories.

15462 System V Release 3.2 and later *vi* implementations added the option **[no]exrc**. The behavior is  
15463 that local **.exrc** files are read-only if the **exrc** option is set. The default for the **exrc** option was off,  
15464 so by default, local **.exrc** files were not read. The problem this was intended to solve was that  
15465 System V permitted users to give away files, so there is no possible ownership or writeability  
15466 test to ensure that the file is safe. This is still a security problem on systems where users can give  
15467 away files, but there is nothing additional that IEEE Std 1003.1-200x can do. The  
15468 implementation-defined exception is intended to permit groups to have local **.exrc** files that are  
15469 shared by users, by creating pseudo-users to own the shared files.

15470 IEEE Std 1003.1-200x does not mention system-wide *ex* and *vi* start-up files. While they exist in  
15471 several implementations of *ex* and *vi*, they are not present in any implementations considered  
15472 historical practice by IEEE Std 1003.1-200x. Implementations that have such files should use  
15473 them only if they are owned by the real user ID or an appropriate user (for example, root on  
15474 UNIX systems) and if they are not writeable by any user other than their owner. System-wide  
15475 start-up files should be read before the *EXINIT* variable, **\$HOME/.exrc** or local **.exrc** files are  
15476 evaluated.

15477 Historically, any *ex* command could be entered in the *EXINIT* variable or the **.exrc** file, although  
15478 ones requiring that the edit buffer already contain lines of text generally caused historical  
15479 implementations of the editor to drop core. IEEE Std 1003.1-200x requires that any *ex* command  
15480 be permitted in the *EXINIT* variable and **.exrc** files, for simplicity of specification and  
15481 consistency, although many of them will obviously fail under many circumstances.

15482 The initialization of the contents of the edit buffer uses the phrase “the effect shall be” with  
15483 regard to various *ex* commands. The intent of this phrase is that edit buffer contents loaded  
15484 during the initialization phase not be lost; that is, loading the edit buffer should fail if the **.exrc**  
15485 file read in the contents of a file and did not subsequently write the edit buffer. An additional  
15486 intent of this phrase is to specify that the initial current line and column is set as specified for the  
15487 individual *ex* commands.

15488 Historically, the **-t** option behaved as if the tag search were a *+command*; that is, it was executed  
15489 from the last line of the file specified by the tag. This resulted in the search failing if the pattern  
15490 was a forward search pattern and the **wrapsan** edit option was not set. IEEE Std 1003.1-200x  
15491 does not permit this behavior, requiring that the search for the tag pattern be performed on the  
15492 entire file, and, if not found, that the current line be set to a more reasonable location in the file.

15493 Historically, the empty edit buffer presented for editing when a file was not specified by the user  
15494 was unnamed. This is permitted by IEEE Std 1003.1-200x; however, implementations are  
15495 encouraged to provide users a temporary filename for this buffer because it permits them the  
15496 use of *ex* commands that use the current pathname during temporary edit sessions.

15497 Historically, the file specified using the **-t** option was not part of the current argument list. This  
15498 practice is permitted by IEEE Std 1003.1-200x; however, implementations are encouraged to  
15499 include its name in the current argument list for consistency.

15500 Historically, the **-c** command was generally not executed until a file that already exists was  
15501 edited. IEEE Std 1003.1-200x requires conformance to this historical practice. Commands that  
15502 could cause the **-c** command to be executed include the *ex* commands **edit**, **next**, **recover**,  
15503 **rewind**, and **tag**, and the *vi* commands **<control>-^** and **<control>-]**. Historically, reading a file  
15504 into an edit buffer did not cause the **-c** command to be executed (even though it might set the

15505 current pathname) with the exception that it did cause the `-c` command to be executed if: the  
 15506 editor was in `ex` mode, the edit buffer had no current pathname, the edit buffer was empty, and  
 15507 no read commands had yet been attempted. For consistency and simplicity of specification,  
 15508 IEEE Std 1003.1-200x does not permit this behavior.

15509 Historically, the `-r` option was the same as a normal edit session if there was no recovery  
 15510 information available for the file. This allowed users to enter:

```
15511 vi -r *.c
```

15512 and recover whatever files were recoverable. In some implementations, recovery was attempted  
 15513 only on the first file named, and the file was not entered into the argument list; in others,  
 15514 recovery was attempted for each file named. In addition, some historical implementations  
 15515 ignored `-r` if `-t` was specified or did not support command line *file* arguments with the `-t` option.  
 15516 For consistency and simplicity of specification, IEEE Std 1003.1-200x disallows these special  
 15517 cases, and requires that recovery be attempted the first time each file is edited.

15518 Historically, `vi` initialized the `'` and `'` marks, but `ex` did not. This meant that if the first command  
 15519 in `ex` mode was **visual** or if an `ex` command was executed first (for example, `vi +10 file`), `vi` was  
 15520 entered without the marks being initialized. Because the standard developers believed the marks  
 15521 to be generally useful, and for consistency and simplicity of specification, IEEE Std 1003.1-200x  
 15522 requires that they always be initialized if in open or visual mode, or if in `ex` mode and the edit  
 15523 buffer is not empty. Not initializing it in `ex` mode if the edit buffer is empty is historical practice;  
 15524 however, it has always been possible to set (and use) marks in empty edit buffers in open and  
 15525 visual mode edit sessions.

## 15526 Addressing

15527 Historically, `ex` and `vi` accepted the additional addressing forms `'\/'` and `'\?'`. They were  
 15528 equivalent to `"//"` and `"??"`, respectively. They are not required by IEEE Std 1003.1-200x,  
 15529 mostly because nobody can remember whether they ever did anything different historically.

15530 Historically, `ex` and `vi` permitted an address of zero for several commands, and permitted the `%`  
 15531 address in empty files for others. For consistency, IEEE Std 1003.1-200x requires support for the  
 15532 former in the few commands where it makes sense, and disallows it otherwise. In addition,  
 15533 because IEEE Std 1003.1-200x requires that `%` be logically equivalent to `"1,$"`, it is also  
 15534 supported where it makes sense and disallowed otherwise.

15535 Historically, the `%` address could not be followed by further addresses. For consistency and  
 15536 simplicity of specification, IEEE Std 1003.1-200x requires that additional addresses be supported.

15537 All of the following are valid *addresses*:

```
15538 +++          Three lines after the current line.
```

```
15539 /re/-        One line before the next occurrence of re.
```

```
15540 -2           Two lines before the current line.
```

```
15541 3 ---- 2    Line one (note intermediate negative address).
```

```
15542 1 2 3       Line six.
```

15543 Any number of addresses can be provided to commands taking addresses; for example,  
 15544 `"1,2,3,4,5p"` prints lines 4 and 5, because two is the greatest valid number of addresses  
 15545 accepted by the **print** command. This, in combination with the semicolon delimiter, permits  
 15546 users to create commands based on ordered patterns in the file. For example, the command  
 15547 **3;/foo/;+2print** will display the first line after line 3 that contains the pattern `foo`, plus the next  
 15548 two lines. Note that the address `3`; must be evaluated before being discarded because the search

15549 origin for the **/foo/** command depends on this.

15550 Historically, values could be added to addresses by including them after one or more <blank>;  
 15551 for example, **3 – 5p** wrote the seventh line of the file, and **/foo/ 5** was the same as **/foo/+5**.  
 15552 However, only absolute values could be added; for example, **5 /foo/** was an error.  
 15553 IEEE Std 1003.1-200x requires conformance to historical practice. Address offsets are separately  
 15554 specified from addresses because they could historically be provided to visual mode search  
 15555 commands.

15556 Historically, any missing addresses defaulted to the current line. This was true for leading and  
 15557 trailing comma-delimited addresses, and for trailing semicolon-delimited addresses. For  
 15558 consistency, IEEE Std 1003.1-200x requires it for leading semicolon addresses as well.

15559 Historically, *ex* and *vi* accepted the '^' character as both an address and as a flag offset for  
 15560 commands. In both cases it was identical to the '-' character. IEEE Std 1003.1-200x does not  
 15561 require or prohibit this behavior.

15562 Historically, the enhancements to basic regular expressions could be used in addressing; for  
 15563 example, '~', '\<', and '\>'. IEEE Std 1003.1-200x requires conformance to historical  
 15564 practice; that is, that regular expression usage be consistent, and that regular expression  
 15565 enhancements be supported wherever regular expressions are used.

### 15566 **Command Line Parsing in ex**

15567 Historical *ex* command parsing was even more complex than that described here.  
 15568 IEEE Std 1003.1-200x requires the subset of the command parsing that the standard developers  
 15569 believed was documented and that users could reasonably be expected to use in a portable  
 15570 fashion, and that was historically consistent between implementations. (The discarded  
 15571 functionality is obscure, at best.) Historical implementations will require changes in order to  
 15572 comply with IEEE Std 1003.1-200x; however, users are not expected to notice any of these  
 15573 changes. Most of the complexity in *ex* parsing is to handle three special termination cases:

- 15574 1. The **!**, **global**, **v**, and the filter versions of the **read** and **write** commands are delimited by  
 15575 <newline>s (they can contain vertical-line characters that are usually shell pipes).
- 15576 2. The **ex**, **edit**, **next**, and **visual** in open and visual mode commands all take *ex* commands,  
 15577 optionally containing vertical-line characters, as their first arguments.
- 15578 3. The **s** command takes a regular expression as its first argument, and uses the delimiting  
 15579 characters to delimit the command.

15580 Historically, vertical-line characters in the *+command* argument of the **ex**, **edit**, **next**, **vi**, and  
 15581 **visual** commands, and in the *pattern* and *replacement* parts of the **s** command, did not delimit the  
 15582 command, and in the filter cases for **read** and **write**, and the **!**, **global**, and **v** commands, they did  
 15583 not delimit the command at all. For example, the following commands are all valid:

```
15584 :edit +25 | s/abc/ABC/ file.c
15585 :s/ | /PIPE/
15586 :read !spell % | columnate
15587 :global/pattern/p | l
15588 :s/a/b/ | s/c/d | set
```

15589 Historically, empty or <blank> filled lines in **.exrc** files and **sourced** files (as well as *EXINIT*  
 15590 variables and *ex* command scripts) were treated as default commands; that is, **print** commands.  
 15591 IEEE Std 1003.1-200x specifically requires that they be ignored when encountered in **.exrc** and  
 15592 **sourced** files to eliminate a common source of new user error.

15593 Historically, *ex* commands with multiple adjacent (or <blank>-separated) vertical lines were  
15594 handled oddly when executed from *ex* mode. For example, the command | | | <carriage-return>,  
15595 when the cursor was on line 1, displayed lines 2, 3, and 5 of the file. In addition, the command |  
15596 would only display the line after the next line, instead of the next two lines. The former worked  
15597 more logically when executed from *vi* mode, and displayed lines 2, 3, and 4.  
15598 IEEE Std 1003.1-200x requires the *vi* behavior; that is, a single default command and line number  
15599 increment for each command separator, and trailing <newline>s after vertical-line separators are  
15600 discarded.

15601 Historically, *ex* permitted a single extra colon as a leading command character; for example,  
15602 **:g/pattern/:p** was a valid command. IEEE Std 1003.1-200x generalizes this to require that any  
15603 number of leading colon characters be stripped.

15604 Historically, any prefix of the **delete** command could be followed without intervening <blank>s  
15605 by a flag character because in the command **d p**, *p* is interpreted as the buffer *p*.  
15606 IEEE Std 1003.1-200x requires conformance to historical practice.

15607 Historically, the **k** command could be followed by the mark name without intervening  
15608 <blank>s. IEEE Std 1003.1-200x requires conformance to historical practice.

15609 Historically, the **s** command could be immediately followed by flag and option characters; for  
15610 example, **s/e/E/|s|sgc3p** was a valid command. However, flag characters could not stand alone;  
15611 for example, the commands **sp** and **s l** would fail, while the command **sgp** and **s gl** would  
15612 succeed. (Obviously, the '#' flag character was used as a delimiter character if it followed the  
15613 command.) Another issue was that option characters had to precede flag characters even when  
15614 the command was fully specified; for example, the command **s/e/E/pg** would fail, while the  
15615 command **s/e/E/gp** would succeed. IEEE Std 1003.1-200x requires conformance to historical  
15616 practice.

15617 Historically, the first command name that had a prefix matching the input from the user was the  
15618 executed command; for example, **ve**, **ver**, and **vers** all executed the **version** command.  
15619 Commands were in a specific order, however, so that **a** matched **append**, not **abbreviate**.  
15620 IEEE Std 1003.1-200x requires conformance to historical practice. The restriction on command  
15621 search order for implementations with extensions is to avoid the addition of commands such  
15622 that the historical prefixes would fail to work portably.

15623 Historical implementations of *ex* and *vi* did not correctly handle multiple *ex* commands,  
15624 separated by vertical-line characters, that entered or exited visual mode or the editor. Because  
15625 implementations of *vi* exist that do not exhibit this failure mode, IEEE Std 1003.1-200x does not  
15626 permit it.

15627 The requirement that alphabetic command names consist of all following alphabetic characters  
15628 up to the next non-alphabetic character means that alphabetic command names must be  
15629 separated from their arguments by one or more non-alphabetic characters, normally a <blank>  
15630 or '!' character, except as specified for the exceptions, the **delete**, **k**, and **s** commands.

15631 Historically, the repeated execution of the *ex* default **print** commands (<control>-D, *eof*,  
15632 <newline>, <carriage-return>) erased any prompting character and displayed the next lines  
15633 without scrolling the terminal; that is, immediately below any previously displayed lines. This  
15634 provided a cleaner presentation of the lines in the file for the user. IEEE Std 1003.1-200x does not  
15635 require this behavior because it may be impossible in some situations; however,  
15636 implementations are strongly encouraged to provide this semantic if possible.

15637 Historically, it was possible to change files in the middle of a command, and have the rest of the  
15638 command executed in the new file; for example:

15639 :edit +25 file.c | s/abc/ABC/ | 1

15640 was a valid command, and the substitution was attempted in the newly edited file.  
 15641 IEEE Std 1003.1-200x requires conformance to historical practice. The following commands are  
 15642 examples that exercise the *ex* parser:

15643 echo 'foo | bar' > file1; echo 'foo/bar' > file2;

15644 vi

15645 :edit +1 | s/|/PIPE/ | w file1 | e file2 | 1 | s/\/SLASH/ | wq

15646 Historically, there was no protection in editor implementations to avoid *ex* **global**, **v**, **@**, or **\***  
 15647 commands changing edit buffers during execution of their associated commands. Because this  
 15648 would almost invariably result in catastrophic failure of the editor, and implementations exist  
 15649 that do exhibit these problems, IEEE Std 1003.1-200x requires that changing the edit buffer  
 15650 during a **global** or **v** command, or during a **@** or **\*** command for which there will be more than a  
 15651 single execution, be an error. Implementations supporting multiple edit buffers simultaneously  
 15652 are strongly encouraged to apply the same semantics to switching between buffers as well.

15653 The *ex* command quoting required by IEEE Std 1003.1-200x is a superset of the quoting in  
 15654 historical implementations of the editor. For example, it was not historically possible to escape a  
 15655 <blank> in a filename; for example, **:edit foo\\\ bar** would report that too many filenames had  
 15656 been entered for the edit command, and there was no method of escaping a <blank> in the first  
 15657 argument of an **edit**, **ex**, **next**, or **visual** command at all. IEEE Std 1003.1-200x extends historical  
 15658 practice, requiring that quoting behavior be made consistent across all *ex* commands, except for  
 15659 the **map**, **unmap**, **abbreviate**, and **unabbreviate** commands, which historically used <control>-V  
 15660 instead of backslashes for quoting. For those four commands, IEEE Std 1003.1-200x requires  
 15661 conformance to historical practice.

15662 Backslash quoting in *ex* is non-intuitive. Backslash escapes are ignored unless they escape a  
 15663 special character; for example, when performing *file* argument expansion, the string "\\%" is  
 15664 equivalent to '\%', not "\<current path name>". This can be confusing for users because  
 15665 backslash is usually one of the characters that causes shell expansion to be performed, and  
 15666 therefore shell quoting rules must be taken into consideration. Generally, quoting characters are  
 15667 only considered if they escape a special character, and a quoting character must be provided for  
 15668 each layer of parsing for which the character is special. As another example, only a single  
 15669 backslash is necessary for the '\1' sequence in substitute replacement patterns, because the  
 15670 character '1' is not special to any parsing layer above it.

15671 <control>-V quoting in *ex* is slightly different from backslash quoting. In the four commands  
 15672 where <control>-V quoting applies (**abbreviate**, **unabbreviate**, **map**, and **unmap**), any character  
 15673 may be escaped by a <control>-V whether it would have a special meaning or not.  
 15674 IEEE Std 1003.1-200x requires conformance to historical practice.

15675 Historical implementations of the editor did not require delimiters within character classes to be  
 15676 escaped; for example, the command **:s/[//** on the string "xxx/yyy" would delete the '/' from  
 15677 the string. IEEE Std 1003.1-200x disallows this historical practice for consistency and because it  
 15678 places a large burden on implementations by requiring that knowledge of regular expressions be  
 15679 built into the editor parser.

15680 Historically, quoting <newline>s in *ex* commands was handled inconsistently. In most cases, the  
 15681 <newline> always terminated the command, regardless of any preceding escape character,  
 15682 because backslash characters did not escape <newline>s for most *ex* commands. However, some  
 15683 *ex* commands (for example, **s**, **map**, and **abbreviation**) permitted <newline>s to be escaped  
 15684 (although in the case of **map** and **abbreviation**, <control>-V characters escaped them instead of  
 15685 backslashes). This was true in not only the command line, but also **.exrc** and **sourced** files. For  
 15686 example, the command:

15687 map = foo<control-V><newline>bar

15688 would succeed, although it was sometimes difficult to get the <control>-V and the inserted  
15689 <newline> passed to the *ex* parser. For consistency and simplicity of specification,  
15690 IEEE Std 1003.1-200x requires that it be possible to escape <newline>s in *ex* commands at all  
15691 times, using backslashes for most *ex* commands, and using <control>-V characters for the **map**  
15692 and **abbreviation** commands. For example, the command **print<newline>list** is required to be  
15693 parsed as the single command **print<newline>list**. While this differs from historical practice,  
15694 IEEE Std 1003.1-200x developers believed it unlikely that any script or user depended on the  
15695 historical behavior.

15696 Historically, an error in a command specified using the **-c** option did not cause the rest of the **-c**  
15697 commands to be discarded. IEEE Std 1003.1-200x disallows this for consistency with mapped  
15698 keys, the **@**, **global**, **source**, and **v** commands, the *EXINIT* environment variable, and the **.exrc**  
15699 files.

### 15700 **Input Editing in ex**

15701 One of the common uses of the historical *ex* editor is over slow network connections. Editors  
15702 that run in canonical mode can require far less traffic to and from, and far less processing on, the  
15703 host machine, as well as more easily supporting block-mode terminals. For these reasons,  
15704 IEEE Std 1003.1-200x requires that *ex* be implemented using canonical mode input processing, as  
15705 was done historically.

15706 IEEE Std 1003.1-200x does not require the historical 4 BSD input editing characters “word erase”  
15707 or “literal next”. For this reason, it is unspecified how they are handled by *ex*, although they  
15708 must have the required effect. Implementations that resolve them after the line has been ended  
15709 using a <newline> or <control>-M character, and implementations that rely on the underlying  
15710 system terminal support for this processing, are both conforming. Implementations are strongly  
15711 urged to use the underlying system functionality, if at all possible, for compatibility with other  
15712 system text input interfaces.

15713 Historically, when the *eof* character was used to decrement the **autoindent** level, the cursor  
15714 moved to display the new end of the **autoindent** characters, but did not move the cursor to a  
15715 new line, nor did it erase the <control>-D character from the line. IEEE Std 1003.1-200x does not  
15716 specify that the cursor remain on the same line or that the rest of the line is erased; however,  
15717 implementations are strongly encouraged to provide the best possible user interface; that is, the  
15718 cursor should remain on the same line, and any <control>-D character on the line should be  
15719 erased.

15720 IEEE Std 1003.1-200x does not require the historical 4 BSD input editing character “reprint”,  
15721 traditionally <control>-R, which redisplayed the current input from the user. For this reason,  
15722 and because the functionality cannot be implemented after the line has been terminated by the  
15723 user, IEEE Std 1003.1-200x makes no requirements about this functionality. Implementations are  
15724 strongly urged to make this historical functionality available, if possible.

15725 Historically, <control>-Q did not perform a literal next function in *ex*, as it did in *vi*.  
15726 IEEE Std 1003.1-200x requires conformance to historical practice to avoid breaking historical *ex*  
15727 scripts and **.exrc** files.

15728

**eof**

15729

15730

15731

15732

Whether the *eof* character immediately modifies the **autoindent** characters in the prompt is left unspecified so that implementations can conform in the presence of systems that do not support this functionality. Implementations are encouraged to modify the line and redisplay it immediately, if possible.

15733

15734

15735

The specification of the handling of the *eof* character differs from historical practice only in that *eof* characters are not discarded if they follow normal characters in the text input. Historically, they were always discarded.

15736

**Command Descriptions in ex**

15737

15738

15739

15740

15741

15742

15743

Historically, several commands (for example, **global**, **v**, **visual**, **s**, **write**, **wq**, **yank**, **!**, **<**, **>**, **&**, and **-**) were executable in empty files (that is, the default address(es) were 0), or permitted explicit addresses of 0 (for example, 0 was a valid address, or 0,0 was a valid range). Addresses of 0, or command execution in an empty file, make sense only for commands that add new text to the edit buffer or write commands (because users may wish to write empty files). IEEE Std 1003.1-200x requires this behavior for such commands and disallows it otherwise, for consistency and simplicity of specification.

15744

15745

15746

A count to an *ex* command has been historically corrected to be no greater than the last line in a file; for example, in a five-line file, the command **1,6print** would fail, but the command **1print300** would succeed. IEEE Std 1003.1-200x requires conformance to historical practice.

15747

15748

15749

15750

15751

15752

15753

15754

15755

Historically, the use of flags in *ex* commands could be obscure. General historical practice was as described by IEEE Std 1003.1-200x, but there were some special cases. For example, the **list**, **number**, and **print** commands ignored trailing address offsets; for example, **3p +++#** would display line 3, and 3 would be the current line after the execution of the command. The **open** and **visual** commands ignored both the trailing offsets and the trailing flags. Also, flags specified to the **open** and **visual** commands interacted badly with the **list** edit option, and setting and then unsetting it during the open/visual session would cause *vi* to stop displaying lines in the specified format. For consistency and simplicity of specification, IEEE Std 1003.1-200x does not permit any of these exceptions to the general rule.

15756

15757

IEEE Std 1003.1-200x uses the word *copy* in several places when discussing buffers. This is not intended to imply implementation.

15758

15759

15760

15761

Historically, *ex* users could not specify numeric buffers because of the ambiguity this would cause; for example, in the command **3 delete 2**, it is unclear whether 2 is a buffer name or a *count*. IEEE Std 1003.1-200x requires conformance to historical practice by default, but does not preclude extensions.

15762

15763

15764

Historically, the contents of the unnamed buffer were frequently discarded after commands that did not explicitly affect it; for example, when using the **edit** command to switch files. For consistency and simplicity of specification, IEEE Std 1003.1-200x does not permit this behavior.

15765

15766

15767

15768

15769

15770

The *ex* utility did not historically have access to the numeric buffers, and, furthermore, deleting lines in *ex* did not modify their contents. For example, if, after doing a delete in *vi*, the user switched to *ex*, did another delete, and then switched back to *vi*, the contents of the numeric buffers would not have changed. IEEE Std 1003.1-200x requires conformance to historical practice. Numeric buffers are described in the *ex* utility in order to confine the description of buffers to a single location in IEEE Std 1003.1-200x.

15771

15772

15773

The metacharacters that trigger shell expansion in *file* arguments match historical practice, as does the method for doing shell expansion. Implementations wishing to provide users with the flexibility to alter the set of metacharacters are encouraged to provide a **shellmeta** string edit

15774 option.

15775 Historically, `ex` commands executed from `vi` refreshed the screen when it did not strictly need to  
 15776 do so; for example, `!date > /dev/null` does not require a screen refresh because the output of the  
 15777 UNIX `date` command requires only a single line of the screen. IEEE Std 1003.1-200x requires that  
 15778 the screen be refreshed if it has been overwritten, but makes no requirements as to how an  
 15779 implementation should make that determination. Implementations may prompt and refresh the  
 15780 screen regardless.

### 15781 **Abbreviate**

15782 Historical practice was that characters that were entered as part of an abbreviation replacement  
 15783 were subject to **map** expansions, the **showmatch** edit option, further abbreviation expansions,  
 15784 and so on; that is, they were logically pushed onto the terminal input queue, and were not a  
 15785 simple replacement. IEEE Std 1003.1-200x requires conformance to historical practice. Historical  
 15786 practice was that whenever a non-word character (that had not been escaped by a `<control>-V`)  
 15787 was entered after a word character, `vi` would check for abbreviations. The check was based on  
 15788 the type of the character entered before the word character of the word/non-word pair that  
 15789 triggered the check. The word character of the word/non-word pair that triggered the check and  
 15790 all characters entered before the trigger pair that were of that type were included in the check,  
 15791 with the exception of `<blank>s`, which always delimited the abbreviation.

15792 This means that, for the abbreviation to work, the *lhs* must end with a word character, there can  
 15793 be no transitions from word to non-word characters (or *vice versa*) other than between the last  
 15794 and next-to-last characters in the *lhs*, and there can be no `<blank>s` in the *lhs*. In addition,  
 15795 because of the historical quoting rules, it was impossible to enter a literal `<control>-V` in the *lhs*.  
 15796 IEEE Std 1003.1-200x requires conformance to historical practice. Historical implementations did  
 15797 not inform users when abbreviations that could never be used were entered; implementations  
 15798 are strongly encouraged to do so.

15799 For example, the following abbreviations will work:

```
15800 :ab (p REPLACE
15801 :ab p REPLACE
15802 :ab ( (p REPLACE
```

15803 The following abbreviations will not work:

```
15804 :ab ( REPLACE
15805 :ab (pp REPLACE
```

15806 Historical practice is that words on the `vi` colon command line were subject to abbreviation  
 15807 expansion, including the arguments to the **abbrev** (and more interestingly) the **unabbrev**  
 15808 command. Because there are implementations that do not do abbreviation expansion for the first  
 15809 argument to those commands, this is permitted, but not required, by IEEE Std 1003.1-200x.  
 15810 However, the following sequence:

```
15811 :ab foo bar
15812 :ab foo baz
```

15813 resulted in the addition of an abbreviation of `"baz"` for the string `"bar"` in historical `ex/vi`, and  
 15814 the sequence:

```
15815 :ab foo1 bar
15816 :ab foo2 bar
15817 :unabbreviate foo2
```

15818 deleted the abbreviation "foo1", not "foo2". These behaviors are not permitted by  
15819 IEEE Std 1003.1-200x because they clearly violate the expectations of the user.

15820 It was historical practice that <control>-V, not backslash, characters be interpreted as escaping  
15821 subsequent characters in the **abbreviate** command. IEEE Std 1003.1-200x requires conformance  
15822 to historical practice; however, it should be noted that an abbreviation containing a <blank> will  
15823 never work.

## 15824 **Append**

15825 Historically, any text following a vertical-line command separator after an **append**, **change**, or  
15826 **insert** command became part of the insert text. For example, in the command:

```
15827 :g/pattern/append|stuff1
```

15828 a line containing the text "stuff1" would be appended to each line matching pattern. It was  
15829 also historically valid to enter:

```
15830 :append|stuff1
```

```
15831 stuff2
```

```
15832 .
```

15833 and the text on the *ex* command line would be appended along with the text inserted after it.  
15834 There was an historical bug, however, that the user had to enter two terminating lines (the ' .' lines)  
15835 to terminate text input mode in this case. IEEE Std 1003.1-200x requires conformance to  
15836 historical practice, but disallows the historical need for multiple terminating lines.

## 15837 **Change**

15838 See the RATIONALE for the **append** command. Historical practice for cursor positioning after  
15839 the change command when no text is input, is as described in IEEE Std 1003.1-200x. However,  
15840 one System V implementation is known to have been modified such that the cursor is positioned  
15841 on the first address specified, and not on the line before the first address. IEEE Std 1003.1-200x  
15842 disallows this modification for consistency.

15843 Historically, the **change** command did not support buffer arguments, although some  
15844 implementations allow the specification of an optional buffer. This behavior is neither required  
15845 nor disallowed by IEEE Std 1003.1-200x.

## 15846 **Change Directory**

15847 A common extension in *ex* implementations is to use the elements of a **cdpath** edit option as  
15848 prefix directories for *path* arguments to **chdir** that are relative pathnames and that do not have  
15849 ' .' or " ." as their first component. Elements in the **cdpath** edit option are colon-separated.  
15850 The initial value of the **cdpath** edit option is the value of the shell *CDPATH* environment  
15851 variable. This feature was not included in IEEE Std 1003.1-200x because it does not exist in any  
15852 of the implementations considered historical practice.

## 15853 **Copy**

15854 Historical implementations of *ex* permitted copies to lines inside of the specified range; for  
15855 example, **:2,5copy3** was a valid command. IEEE Std 1003.1-200x requires conformance to  
15856 historical practice.

- 15857           **Delete**
- 15858           IEEE Std 1003.1-200x requires support for the historical parsing of a **delete** command followed  
15859           by flags, without any intervening <blank>s. For example:
- 15860           **1dp**     Deletes the first line and prints the line that was second.
- 15861           **1delep** As for **1dp**.
- 15862           **1d**     Deletes the first line, saving it in buffer *p*.
- 15863           **1d p11** (Pee-one-ell.) Deletes the first line, saving it in buffer *p*, and listing the line that was  
15864           second.
- 15865           **Edit**
- 15866           Historically, any *ex* command could be entered as a *+command* argument to the **edit** command,  
15867           although some (for example, **insert** and **append**) were known to confuse historical  
15868           implementations. For consistency and simplicity of specification, IEEE Std 1003.1-200x requires  
15869           that any command be supported as an argument to the **edit** command.
- 15870           Historically, the command argument was executed with the current line set to the last line of the  
15871           file, regardless of whether the **edit** command was executed from visual mode or not.  
15872           IEEE Std 1003.1-200x requires conformance to historical practice.
- 15873           Historically, the *+command* specified to the **edit** and **next** commands was delimited by the first  
15874           <blank>, and there was no way to quote them. For consistency, IEEE Std 1003.1-200x requires  
15875           that the usual *ex* backslash quoting be provided.
- 15876           Historically, specifying the *+command* argument to the edit command required a filename to be  
15877           specified as well; for example, **:edit +100** would always fail. For consistency and simplicity of  
15878           specification, IEEE Std 1003.1-200x does not permit this usage to fail for that reason.
- 15879           Historically, only the cursor position of the last file edited was remembered by the editor.  
15880           IEEE Std 1003.1-200x requires that this be supported; however, implementations are permitted  
15881           to remember and restore the cursor position for any file previously edited.
- 15882           **File**
- 15883           Historical versions of the *ex* editor **file** command displayed a current line and number of lines in  
15884           the edit buffer of 0 when the file was empty, while the *vi* <control>-G command displayed a  
15885           current line and number of lines in the edit buffer of 1 in the same situation.  
15886           IEEE Std 1003.1-200x does not permit this discrepancy, instead requiring that a message be  
15887           displayed indicating that the file is empty.
- 15888           **Global**
- 15889           The two-pass operation of the **global** and **v** commands is not intended to imply implementation,  
15890           only the required result of the operation.
- 15891           The current line and column are set as specified for the individual *ex* commands. This  
15892           requirement is cumulative; that is, the current line and column must track across all the  
15893           commands executed by the **global** or **v** commands.

15894 **Insert**

15895 See the RATIONALE for the **append** command.

15896 Historically, **insert** could not be used with an address of zero; that is, not when the edit buffer  
15897 was empty. IEEE Std 1003.1-200x requires that this command behave consistently with the  
15898 **append** command.

15899 **Join**

15900 The action of the **join** command in relation to the special characters is only defined for the  
15901 POSIX locale because the correct amount of white space after a period varies; in Japanese none is  
15902 required, in French only a single space, and so on.

15903 **List**

15904 The historical output of the **list** command was potentially ambiguous. The standard developers  
15905 believed correcting this to be more important than adhering to historical practice, and  
15906 IEEE Std 1003.1-200x requires unambiguous output.

15907 **Map**

15908 Historically, command mode maps only applied to command names; for example, if the  
15909 character 'x' was mapped to 'y', the command **fx** searched for the 'x' character, not the 'y'  
15910 character. IEEE Std 1003.1-200x requires this behavior. Historically, entering <control>-V as the  
15911 first character of a *vi* command was an error. Several implementations have extended the  
15912 semantics of *vi* such that <control>-V means that the subsequent command character is not  
15913 mapped. This is permitted, but not required, by IEEE Std 1003.1-200x. Regardless, using  
15914 <control>-V to escape the second or later character in a sequence of characters that might match  
15915 a **map** command, or any character in text input mode, is historical practice, and stops the entered  
15916 keys from matching a map. IEEE Std 1003.1-200x requires conformance to historical practice.

15917 Historical implementations permitted digits to be used as a **map** command *lhs*, but then ignored  
15918 the map. IEEE Std 1003.1-200x requires that the mapped digits not be ignored.

15919 The historical implementation of the **map** command did not permit **map** commands that were  
15920 more than a single character in length if the first character was printable. This behavior is  
15921 permitted, but not required, by IEEE Std 1003.1-200x.

15922 Historically, mapped characters were remapped unless the **remap** edit option was not set, or the  
15923 prefix of the mapped characters matched the mapping characters; for example, in the **map**:

15924 `:map ab abcd`

15925 the characters "ab" were used as is and were not remapped, but the characters "cd" were  
15926 mapped if appropriate. This can cause infinite loops in the *vi* mapping mechanisms.  
15927 IEEE Std 1003.1-200x requires conformance to historical practice, and that such loops be  
15928 interruptible.

15929 Text input maps had the same problems with expanding the *lhs* for the **ex map!** and **unmap!**  
15930 command as did the **ex abbreviate** and **unabbreviate** commands. See the RATIONALE for the **ex**  
15931 **abbreviate** command. IEEE Std 1003.1-200x requires similar modification of some historical  
15932 practice for the **map** and **unmap** commands, as described for the **abbreviate** and **unabbreviate**  
15933 commands.

15934 Historically, **maps** that were subsets of other **maps** behaved differently depending on the order  
15935 in which they were defined. For example:

```
15936      :map! ab      short
15937      :map! abc     long
```

15938 would always translate the characters "ab" to "short", regardless of how fast the characters  
15939 "abc" were entered. If the entry order was reversed:

```
15940      :map! abc     long
15941      :map! ab      short
```

15942 the characters "ab" would cause the editor to pause, waiting for the completing 'c' character,  
15943 and the characters might never be mapped to "short". For consistency and simplicity of  
15944 specification, IEEE Std 1003.1-200x requires that the shortest match be used at all times.

15945 The length of time the editor spends waiting for the characters to complete the *lhs* is unspecified  
15946 because the timing capabilities of systems are often inexact and variable, and it may depend on  
15947 other factors such as the speed of the connection. The time should be long enough for the user to  
15948 be able to complete the sequence, but not long enough for the user to have to wait. Some  
15949 implementations of *vi* have added a **keytime** option, which permits users to set the number of  
15950 0,1 seconds the editor waits for the completing characters. Because mapped terminal function  
15951 and cursor keys tend to start with an <ESC> character, and <ESC> is the key ending *vi* text input  
15952 mode, **maps** starting with <ESC> characters are generally exempted from this timeout period,  
15953 or, at least timed out differently.

#### 15954 **Mark**

15955 Historically, users were able to set the "previous context" marks explicitly. In addition, the *ex*  
15956 commands ' and ' and the *vi* commands ', ', ', and ' all referred to the same mark. In addition,  
15957 the previous context marks were not set if the command, with which the address setting the  
15958 mark was associated, failed. IEEE Std 1003.1-200x requires conformance to historical practice.  
15959 Historically, if marked lines were deleted, the mark was also deleted, but would reappear if the  
15960 change was undone. IEEE Std 1003.1-200x requires conformance to historical practice.

15961 The description of the special events that set the ' and ' marks matches historical practice. For  
15962 example, historically the command */a,/b/* did not set the ' and ' marks, but the command  
15963 */a,/b/delete* did.

#### 15964 **Next**

15965 Historically, any *ex* command could be entered as a *+command* argument to the **next** command,  
15966 although some (for example, **insert** and **append**) were known to confuse historical  
15967 implementations. IEEE Std 1003.1-200x requires that any command be permitted and that it  
15968 behave as specified. The **next** command can accept more than one file, so usage such as:

```
15969      next `ls [abc] `
```

15970 is valid; it need not be valid for the **edit** or **read** commands, for example, because they expect  
15971 only one filename.

15972 Historically, the **next** command behaved differently from the **rewind** command in that it  
15973 ignored the force flag if the **autowrite** flag was set. For consistency, IEEE Std 1003.1-200x does  
15974 not permit this behavior.

15975 Historically, the **next** command positioned the cursor as if the file had never been edited before,  
15976 regardless. IEEE Std 1003.1-200x does not permit this behavior, for consistency with the **edit**  
15977 command.

15978 Implementations wanting to provide a counterpart to the **next** command that edited the  
15979 previous file have used the command **prev[ious]**, which takes no *file* argument.

15980 IEEE Std 1003.1-200x does not require this command.

### 15981 **Open**

15982 Historically, the **open** command would fail if the **open** edit option was not set.  
 15983 IEEE Std 1003.1-200x does not mention the **open** edit option and does not require this behavior.  
 15984 Some historical implementations do not permit entering open mode from open or visual mode,  
 15985 only from *ex* mode. For consistency, IEEE Std 1003.1-200x does not permit this behavior.

15986 Historically, entering open mode from the command line (that is, *vi +open*) resulted in  
 15987 anomalous behaviors; for example, the *ex* file and *set* commands, and the *vi* command  
 15988 <control>-G did not work. For consistency, IEEE Std 1003.1-200x does not permit this behavior.

15989 Historically, the **open** command only permitted ' / ' characters to be used as the search pattern  
 15990 delimiter. For consistency, IEEE Std 1003.1-200x requires that the search delimiters used by the **s**,  
 15991 **global**, and **v** commands be accepted as well.

### 15992 **Preserve**

15993 The **preserve** command does not historically cause the file to be considered unmodified for the  
 15994 purposes of future commands that may exit the editor. IEEE Std 1003.1-200x requires  
 15995 conformance to historical practice.

15996 Historical documentation stated that mail was not sent to the user when preserve was executed;  
 15997 however, historical implementations did send mail in this case. IEEE Std 1003.1-200x requires  
 15998 conformance to the historical implementations.

### 15999 **Print**

16000 The writing of NUL by the **print** command is not specified as a special case because the standard  
 16001 developers did not want to require *ex* to support NUL characters. Historically, characters were  
 16002 displayed using the ARPA standard mappings, which are as follows:

- 16003 1. Printable characters are left alone.
- 16004 2. Control characters less than \177 are represented as '^' followed by the character offset  
 16005 from the '@' character in the ASCII map; for example, \007 is represented as '^G'.
- 16006 3. \177 is represented as '^?' followed by '? '.

16007 The display of characters having their eighth bit set was less standard. Existing implementations  
 16008 use hex (0x00), octal (\000), and a meta-bit display. (The latter displayed bytes that had their  
 16009 eighth bit set as the two characters "M-" followed by the seven-bit display as described above.)  
 16010 The latter probably has the best claim to historical practice because it was used for the -v option  
 16011 of 4 BSD and 4 BSD-derived versions of the *cat* utility since 1980.

16012 No specific display format is required by IEEE Std 1003.1-200x.

16013 Explicit dependence on the ASCII character set has been avoided where possible, hence the use  
 16014 of the phrase an "implementation-defined multi-character sequence" for the display of non-  
 16015 printable characters in preference to the historical usage of, for instance, "^I" for the <tab>.  
 16016 Implementations are encouraged to conform to historical practice in the absence of any strong  
 16017 reason to diverge.

16018 Historically, all *ex* commands beginning with the letter 'p' could be entered using capitalized  
 16019 versions of the commands; for example, **P[rint]**, **Pre[serve]**, and **Pu[t]** were all valid command  
 16020 names. IEEE Std 1003.1-200x permits, but does not require, this historical practice because  
 16021 capital forms of the commands are used by some implementations for other purposes.

16022 **Put**

16023 Historically, an **ex put** command, executed from open or visual mode, was the same as the open  
16024 or visual mode **P** command, if the buffer was named and was cut in character mode, and the  
16025 same as the **p** command if the buffer was named and cut in line mode. If the unnamed buffer  
16026 was the source of the text, the entire line from which the text was taken was usually **put**, and the  
16027 buffer was handled as if in line mode, but it was possible to get extremely anomalous behavior.  
16028 In addition, using the **Q** command to switch into **ex** mode, and then doing a **put** often resulted in  
16029 errors as well, such as appending text that was unrelated to the (supposed) contents of the  
16030 buffer. For consistency and simplicity of specification, IEEE Std 1003.1-200x does not permit  
16031 these behaviors. All **ex put** commands are required to operate in line mode, and the contents of  
16032 the buffers are not altered by changing the mode of the editor.

16033 **Read**

16034 Historically, an **ex read** command executed from open or visual mode, executed in an empty file,  
16035 left an empty line as the first line of the file. For consistency and simplicity of specification,  
16036 IEEE Std 1003.1-200x does not permit this behavior. Historically, a **read** in open or visual mode  
16037 from a program left the cursor at the last line read in, not the first. For consistency,  
16038 IEEE Std 1003.1-200x does not permit this behavior.

16039 Historical implementations of **ex** were unable to undo **read** commands that read from the output  
16040 of a program. For consistency, IEEE Std 1003.1-200x does not permit this behavior.

16041 Historically, the **ex** and **vi** message after a successful **read** or **write** command specified  
16042 “characters”, not “bytes”. IEEE Std 1003.1-200x requires that the number of bytes be displayed,  
16043 not the number of characters, because it may be difficult in multi-byte implementations to  
16044 determine the number of characters read. Implementations are encouraged to clarify the  
16045 message displayed to the user.

16046 Historically, reads were not permitted on files other than type regular, except that FIFO files  
16047 could be read (probably only because they did not exist when **ex** and **vi** were originally written).  
16048 Because the historical **ex** evaluated **read!** and **read !** equivalently, there can be no optional way  
16049 to force the read. IEEE Std 1003.1-200x permits, but does not require, this behavior.

16050 **Recover**

16051 Some historical implementations of the editor permitted users to recover the edit buffer contents  
16052 from a previous edit session, and then exit without saving those contents (or explicitly  
16053 discarding them). The intent of IEEE Std 1003.1-200x in requiring that the edit buffer be treated  
16054 as already modified is to prevent this user error.

16055 **Rewind**

16056 Historical implementations supported the **rewind** command when the user was editing the first  
16057 file in the list; that is, the file that the **rewind** command would edit. IEEE Std 1003.1-200x  
16058 requires conformance to historical practice.

16059 **Substitute**

16060 Historically, *ex* accepted an **r** option to the **s** command. The effect of the **r** option was to use the  
 16061 last regular expression used in any command as the pattern, the same as the **~** command. The **r**  
 16062 option is not required by IEEE Std 1003.1-200x. Historically, the **c** and **g** options were toggled; for  
 16063 example, the command **:s/abc/def/** was the same as **s/abc/def/ccccgggg**. For simplicity of  
 16064 specification, IEEE Std 1003.1-200x does not permit this behavior.

16065 The tilde command is often used to replace the last search RE. For example, in the sequence:

```
16066 s/red/blue/
16067 /green
16068 ~
```

16069 the **~** command is equivalent to:

```
16070 s/green/blue/
```

16071 Historically, *ex* accepted all of the following forms:

```
16072 s/abc/def/
16073 s/abc/def
16074 s/abc/
16075 s/abc
```

16076 IEEE Std 1003.1-200x requires conformance to this historical practice.

16077 The **s** command presumes that the **'^'** character only occupies a single column in the display.  
 16078 Much of the *ex* and *vi* specification presumes that the **<space>** only occupies a single column in  
 16079 the display. There are no known character sets for which this is not true.

16080 Historically, the final column position for the substitute commands was based on previous  
 16081 column movements; a search for a pattern followed by a substitution would leave the column  
 16082 position unchanged, while a **0** command followed by a substitution would change the column  
 16083 position to the first non-**<blank>**. For consistency and simplicity of specification,  
 16084 IEEE Std 1003.1-200x requires that the final column position always be set to the first non-  
 16085 **<blank>**.

16086 **Set**

16087 Historical implementations redisplayed all of the options for each occurrence of the **all** keyword.  
 16088 IEEE Std 1003.1-200x permits, but does not require, this behavior.

16089 **Tag**

16090 No requirement is made as to where *ex* and *vi* shall look for the file referenced by the tag entry.  
 16091 Historical practice has been to look for the path found in the **tags** file, based on the current  
 16092 directory. A useful extension found in some implementations is to look based on the directory  
 16093 containing the tags file that held the entry, as well. No requirement is made as to which  
 16094 reference for the tag in the tags file is used. This is deliberate, in order to permit extensions such  
 16095 as multiple entries in a tags file for a tag.

16096 Because users often specify many different tags files, some of which need not be relevant or exist  
 16097 at any particular time, IEEE Std 1003.1-200x requires that error messages about problem tags  
 16098 files be displayed only if the requested tag is not found, and then, only once for each time that  
 16099 the **tag** edit option is changed.

16100 The requirement that the current edit buffer be unmodified is only necessary if the file indicated  
 16101 by the tag entry is not the same as the current file (as defined by the current pathname).

16102 Historically, the file would be reloaded if the filename had changed, as well as if the filename  
16103 was different from the current pathname. For consistency and simplicity of specification,  
16104 IEEE Std 1003.1-200x does not permit this behavior, requiring that the name be the only factor in  
16105 the decision.

16106 Historically, *vi* only searched for tags in the current file from the current cursor to the end of the  
16107 file, and therefore, if the **wrapsan** option was not set, tags occurring before the current cursor  
16108 were not found. IEEE Std 1003.1-200x considers this a bug, and implementations are required to  
16109 search for the first occurrence in the file, regardless.

#### 16110 **Undo**

16111 The **undo** description deliberately uses the word “modified”. The **undo** command is not  
16112 intended to undo commands that replace the contents of the edit buffer, such as **edit**, **next**, **tag**,  
16113 or **recover**.

16114 Cursor positioning after the **undo** command was inconsistent in the historical *vi*, sometimes  
16115 attempting to restore the original cursor position (**global**, **undo**, and **v** commands), and  
16116 sometimes, in the presence of maps, placing the cursor on the last line added or changed instead  
16117 of the first. IEEE Std 1003.1-200x requires a simplified behavior for consistency and simplicity of  
16118 specification.

#### 16119 **Version**

16120 The **version** command cannot be exactly specified since there is no widely-accepted definition of  
16121 what the version information should contain. Implementations are encouraged to do something  
16122 reasonably intelligent.

#### 16123 **Write**

16124 Historically, the *ex* and *vi* message after a successful **read** or **write** command specified  
16125 “characters”, not “bytes”. IEEE Std 1003.1-200x requires that the number of bytes be displayed,  
16126 not the number of characters because it may be difficult in multi-byte implementations to  
16127 determine the number of characters written. Implementations are encouraged to clarify the  
16128 message displayed to the user.

16129 Implementation-defined tests are permitted so that implementations can make additional  
16130 checks; for example, for locks or file modification times.

16131 Historically, attempting to append to a nonexistent file caused an error. It has been left  
16132 unspecified in IEEE Std 1003.1-200x to permit implementations to let the **write** succeed, so that  
16133 the append semantics are similar to those of the historical *csh*.

16134 Historical *vi* permitted empty edit buffers to be written. However, since the way *vi* got around  
16135 dealing with “empty” files was to always have a line in the edit buffer, no matter what, it wrote  
16136 them as files of a single, empty line. IEEE Std 1003.1-200x does not permit this behavior.

16137 Historically, *ex* restored standard output and standard error to their values as of when *ex* was  
16138 invoked, before writes to programs were performed. This could disturb the terminal  
16139 configuration as well as be a security issue for some terminals. IEEE Std 1003.1-200x does not  
16140 permit this, requiring that the program output be captured and displayed as if by the *ex* **print**  
16141 command.

**16142 Adjust Window**

16143 Historically, the line count was set to the value of the **scroll** option if the type character was  
16144 end-of-file. This feature was broken on most historical implementations long ago, however, and  
16145 is not documented anywhere. For this reason, IEEE Std 1003.1-200x is resolutely silent.

16146 Historically, the **z** command was <blank>-sensitive and **z +** and **z -** did different things than **z+**  
16147 and **z-** because the type could not be distinguished from a flag. (The commands **z .** and **z =**  
16148 were historically invalid.) IEEE Std 1003.1-200x requires conformance to this historical practice.

16149 Historically, the **z** command was further <blank>-sensitive in that the *count* could not be  
16150 <blank>-delimited; for example, the commands **z= 5** and **z- 5** were also invalid. Because the  
16151 *count* is not ambiguous with respect to either the type character or the flags, this is not permitted  
16152 by IEEE Std 1003.1-200x.

**16153 Escape**

16154 Historically, *ex* filter commands only read the standard output of the commands, letting  
16155 standard error appear on the terminal as usual. The *vi* utility, however, read both standard  
16156 output and standard error. IEEE Std 1003.1-200x requires the latter behavior for both *ex* and *vi*,  
16157 for consistency.

**16158 Shift Left and Shift Right**

16159 Historically, it was possible to add shift characters to increase the effect of the command; for  
16160 example, <<< outdented (or >>> indented) the lines 3 levels of indentation instead of the default  
16161 1. IEEE Std 1003.1-200x requires conformance to historical practice.

**16162 <control>-D**

16163 Historically, the <control>-D command erased the prompt, providing the user with an unbroken  
16164 presentation of lines from the edit buffer. This is not required by IEEE Std 1003.1-200x;  
16165 implementations are encouraged to provide it if possible. Historically, the <control>-D  
16166 command took, and then ignored, a *count*. IEEE Std 1003.1-200x does not permit this behavior.

**16167 Write Line Number**

16168 Historically, the *ex =* command, when executed in *ex* mode in an empty edit buffer, reported 0,  
16169 and from open or visual mode, reported 1. For consistency and simplicity of specification,  
16170 IEEE Std 1003.1-200x does not permit this behavior.

**16171 Execute**

16172 Historically, *ex* did not correctly handle the inclusion of text input commands (that is, **append**,  
16173 **insert**, and **change**) in executed buffers. IEEE Std 1003.1-200x does not permit this exclusion for  
16174 consistency.

16175 Historically, the logical contents of the buffer being executed did not change if the buffer itself  
16176 were modified by the commands being executed; that is, buffer execution did not support self-  
16177 modifying code. IEEE Std 1003.1-200x requires conformance to historical practice.

16178 Historically, the **@** command took a range of lines, and the **@** buffer was executed once per line,  
16179 with the current line ( ' . ' ) set to each specified line. IEEE Std 1003.1-200x requires conformance  
16180 to historical practice.

16181 Some historical implementations did not notice if errors occurred during buffer execution. This,  
16182 coupled with the ability to specify a range of lines for the *ex @* command, makes it trivial to  
16183 cause them to drop core. IEEE Std 1003.1-200x requires that implementations stop buffer

16184 execution if any error occurs, if the specified line doesn't exist, or if the contents of the edit buffer  
16185 itself are replaced (for example, the buffer executes the `ex:edit` command).

### 16186 **Regular Expressions in ex**

16187 Historical practice is that the characters in the replacement part of the last `s` command—that is,  
16188 those matched by entering a `'~'` in the regular expression—were not further expanded by the  
16189 regular expression engine. So, if the characters contained the string `"a.,"` they would match  
16190 `'a'` followed by `","` and not `'a'` followed by any character. IEEE Std 1003.1-200x requires con-  
16191 formance to historical practice.

### 16192 **Edit Options in ex**

16193 The following paragraphs describe the historical behavior of some edit options that were not, for  
16194 whatever reason, included in IEEE Std 1003.1-200x. Implementations are strongly encouraged to  
16195 only use these names if the functionality described here is fully supported.

16196 **extended** The **extended** edit option has been used in some implementations of `vi` to provide  
16197 extended regular expressions instead of basic regular expressions. This option was  
16198 omitted from IEEE Std 1003.1-200x because it is not widespread historical practice.

16199 **flash** The **flash** edit option historically caused the screen to flash instead of beeping on  
16200 error. This option was omitted from IEEE Std 1003.1-200x because it is not found in  
16201 some historical implementations.

16202 **hardtabs** The **hardtabs** edit option historically defined the number of columns between  
16203 hardware tab settings. This option was omitted from IEEE Std 1003.1-200x because  
16204 it was believed to no longer be generally useful.

16205 **modeline** The **modeline** (sometimes named **modelines**) edit option historically caused `ex` or  
16206 `vi` to read the five first and last lines of the file for editor commands. This option is  
16207 a security problem, and vendors are strongly encouraged to delete it from  
16208 historical implementations.

16209 **open** The **open** edit option historically disallowed the `ex open` and **visual** commands.  
16210 This edit option was omitted because these commands are required by  
16211 IEEE Std 1003.1-200x.

16212 **optimize** The **optimize** edit option historically expedited text throughput by setting the  
16213 terminal to not do automatic carriage returns when printing more than one logical  
16214 line of output. This option was omitted from IEEE Std 1003.1-200x because it was  
16215 intended for terminals without addressable cursors, which are rarely, if ever, still  
16216 used.

16217 **ruler** The **ruler** edit option has been used in some implementations of `vi` to present a  
16218 current row/column ruler for the user. This option was omitted from  
16219 IEEE Std 1003.1-200x because it is not widespread historical practice.

16220 **sourceany** The **sourceany** edit option historically caused `ex` or `vi` to source start-up files that  
16221 were owned by users other than the user running the editor. This option is a  
16222 security problem, and vendors are strongly encouraged to remove it from their  
16223 implementations.

16224 **timeout** The **timeout** edit option historically enabled the (now standard) feature of only  
16225 waiting for a short period before returning keys that could be part of a macro. This  
16226 feature was omitted from IEEE Std 1003.1-200x because its behavior is now  
16227 standard, it is not widely useful, and it was rarely documented.

- 16228       **verbose**       The **verbose** edit option has been used in some implementations of *vi* to cause *vi* to  
 16229       output error messages for common errors; for example, attempting to move the  
 16230       cursor past the beginning or end of the line instead of only alerting the screen. (The  
 16231       historical *vi* only alerted the terminal and presented no message for such errors.  
 16232       The historical editor option **terse** did not select when to present error messages, it  
 16233       only made existing error messages more or less verbose.) This option was omitted  
 16234       from IEEE Std 1003.1-200x because it is not widespread historical practice;  
 16235       however, implementors are encouraged to use it if they wish to provide error  
 16236       messages for naive users.
- 16237       **wrapslen**       The **wrapslen** edit option has been used in some implementations of *vi* to specify an  
 16238       automatic margin measured from the left margin instead of from the right margin.  
 16239       This is useful when multiple screen sizes are being used to edit a single file. This  
 16240       option was omitted from IEEE Std 1003.1-200x because it is not widespread  
 16241       historical practice; however, implementors are encouraged to use it if they add this  
 16242       functionality.
- 16243       **autoindent, ai**
- 16244       Historically, the command **Oa** did not do any autoindentation, regardless of the current  
 16245       indentation of line 1. IEEE Std 1003.1-200x requires that any indentation present in line 1 be used.
- 16246       **autoprint, ap**
- 16247       Historically, the **autoprint** edit option was not completely consistent or based solely on  
 16248       modifications to the edit buffer. Exceptions were the **read** command (when reading from a file,  
 16249       but not from a filter), the **append**, **change**, **insert**, **global**, and **v** commands, all of which were not  
 16250       affected by **autoprint**, and the **tag** command, which was affected by **autoprint**.  
 16251       IEEE Std 1003.1-200x requires conformance to historical practice.
- 16252       Historically, the **autoprint** option only applied to the last of multiple commands entered using  
 16253       vertical-bar delimiters; for example, **delete** <newline> was affected by **autoprint**, but  
 16254       **delete|version** <newline> was not. IEEE Std 1003.1-200x requires conformance to historical  
 16255       practice.
- 16256       **autowrite, aw**
- 16257       Appending the '!' character to the *ex* **next** command to avoid performing an automatic write  
 16258       was not supported in historical implementations. IEEE Std 1003.1-200x requires that the  
 16259       behavior match the other *ex* commands for consistency.
- 16260       **ignorecase, ic**
- 16261       Historical implementations of case-insensitive matching (the **ignorecase** edit option) lead to  
 16262       counterintuitive situations when uppercase characters were used in range expressions.  
 16263       Historically, the process was as follows:
- 16264       1. Take a line of text from the edit buffer.
  - 16265       2. Convert uppercase to lowercase in text line.
  - 16266       3. Convert uppercase to lowercase in regular expressions, except in character class  
 16267       specifications.
  - 16268       4. Match regular expressions against text.
- 16269       This would mean that, with **ignorecase** in effect, the text:

16270 The cat sat on the mat

16271 would be matched by

16272 /^the/

16273 but not by:

16274 /^[A-Z]he/

16275 For consistency with other commands implementing regular expressions, IEEE Std 1003.1-200x  
16276 does not permit this behavior.

### 16277 **paragraphs, para**

16278 Earlier versions of IEEE Std 1003.1-200x made the default **paragraphs** and **sections** edit options  
16279 implementation-defined, arguing they were historically oriented to the UNIX system *troff* text  
16280 formatter, and a “portable user” could use the { }, [[, ]], (, and ) commands in open or visual  
16281 mode and have the cursor stop in unexpected places. IEEE Std 1003.1-200x specifies their values  
16282 in the POSIX locale because the unusual grouping (they only work when grouped into two  
16283 characters at a time) means that they cannot be used for general purpose movement, regardless.

### 16284 **readonly**

16285 Implementations are encouraged to provide the best possible information to the user as to the  
16286 read-only status of the file, with the exception that they should not consider the current special  
16287 privileges of the process. This provides users a safety net because they must force the overwrite  
16288 of read-only files, even when running with additional privileges.

16289 The **readonly** edit option specification largely conforms to historical practice. The only  
16290 difference is that historical implementations did not notice that the user had set the **readonly**  
16291 edit option in cases where the file was already marked read-only for some reason, and would  
16292 therefore reinitialize the **readonly** edit option the next time the contents of the edit buffer were  
16293 replaced. This behavior is disallowed by IEEE Std 1003.1-200x.

### 16294 **report**

16295 The requirement that lines copied to a buffer interact differently than deleted lines is historical  
16296 practice. For example, if the **report** edit option is set to 3, deleting 3 lines will cause a report to be  
16297 written, but 4 lines must be copied before a report is written.

16298 The requirement that the **ex global**, **v**, **open**, **undo**, and **visual** commands present reports based  
16299 on the total number of lines added or deleted during the command execution, and that  
16300 commands executed by the **global** and **v** commands not present reports, is historical practice.  
16301 IEEE Std 1003.1-200x extends historical practice by requiring that buffer execution be treated  
16302 similarly. The reasons for this are two-fold. Historically, only the report by the last command  
16303 executed from the buffer would be seen by the user, as each new report would overwrite the  
16304 last. In addition, the standard developers believed that buffer execution had more in common  
16305 with **global** and **v** commands than it did with other **ex** commands, and should behave similarly,  
16306 for consistency and simplicity of specification.

**16307 showmatch, sm**

16308 The length of time the cursor spends on the matching character is unspecified because the  
16309 timing capabilities of systems are often inexact and variable. The time should be long enough for  
16310 the user to notice, but not long enough for the user to become annoyed. Some implementations  
16311 of *vi* have added a **matchtime** option that permits users to set the number of 0,1 second intervals  
16312 the cursor pauses on the matching character.

**16313 showmode**

16314 The **showmode** option has been used in some historical implementations of *ex* and *vi* to display  
16315 the current editing mode when in open or visual mode. The editing modes have generally  
16316 included “command” and “input”, and sometimes other modes such as “replace” and  
16317 “change”. The string was usually displayed on the bottom line of the screen at the far right-hand  
16318 corner. In addition, a preceding ‘\*’ character often denoted if the contents of the edit buffer had  
16319 been modified. The latter display has sometimes been part of the **showmode** option, and  
16320 sometimes based on another option. This option was not available in the 4 BSD historical  
16321 implementation of *vi*, but was viewed as generally useful, particularly to novice users, and is  
16322 required by IEEE Std 1003.1-200x.

16323 The **smd** shorthand for the **showmode** option was not present in all historical implementations  
16324 of the editor. IEEE Std 1003.1-200x requires it, for consistency.

16325 Not all historical implementations of the editor displayed a mode string for command mode,  
16326 differentiating command mode from text input mode by the absence of a mode string.  
16327 IEEE Std 1003.1-200x permits this behavior for consistency with historical practice, but  
16328 implementations are encouraged to provide a display string for both modes.

**16329 slowopen**

16330 Historically the **slowopen** option was automatically set if the terminal baud rate was less than  
16331 1 200 baud, or if the baud rate was 1 200 baud and the **redraw** option was not set. The **slowopen**  
16332 option had two effects. First, when inserting characters in the middle of a line, characters after  
16333 the cursor would not be pushed ahead, but would appear to be overwritten. Second, when  
16334 creating a new line of text, lines after the current line would not be scrolled down, but would  
16335 appear to be overwritten. In both cases, ending text input mode would cause the screen to be  
16336 refreshed to match the actual contents of the edit buffer. Finally, terminals that were sufficiently  
16337 intelligent caused the editor to ignore the **slowopen** option. IEEE Std 1003.1-200x permits most  
16338 historical behavior, extending historical practice to require **slowopen** behaviors if the edit option  
16339 is set by the user.

**16340 tags**

16341 The default path for tags files is left unspecified as implementations may have their own **tags**  
16342 implementations that do not correspond to the historical ones. The default **tags** option value  
16343 should probably at least include the file **./tags**.

16344 **term**

16345 Historical implementations of *ex* and *vi* ignored changes to the **term** edit option after the initial  
 16346 terminal information was loaded. This is permitted by IEEE Std 1003.1-200x; however,  
 16347 implementations are encouraged to permit the user to modify their terminal type at any time.

16348 **terse**

16349 Historically, the **terse** edit option optionally provided a shorter, less descriptive error message,  
 16350 for some error messages. This is permitted, but not required, by IEEE Std 1003.1-200x.  
 16351 Historically, most common visual mode errors (for example, trying to move the cursor past the  
 16352 end of a line) did not result in an error message, but simply alerted the terminal.  
 16353 Implementations wishing to provide messages for novice users are urged to do so based on the  
 16354 **edit** option **verbose**, and not **terse**.

16355 **window**

16356 In historical implementations, the default for the **window** edit option was based on the baud  
 16357 rate as follows:

16358 1. If the baud rate was less than 1 200, the **edit** option **w300** set the window value; for  
 16359 example, the line:

```
16360 set w300=12
```

16361 would set the window option to 12 if the baud rate was less than 1 200.

16362 2. If the baud rate was equal to 1 200, the **edit** option **w1200** set the window value.

16363 3. If the baud rate was greater than 1 200, the **edit** option **w9600** set the window value.

16364 The **w300**, **w1200**, and **w9600** options do not appear in IEEE Std 1003.1-200x because of their  
 16365 dependence on specific baud rates.

16366 In historical implementations, the size of the window displayed by various commands was  
 16367 related to, but not necessarily the same as, the **window** edit option. For example, the size of the  
 16368 window was set by the *ex* command **visual 10**, but it did not change the value of the **window**  
 16369 edit option. However, changing the value of the **window** edit option did change the number of  
 16370 lines that were displayed when the screen was repainted. IEEE Std 1003.1-200x does not permit  
 16371 this behavior in the interests of consistency and simplicity of specification, and requires that all  
 16372 commands that change the number of lines that are displayed do it by setting the value of the  
 16373 **window** edit option.

16374 **wrapmargin, wm**

16375 Historically, the **wrapmargin** option did not affect maps inserting characters that also had  
 16376 associated *counts*; for example **:map K 5aABC DEF**. Unfortunately, there are widely used  
 16377 maps that depend on this behavior. For consistency and simplicity of specification,  
 16378 IEEE Std 1003.1-200x does not permit this behavior.

16379 Historically, **wrapmargin** was calculated using the column display width of all characters on the  
 16380 screen. For example, an implementation using "**^I**" to represent <tab>s when the **list** edit  
 16381 option was set, where '**^**' and '**I**' each took up a single column on the screen, would calculate  
 16382 the **wrapmargin** based on a value of 2 for each <tab>. The **number** edit option similarly  
 16383 changed the effective length of the line as well. IEEE Std 1003.1-200x requires conformance to  
 16384 historical practice.

16385 **FUTURE DIRECTIONS**

16386 None.

16387 **SEE ALSO**16388 *ed, sed, stty, vi*, the System Interfaces volume of IEEE Std 1003.1-200x, *access()*16389 **CHANGE HISTORY**

16390 First released in Issue 2.

16391 **Issue 5**

16392 The FUTURE DIRECTIONS section is added.

16393 **Issue 6**

16394 This utility is now marked as part of the User Portability Utilities option.

16395 The obsolescent SYNOPSIS is removed, removing the *+command* and *-* options.16396 The following new requirements on POSIX implementations derive from alignment with the  
16397 Single UNIX Specification:

- 16398 • In the *map* command description, the sequence *#digit* is added.
- 16399 • The **directory**, **edcompatible**, **redraw**, and **slowopen** edit options are added.

16400 The *ex* utility is extensively changed for alignment with the IEEE P1003.2b draft standard. This  
16401 includes changes as a result of the IEEE PASC Interpretations 1003.2 #31, #38, #49, #50, #51, #52,  
16402 #55, #56, #57, #61, #62, #63, #64, #65, and #78.

16403 **NAME**16404 `expand` — convert tabs to spaces16405 **SYNOPSIS**16406 UP `expand [-t tablist][file ...]`

16407

16408 **DESCRIPTION**

16409 The *expand* utility shall write files or the standard input to the standard output with `<tab>`s  
 16410 replaced with one or more `<space>`s needed to pad to the next tab stop. Any `<backspace>`s shall  
 16411 be copied to the output and cause the column position count for tab stop calculations to be  
 16412 decremented; the column position count shall not be decremented below zero.

16413 **OPTIONS**

16414 The *expand* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section  
 16415 12.2, Utility Syntax Guidelines.

16416 The following option shall be supported:

16417 `-t tablist` Specify the tab stops. The application shall ensure that the argument *tablist*  
 16418 consists of either a single positive decimal integer or a list of tabstops. If a single  
 16419 number is given, tabs shall be set that number of column positions apart instead of  
 16420 the default 8.

16421 If a list of tabstops is given, the application shall ensure that it consists of a list of  
 16422 two or more positive decimal integers, separated by `<blank>`s or commas, in  
 16423 ascending order. The tabs shall be set at those specific column positions. Each tab  
 16424 stop *N* shall be an integer value greater than zero, and the list is in strictly  
 16425 ascending order. This is taken to mean that, from the start of a line of output,  
 16426 tabbing to position *N* shall cause the next character output to be in the (*N*+1)th  
 16427 column position on that line.

16428 In the event of *expand* having to process a `<tab>` at a position beyond the last of  
 16429 those specified in a multiple tab-stop list, the `<tab>` shall be replaced by a single  
 16430 `<space>` in the output.

16431 **OPERANDS**

16432 The following operand shall be supported:

16433 *file* The pathname of a text file to be used as input.

16434 **STDIN**

16435 See the INPUT FILES section.

16436 **INPUT FILES**

16437 Input files shall be text files.

16438 **ENVIRONMENT VARIABLES**

16439 The following environment variables shall affect the execution of *expand*:

16440 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 16441 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
 16442 Internationalization Variables for the precedence of internationalization variables  
 16443 used to determine the values of locale categories.)

16444 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 16445 internationalization variables.

16446 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 16447 characters (for example, single-byte as opposed to multi-byte characters in

- 16448 arguments and input files), the processing of <tab>s and <space>s, and for the  
16449 determination of the width in column positions each character would occupy on  
16450 an output device.
- 16451 **LC\_MESSAGES**
- 16452 Determine the locale that should be used to affect the format and contents of  
16453 diagnostic messages written to standard error.
- 16454 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 16455 **ASYNCHRONOUS EVENTS**
- 16456 Default.
- 16457 **STDOUT**
- 16458 The standard output shall be equivalent to the input files with <tab>s converted into the  
16459 appropriate number of <space>s.
- 16460 **STDERR**
- 16461 The standard error shall be used only for diagnostic messages.
- 16462 **OUTPUT FILES**
- 16463 None.
- 16464 **EXTENDED DESCRIPTION**
- 16465 None.
- 16466 **EXIT STATUS**
- 16467 The following exit values shall be returned:
- 16468 0 Successful completion
- 16469 >0 An error occurred.
- 16470 **CONSEQUENCES OF ERRORS**
- 16471 The *expand* utility shall terminate with an error message and non-zero exit status upon  
16472 encountering difficulties accessing one of the *file* operands.
- 16473 **APPLICATION USAGE**
- 16474 None.
- 16475 **EXAMPLES**
- 16476 None.
- 16477 **RATIONALE**
- 16478 The *expand* utility is useful for preprocessing text files (before sorting, looking at specific  
16479 columns, and so on) that contain <tab>s.
- 16480 See the Base Definitions volume of IEEE Std 1003.1-200x, Section 3.103, Column Position.
- 16481 The *tablist* option-argument consists of integers in ascending order. Utility Syntax Guideline 8  
16482 mandates that *expand* shall accept the integers (within the single argument) separated using  
16483 either commas or <blank>s.
- 16484 **FUTURE DIRECTIONS**
- 16485 None.
- 16486 **SEE ALSO**
- 16487 *tabs*, *unexpand*

16488 **CHANGE HISTORY**

16489 First released in Issue 4.

16490 **Issue 6**

16491 This utility is now marked as part of the User Portability Utilities option.

16492 The APPLICATION USAGE section is added.

16493 The obsolescent SYNOPSIS is removed.

16494 The *LC\_CTYPE* environment variable description is updated to align with the IEEE P1003.2b draft standard.

16496 The normative text is reworded to avoid use of the term “must” for application requirements.

16497 **NAME**

16498            *expr* — evaluate arguments as an expression

16499 **SYNOPSIS**

16500            *expr operand*

16501 **DESCRIPTION**

16502            The *expr* utility shall evaluate an expression and write the result to standard output.

16503 **OPTIONS**

16504            None.

16505 **OPERANDS**

16506            The single expression evaluated by *expr* shall be formed from the operands, as described in the  
 16507            EXTENDED DESCRIPTION section. The application shall ensure that each of the expression  
 16508            operator symbols:

16509            ( ) | & = > >= < <= != + - \* / % :

16510            and the symbols *integer* and *string* in the table are provided as separate arguments to *expr*.

16511 **STDIN**

16512            Not used.

16513 **INPUT FILES**

16514            None.

16515 **ENVIRONMENT VARIABLES**

16516            The following environment variables shall affect the execution of *expr*:

16517            *LANG*        Provide a default value for the internationalization variables that are unset or null.  
 16518            (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
 16519            Internationalization Variables for the precedence of internationalization variables  
 16520            used to determine the values of locale categories.)

16521            *LC\_ALL*       If set to a non-empty string value, override the values of all the other  
 16522            internationalization variables.

16523            *LC\_COLLATE*

16524            Determine the locale for the behavior of ranges, equivalence classes, and multi-  
 16525            character collating elements within regular expressions and by the string  
 16526            comparison operators.

16527            *LC\_CTYPE*    Determine the locale for the interpretation of sequences of bytes of text data as  
 16528            characters (for example, single-byte as opposed to multi-byte characters in  
 16529            arguments) and the behavior of character classes within regular expressions.

16530            *LC\_MESSAGES*

16531            Determine the locale that should be used to affect the format and contents of  
 16532            diagnostic messages written to standard error.

16533 *XSI*        *NLSPATH*    Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

16534 **ASYNCHRONOUS EVENTS**

16535            Default.

16536 **STDOUT**

16537            The *expr* utility shall evaluate the expression and write the result, followed by a <newline>, to  
 16538            standard output.

16539 **STDERR**

16540 The standard error shall be used only for diagnostic messages.

16541 **OUTPUT FILES**

16542 None.

16543 **EXTENDED DESCRIPTION**

16544 The formation of the expression to be evaluated is shown in the following table. The symbols  
 16545 *expr*, *expr1*, and *expr2* represent expressions formed from *integer* and *string* symbols and the  
 16546 expression operator symbols (all separate arguments) by recursive application of the constructs  
 16547 described in the table. The expressions are listed in order of increasing precedence, with equal-  
 16548 precedence operators grouped between horizontal lines. All of the operators shall be left-  
 16549 associative.

| Expression                                                                                                                                                                                | Description                                                                                                                                                                                                                                                                                                                                                                                                |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>expr1</i>   <i>expr2</i>                                                                                                                                                               | Returns the evaluation of <i>expr1</i> if it is neither null nor zero; otherwise, returns the evaluation of <i>expr2</i> if it is not null; otherwise, zero.                                                                                                                                                                                                                                               |
| <i>expr1</i> & <i>expr2</i>                                                                                                                                                               | Returns the evaluation of <i>expr1</i> if neither expression evaluates to null or zero; otherwise, returns zero.                                                                                                                                                                                                                                                                                           |
| <i>expr1</i> = <i>expr2</i><br><i>expr1</i> > <i>expr2</i><br><i>expr1</i> >= <i>expr2</i><br><i>expr1</i> < <i>expr2</i><br><i>expr1</i> <= <i>expr2</i><br><i>expr1</i> != <i>expr2</i> | Returns the result of a decimal integer comparison if both arguments are integers; otherwise, returns the result of a string comparison using the locale-specific collation sequence. The result of each comparison is 1 if the specified relationship is true, or 0 if the relationship is false.<br>Equal.<br>Greater than.<br>Greater than or equal.<br>Less than.<br>Less than or equal.<br>Not equal. |
| <i>expr1</i> + <i>expr2</i><br><i>expr1</i> - <i>expr2</i>                                                                                                                                | Addition of decimal integer-valued arguments.<br>Subtraction of decimal integer-valued arguments.                                                                                                                                                                                                                                                                                                          |
| <i>expr1</i> * <i>expr2</i><br><i>expr1</i> / <i>expr2</i><br><i>expr1</i> % <i>expr2</i>                                                                                                 | Multiplication of decimal integer-valued arguments.<br>Integer division of decimal integer-valued arguments, producing an integer result.<br>Remainder of integer division of decimal integer-valued arguments.                                                                                                                                                                                            |
| <i>expr1</i> : <i>expr2</i>                                                                                                                                                               | Matching expression; see below.                                                                                                                                                                                                                                                                                                                                                                            |
| ( <i>expr</i> )                                                                                                                                                                           | Grouping symbols. Any expression can be placed within parentheses. Parentheses can be nested to a depth of {EXPR_NEST_MAX}.                                                                                                                                                                                                                                                                                |
| <i>integer</i>                                                                                                                                                                            | An argument consisting only of an (optional) unary minus followed by digits.                                                                                                                                                                                                                                                                                                                               |
| <i>string</i>                                                                                                                                                                             | A string argument; see below.                                                                                                                                                                                                                                                                                                                                                                              |

16581 **Matching Expression**

16582 The '=' matching operator shall compare the string resulting from the evaluation of *expr1* with  
 16583 the regular expression pattern resulting from the evaluation of *expr2*. Regular expression syntax  
 16584 shall be that defined in the Base Definitions volume of IEEE Std 1003.1-200x, Section 9.3, Basic  
 16585 Regular Expressions, except that all patterns are anchored to the beginning of the string (that is,  
 16586 only sequences starting at the first character of a string are matched by the regular expression)  
 16587 and, therefore, it is unspecified whether '^' is a special character in that context. Usually, the  
 16588 matching operator shall return a string representing the number of characters matched ('0' on  
 16589 failure). Alternatively, if the pattern contains at least one regular expression subexpression  
 16590 "[\\(\\.\\.\\.\\)]", the string corresponding to "\\1" shall be returned.

16591 **String Operand**

16592 A string argument is an argument that cannot be identified as an *integer* argument or as one of  
 16593 the expression operator symbols shown in the OPERANDS section.

16594 The use of string arguments **length**, **substr**, **index**, or **match** produces unspecified results.

16595 **EXIT STATUS**

16596 The following exit values shall be returned:

16597 0 The *expression* evaluates to neither null nor zero.

16598 1 The *expression* evaluates to null or zero.

16599 2 Invalid *expression*.

16600 >2 An error occurred.

16601 **CONSEQUENCES OF ERRORS**

16602 Default.

16603 **APPLICATION USAGE**

16604 After argument processing by the shell, *expr* is not required to be able to tell the difference  
 16605 between an operator and an operand except by the value. If "\$a" is '=', the command:

```
16606 expr $a = '='
```

16607 looks like:

```
16608 expr = = =
```

16609 as the arguments are passed to *expr* (and they all may be taken as the '=' operator). The  
 16610 following works reliably:

```
16611 expr X$a = X=
```

16612 Also note that this volume of IEEE Std 1003.1-200x permits implementations to extend utilities.  
 16613 The *expr* utility permits the integer arguments to be preceded with a unary minus. This means  
 16614 that an integer argument could look like an option. Therefore, the conforming application must  
 16615 employ the "--" construct of Guideline 10 of the Base Definitions volume of  
 16616 IEEE Std 1003.1-200x, Section 12.2, Utility Syntax Guidelines to protect its operands if there is  
 16617 any chance the first operand might be a negative integer (or any string with a leading minus).

16618 **EXAMPLES**

16619 The *expr* utility has a rather difficult syntax:

- 16620 • Many of the operators are also shell control operators or reserved words, so they have to be  
 16621 escaped on the command line.

- 16622 • Each part of the expression is composed of separate arguments, so liberal usage of <blank>s  
 16623 is required. For example:

16624  
 16625  
 16626  
 16627  
 16628

| Invalid                       | Valid                              |
|-------------------------------|------------------------------------|
| <code>expr 1+2</code>         | <code>expr 1 + 2</code>            |
| <code>expr "1 + 2"</code>     | <code>expr 1 + 2</code>            |
| <code>expr 1 + (2 * 3)</code> | <code>expr 1 + \( 2 \* 3 \)</code> |

16629 In many cases, the arithmetic and string features provided as part of the shell command  
 16630 language are easier to use than their equivalents in *expr*. Newly written scripts should avoid  
 16631 *expr* in favor of the new features within the shell; see Section 2.5 (on page 2235) and Section 2.6.4  
 16632 (on page 2243).

16633 The following command:

16634 `a=$(expr $a + 1)`

16635 adds 1 to the variable *a*.

16636 The following command, for "*\$a*" equal to either `/usr/abc/file` or just `file`:

16637 `expr $a : '.*\/\(.*\)' \| $a`

16638 returns the last segment of a pathname (that is, **file**). Applications should avoid the character  
 16639 `/` used alone as an argument: *expr* may interpret it as the division operator.

16640 The following command:

16641 `expr "//$a" : '.*\/\(.*\)'`

16642 is a better representation of the previous example. The addition of the `"/` characters  
 16643 eliminates any ambiguity about the division operator and simplifies the whole expression. Also  
 16644 note that pathnames may contain characters contained in the *IFS* variable and should be quoted  
 16645 to avoid having "*\$a*" expand into multiple arguments.

16646 The following command:

16647 `expr "$VAR" : '.*'`

16648 returns the number of characters in *VAR*.

#### 16649 RATIONALE

16650 In an early proposal, EREs were used in the matching expression syntax. This was changed to  
 16651 BREs to avoid breaking historical applications.

16652 The use of a leading circumflex in the BRE is unspecified because many historical  
 16653 implementations have treated it as a special character, despite their system documentation. For  
 16654 example:

16655 `expr foo : ^foo`      `expr ^foo : ^foo`

16656 return 3 and 0, respectively, on those systems; their documentation would imply the reverse.  
 16657 Thus, the anchoring condition is left unspecified to avoid breaking historical scripts relying on  
 16658 this undocumented feature.

#### 16659 FUTURE DIRECTIONS

16660 None.

16661 **SEE ALSO**

16662           Section 2.6.4

16663 **CHANGE HISTORY**

16664           First released in Issue 2.

16665 **Issue 5**

16666           FUTURE DIRECTIONS section added.

16667 **Issue 6**16668           The *expr* utility is aligned with the IEEE P1003.2b draft standard, to include resolution of IEEE  
16669           PASC Interpretation 1003.2 #104.

16670           The normative text is reworded to avoid use of the term “must” for application requirements.

16671 **NAME**

16672           false — return false value

16673 **SYNOPSIS**

16674           false

16675 **DESCRIPTION**

16676           The *false* utility shall return with a non-zero exit code.

16677 **OPTIONS**

16678           None.

16679 **OPERANDS**

16680           None.

16681 **STDIN**

16682           Not used.

16683 **INPUT FILES**

16684           None.

16685 **ENVIRONMENT VARIABLES**

16686           None.

16687 **ASYNCHRONOUS EVENTS**

16688           Default.

16689 **STDOUT**

16690           Not used.

16691 **STDERR**

16692           None.

16693 **OUTPUT FILES**

16694           None.

16695 **EXTENDED DESCRIPTION**

16696           None.

16697 **EXIT STATUS**

16698           The *false* utility always shall exit with a value other than zero.

16699 **CONSEQUENCES OF ERRORS**

16700           Default.

16701 **APPLICATION USAGE**

16702           None.

16703 **EXAMPLES**

16704           None.

16705 **RATIONALE**

16706           None.

16707 **FUTURE DIRECTIONS**

16708           None.

16709 **SEE ALSO**

16710           *true*

16711 **CHANGE HISTORY**

16712 First released in Issue 2.

## 16713 NAME

16714 `fc` — process the command history list

## 16715 SYNOPSIS

16716 UP `fc [-r][-e editor] [first[last]]`16717 `fc -l[-nr] [first[last]]`16718 `fc -s[old=new][first]`

16719

## 16720 DESCRIPTION

16721 The `fc` utility shall list, or shall edit and re-execute, commands previously entered to an  
16722 interactive *sh*.

16723 The command history list shall reference commands by number. The first number in the list is  
16724 selected arbitrarily. The relationship of a number to its command shall not change except when  
16725 the user logs in and no other process is accessing the list, at which time the system may reset the  
16726 numbering to start the oldest retained command at another number (usually 1). When the  
16727 number reaches an implementation-defined upper limit, which shall be no smaller than the  
16728 value in *HISTSIZ*E or 32 767 (whichever is greater), the shell may wrap the numbers, starting the  
16729 next command with a lower number (usually 1). However, despite this optional wrapping of  
16730 numbers, `fc` shall maintain the time-ordering sequence of the commands. For example, if four  
16731 commands in sequence are given the numbers 32 766, 32 767, 1 (wrapped), and 2 as they are  
16732 executed, command 32 767 is considered the command previous to 1, even though its number is  
16733 higher.

16734 When commands are edited (when the `-l` option is not specified), the resulting lines shall be  
16735 entered at the end of the history list and then re-executed by *sh*. The `fc` command that caused the  
16736 editing shall not be entered into the history list. If the editor returns a non-zero exit status, this  
16737 shall suppress the entry into the history list and the command re-execution. Any command line  
16738 variable assignments or redirection operators used with `fc` shall affect both the `fc` command itself  
16739 as well as the command that results; for example:

16740 `fc -s -- -l 2>/dev/null`16741 reinvokes the previous command, suppressing standard error for both `fc` and the previous  
16742 command.

## 16743 OPTIONS

16744 The `fc` utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2,  
16745 Utility Syntax Guidelines.

16746 The following options shall be supported:

16747 `-e editor` Use the editor named by *editor* to edit the commands. The *editor* string is a utility  
16748 name, subject to search via the *PATH* variable (see the Base Definitions volume of  
16749 IEEE Std 1003.1-200x, Chapter 8, Environment Variables). The value in the *FCEDIT*  
16750 variable shall be used as a default when `-e` is not specified. If *FCEDIT* is null or  
16751 unset, *ed* shall be used as the editor.

16752 `-l` (The letter ell.) List the commands rather than invoking an editor on them. The  
16753 commands shall be written in the sequence indicated by the *first* and *last* operands,  
16754 as affected by `-r`, with each command preceded by the command number.

16755 `-n` Suppress command numbers when listing with `-l`.16756 `-r` Reverse the order of the commands listed (with `-l`) or edited (with neither `-l` nor  
16757 `-s`).

- 16758            -s            Reexecute the command without invoking an editor.
- 16759 **OPERANDS**
- 16760            The following operands shall be supported:
- 16761            *first, last*
- 16762                            Select the commands to list or edit. The number of previous commands that can be  
16763                            accessed shall be determined by the value of the *HISTSIZE* variable. The value of  
16764                            *first* or *last* or both shall be one of the following:
- 16765            [+]*number*    A positive number representing a command number; command  
16766                            numbers can be displayed with the -l option.
- 16767            -*number*        A negative decimal number representing the command that was  
16768                            executed *number* of commands previously. For example, -1 is the  
16769                            immediately previous command.
- 16770            *string*        A string indicating the most recently entered command that begins  
16771                            with that string. If the *old=new* operand is not also specified with -s,  
16772                            the string form of the *first* operand cannot contain an embedded  
16773                            equal sign.
- 16774                            When the synopsis form with -s is used:
- 16775
  - If *first* is omitted, the previous command shall be used.
- 16776                            For the synopsis forms without -s:
- 16777
  - If *last* is omitted, *last* shall default to the previous command when -l is  
16778                            specified; otherwise, it shall default to *first*.
  - If *first* and *last* are both omitted, the previous 16 commands shall be listed or  
16779                            the previous single command shall be edited (based on the -l option).
  - If *first* and *last* are both present, all of the commands from *first* to *last* shall be  
16780                            edited (without -l) or listed (with -l). Editing multiple commands shall be  
16781                            accomplished by presenting to the editor all of the commands at one time, each  
16782                            command starting on a new line. If *first* represents a newer command than *last*,  
16783                            the commands shall be listed or edited in reverse sequence, equivalent to using  
16784                            -r. For example, the following commands on the first line are equivalent to the  
16785                            corresponding commands on the second:  
16786                            fc -r 10 20        fc     30 40  
16787                            fc     20 10        fc -r 40 30
  - When a range of commands is used, it shall not be an error to specify *first* or *last*  
16788                            values that are not in the history list; *fc* shall substitute the value representing  
16789                            the oldest or newest command in the list, as appropriate. For example, if there  
16790                            are only ten commands in the history list, numbered 1 to 10:  
16791                            fc -l  
16792                            fc 1 99  
16793                            shall list and edit, respectively, all ten commands.
- 16794            *old=new*        Replace the first occurrence of string *old* in the commands to be re-executed by the  
16795                            string *new*.
- 16796
- 16797
- 16798

16799 **STDIN**

16800 Not used.

16801 **INPUT FILES**

16802 None.

16803 **ENVIRONMENT VARIABLES**16804 The following environment variables shall affect the execution of *fc*:

16805 *FCEDIT* This variable, when expanded by the shell, shall determine the default value for  
 16806 the *-e editor* option's *editor* option-argument. If *FCEDIT* is null or unset, *ed* shall be  
 16807 used as the editor.

16808 *HISTFILE* Determine a pathname naming a command history file. If the *HISTFILE* variable is  
 16809 not set, the shell may attempt to access or create a file *.sh\_history* in the directory  
 16810 referred to by the *HOME* environment variable. If the shell cannot obtain both read  
 16811 and write access to, or create, the history file, it shall use an unspecified  
 16812 mechanism that allows the history to operate properly. (References to history  
 16813 "file" in this section shall be understood to mean this unspecified mechanism in  
 16814 such cases.) An implementation may choose to access this variable only when  
 16815 initializing the history file; this initialization shall occur when *fc* or *sh* first attempt  
 16816 to retrieve entries from, or add entries to, the file, as the result of commands issued  
 16817 by the user, the file named by the *ENV* variable, or implementation-defined system  
 16818 start-up files. In some historical shells, the history file is initialized just after the  
 16819 *ENV* file has been processed. Therefore, it is implementation-defined whether  
 16820 changes made to *HISTFILE* after the history file has been initialized are effective.  
 16821 Implementations may choose to disable the history list mechanism for users with  
 16822 appropriate privileges who do not set *HISTFILE*; the specific circumstances under  
 16823 which this occurs are implementation-defined. If more than one instance of the  
 16824 shell is using the same history file, it is unspecified how updates to the history file  
 16825 from those shells interact. As entries are deleted from the history file, they shall be  
 16826 deleted oldest first. It is unspecified when history file entries are physically  
 16827 removed from the history file.

16828 *HISTSIZE* Determine a decimal number representing the limit to the number of previous  
 16829 commands that are accessible. If this variable is unset, an unspecified default  
 16830 greater than or equal to 128 shall be used. The maximum number of commands in  
 16831 the history list is unspecified, but shall be at least 128. An implementation may  
 16832 choose to access this variable only when initializing the history file, as described  
 16833 under *HISTFILE*. Therefore, it is unspecified whether changes made to *HISTSIZE*  
 16834 after the history file has been initialized are effective.

16835 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 16836 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
 16837 Internationalization Variables for the precedence of internationalization variables  
 16838 used to determine the values of locale categories.)

16839 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 16840 internationalization variables.

16841 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 16842 characters (for example, single-byte as opposed to multi-byte characters in  
 16843 arguments and input files).

16844 *LC\_MESSAGES*

16845 Determine the locale that should be used to affect the format and contents of  
 16846 diagnostic messages written to standard error.

- 16847 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 16848 **ASYNCHRONOUS EVENTS**
- 16849 Default.
- 16850 **STDOUT**
- 16851 When the **-l** option is used to list commands, the format of each command in the list shall be as follows:
- 16852
- 16853 `"%d\t%s\n", <line number>, <command>`
- 16854 If both the **-l** and **-n** options are specified, the format of each command shall be:
- 16855 `"\t%s\n", <command>`
- 16856 If the *<command>* consists of more than one line, the lines after the first shall be displayed as:
- 16857 `"\t%s\n", <continued-command>`
- 16858 **STDERR**
- 16859 The standard error shall be used only for diagnostic messages.
- 16860 **OUTPUT FILES**
- 16861 None.
- 16862 **EXTENDED DESCRIPTION**
- 16863 None.
- 16864 **EXIT STATUS**
- 16865 The following exit values shall be returned:
- 16866 0 Successful completion of the listing.
- 16867 >0 An error occurred.
- 16868 Otherwise, the exit status shall be that of the commands executed by *fc*.
- 16869 **CONSEQUENCES OF ERRORS**
- 16870 Default.
- 16871 **APPLICATION USAGE**
- 16872 Since editors sometimes use file descriptors as integral parts of their editing, redirecting their file descriptors as part of the *fc* command can produce unexpected results. For example, if *vi* is the *FCEDIT* editor, the command:
- 16873 `fc -s | more`
- 16874 does not work correctly on many systems.
- 16875 Users on windowing systems may want to have separate history files for each window by setting *HISTFILE* as follows:
- 16876
- 16877 `HISTFILE=$HOME/.sh_hist$$`
- 16878
- 16880 **EXAMPLES**
- 16881 None.
- 16882 **RATIONALE**
- 16883 This utility is based on the *fc* built-in of the KornShell.
- 16884 An early proposal specified the **-e** option as `[-e editor [old= new ]]`, which is not historical practice. Historical practice in *fc* of either `[-e editor]` or `[-e - [ old= new ]]` is acceptable, but not both together. To clarify this, a new option **-s** was introduced replacing the `[-e -]`. This resolves the conflict and makes *fc* conform to the Utility Syntax Guidelines.
- 16885
- 16886
- 16887

- 16888        *HISTFILE*    Some implementations of the KornShell check for the superuser and do not create  
 16889        a history file unless *HISTFILE* is set. This is done primarily to avoid creating  
 16890        unlinked files in the root file system when logging in during single-user mode.  
 16891        *HISTFILE* must be set for the superuser to have history.
- 16892        *HISTSIZE*    Needed to limit the size of history files. It is the intent of the standard developers  
 16893        that when two shells share the same history file, commands that are entered in one  
 16894        shell shall be accessible by the other shell. Because of the difficulties of  
 16895        synchronization over a network, the exact nature of the interaction is unspecified.
- 16896        The initialization process for the history file can be dependent on the system start-up files, in  
 16897        that they may contain commands that effectively preempt the settings the user has for *HISTFILE*  
 16898        and *HISTSIZE*. For example, function definition commands are recorded in the history file. If the  
 16899        system administrator includes function definitions in some system start-up file called before the  
 16900        *ENV* file, the history file is initialized before the user can influence its characteristics. In some  
 16901        historical shells, the history file is initialized just after the *ENV* file has been processed. Because  
 16902        of these situations, the text requires the initialization process to be implementation-defined.
- 16903        Consideration was given to omitting the *fc* utility in favor of the command line editing feature in  
 16904        *sh*. For example, in *vi* editing mode, typing "<ESC> v" is equivalent to:
- 16905        `EDITOR=vi fc`
- 16906        However, the *fc* utility allows the user the flexibility to edit multiple commands simultaneously  
 16907        (such as *fc 10 20*) and to use editors other than those supported by *sh* for command line editing.
- 16908        In the KornShell, the alias *r* ("re-do") is preset to *fc -e -* (equivalent to the POSIX *fc -s*). This is  
 16909        probably an easier command name to remember than *fc* ("fix command"), but it does not meet  
 16910        the Utility Syntax Guidelines. Renaming *fc* to *hist* or *redo* was considered, but since this  
 16911        description closely matches historical KornShell practice already, such a renaming was seen as  
 16912        gratuitous. Users are free to create aliases whenever odd historical names such as *fc*, *awk*, *cat*,  
 16913        *grep*, or *yacc* are standardized by POSIX.
- 16914        Command numbers have no ordering effects; they are like serial numbers. The *-r* option and  
 16915        *-number* operand address the sequence of command execution, regardless of serial numbers. So,  
 16916        for example, if the command number wrapped back to 1 at some arbitrary point, there would be  
 16917        no ambiguity associated with traversing the wrap point. For example, if the command history  
 16918        were:
- 16919        `32766: echo 1`  
 16920        `32767: echo 2`  
 16921        `1: echo 3`
- 16922        the number *-2* refers to command 32 767 because it is the second previous command, regardless  
 16923        of serial number.
- 16924        **FUTURE DIRECTIONS**
- 16925        None.
- 16926        **SEE ALSO**
- 16927        *sh*
- 16928        **CHANGE HISTORY**
- 16929        First released in Issue 4.

16930 **Issue 5**

16931 FUTURE DIRECTIONS section added.

16932 **Issue 6**

16933 This utility is now marked as part of the User Portability Utilities option.

16934 In the ENVIRONMENT VARIABLES section, the text “user’s home directory” is updated to  
16935 “directory referred to by the *HOME* environment variable”.

16936 **NAME**

16937 fg — run jobs in the foreground

16938 **SYNOPSIS**16939 UP fg [*job\_id*]

16940

16941 **DESCRIPTION**16942 If job control is enabled (see the description of *set -m*), the *fg* utility shall move a background job  
16943 from the current environment (see Section 2.12 (on page 2263)) into the foreground.16944 Using *fg* to place a job into the foreground shall remove its process ID from the list of those  
16945 “known in the current shell execution environment”; see Section 2.9.3.1 (on page 2252).16946 **OPTIONS**

16947 None.

16948 **OPERANDS**

16949 The following operand shall be supported:

16950 *job\_id* Specify the job to be run as a foreground job. If no *job\_id* operand is given, the  
16951 *job\_id* for the job that was most recently suspended, placed in the background or  
16952 run as a background job, shall be used. The format of *job\_id* is described in the Base  
16953 Definitions volume of IEEE Std 1003.1-200x, Section 3.203, Job Control Job ID.16954 **STDIN**

16955 Not used.

16956 **INPUT FILES**

16957 None.

16958 **ENVIRONMENT VARIABLES**16959 The following environment variables shall affect the execution of *fg*:16960 *LANG* Provide a default value for the internationalization variables that are unset or null.  
16961 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
16962 Internationalization Variables for the precedence of internationalization variables  
16963 used to determine the values of locale categories.)16964 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
16965 internationalization variables.16966 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
16967 characters (for example, single-byte as opposed to multi-byte characters in  
16968 arguments).16969 *LC\_MESSAGES*16970 Determine the locale that should be used to affect the format and contents of  
16971 diagnostic messages written to standard error.16972 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.16973 **ASYNCHRONOUS EVENTS**

16974 Default.

16975 **STDOUT**16976 The *fg* utility shall write the command line of the job to standard output in the following format:16977 "%s\n", <*command*>

16978 **STDERR**

16979 The standard error shall be used only for diagnostic messages.

16980 **OUTPUT FILES**

16981 None.

16982 **EXTENDED DESCRIPTION**

16983 None.

16984 **EXIT STATUS**

16985 The following exit values shall be returned:

16986 0 Successful completion.

16987 >0 An error occurred.

16988 **CONSEQUENCES OF ERRORS**

16989 If job control is disabled, the *fg* utility shall exit with an error and no job shall be placed in the foreground.

16991 **APPLICATION USAGE**

16992 The *fg* utility does not work as expected when it is operating in its own utility execution environment because that environment has no applicable jobs to manipulate. See the APPLICATION USAGE section for *bg* (on page 2410). For this reason, *fg* is generally implemented as a shell regular built-in.

16996 **EXAMPLES**

16997 None.

16998 **RATIONALE**

16999 The extensions to the shell specified in this volume of IEEE Std 1003.1-200x have mostly been based on features provided by the KornShell. The job control features provided by *bg*, *fg*, and *jobs* are also based on the KornShell. The standard developers examined the characteristics of the C shell versions of these utilities and found that differences exist. Despite widespread use of the C shell, the KornShell versions were selected for this volume of IEEE Std 1003.1-200x to maintain a degree of uniformity with the rest of the KornShell features selected (such as the very popular command line editing features).

17006 **FUTURE DIRECTIONS**

17007 None.

17008 **SEE ALSO**

17009 *bg*, *kill*, *jobs*, *wait*

17010 **CHANGE HISTORY**

17011 First released in Issue 4.

17012 **Issue 6**

17013 This utility is now marked as part of the User Portability Utilities option.

17014 The APPLICATION USAGE section is added.

17015 The JC marking is removed from the SYNOPSIS since job control is mandatory in this issue.

## 17016 NAME

17017 file — determine file type

## 17018 SYNOPSIS

17019 UP file [-dhi][-M file][-m file] file ...

17020

## 17021 DESCRIPTION

17022 The *file* utility shall perform a series of tests on each specified *file* in an attempt to classify it:

17023 1. If the file is not a regular file, its file type shall be identified. The file types directory, FIFO, |  
 17024 socket, block special, and character special shall be identified as such. Other |  
 17025 implementation-defined file types may also be identified. |

17026 2. If the file is a regular file, and:

17027 a. The file is zero-length, it shall be identified as an empty file.

17028 b. The file is not zero-length, *file* shall examine an initial segment of the file and shall  
 17029 make a guess at identifying its contents or whether it is an executable binary file.  
 17030 (The answer is not guaranteed to be correct.)

17031 If *file* does not exist, cannot be read, or its file status could not be determined, the output shall  
 17032 indicate that the file was processed, but that its type could not be determined.

17033 If *file* is a symbolic link, by default the link shall be resolved and *file* shall test the type of file  
 17034 referenced by the symbolic link.

## 17035 OPTIONS

17036 The *file* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2,  
 17037 Utility Syntax Guidelines.

17038 The following options shall be supported by the implementation:

17039 **-d** Apply any default system tests to the file.

17040 **-h** When a symbolic link is encountered, identify the file as a symbolic link. If **-h** is  
 17041 not specified and *file* is a symbolic link that refers to a nonexistent file, *file* shall  
 17042 identify the file as a symbolic link, as if **-h** had been specified.

17043 **-i** If a file is a regular file, do not attempt to classify the type of the file further, but  
 17044 identify the file as specified in the STDOUT section, using a *<type>* string that  
 17045 contains the string "regular file".

17046 **-M file** Specify the name of a file containing tests that shall be applied to a file in order to  
 17047 classify it (see the EXTENDED DESCRIPTION). No default system tests shall be  
 17048 applied.

17049 **-m file** Specify the name of a file containing tests that shall be applied to a file in order to  
 17050 classify it (see the EXTENDED DESCRIPTION).

17051 If multiple instances of the **-m**, **-d**, or **-M** options are specified, the concatenation of the tests  
 17052 specified, in the order specified, shall be the set of tests that are applied. If a **-M** option is  
 17053 specified, no tests other than those specified using the **-d**, **-M**, and **-m** options shall be applied  
 17054 to the file. If neither the **-d** nor **-M** options are specified, any default system tests shall be  
 17055 applied after any tests specified using the **-m** option.

17056 **OPERANDS**

17057           The following operand shall be supported:

17058           *file*           A pathname of a file to be tested.

17059 **STDIN**

17060           Not used.

17061 **INPUT FILES**

17062           The *file* can be any file type.

17063 **ENVIRONMENT VARIABLES**

17064           The following environment variables shall affect the execution of *file*:

17065           *LANG*           Provide a default value for the internationalization variables that are unset or null.  
17066                           (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
17067                           Internationalization Variables for the precedence of internationalization variables  
17068                           used to determine the values of locale categories.)

17069           *LC\_ALL*       If set to a non-empty string value, override the values of all the other  
17070                           internationalization variables.

17071           *LC\_CTYPE*   Determine the locale for the interpretation of sequences of bytes of text data as  
17072                           characters (for example, single-byte as opposed to multi-byte characters in  
17073                           arguments and input files).

17074           *LC\_MESSAGES*

17075                           Determine the locale that should be used to affect the format and contents of  
17076                           diagnostic messages written to standard error and informative messages written to  
17077                           standard output.

17078 XSI           *NLSPATH*   Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

17079 **ASYNCHRONOUS EVENTS**

17080           Default.

17081 **STDOUT**

17082           In the POSIX locale, the following format shall be used to identify each operand, *file* specified:

17083           "%s: %s\n", <*file*>, <*type*>

17084           The values for <*type*> are unspecified, except that in the POSIX locale, if *file* is identified as one  
17085           of the types listed in the following table, <*type*> shall contain (but is not limited to) the  
17086           corresponding string. Each space shown in the strings shall be exactly one <space>.

17087

Table 4-8 File Utility Output Strings

17088

| If <i>file</i> is a:                                         | < <i>type</i> > shall contain the string: |
|--------------------------------------------------------------|-------------------------------------------|
| Directory                                                    | directory                                 |
| FIFO                                                         | fifo                                      |
| Socket                                                       | socket                                    |
| Block special                                                | block special                             |
| Character special                                            | character special                         |
| Executable binary                                            | executable                                |
| Empty regular file                                           | empty                                     |
| Symbolic link                                                | symbolic link to                          |
| <i>ar</i> archive library (see <i>ar</i> )                   | archive                                   |
| Extended <i>cpio</i> format (see <i>pax</i> )                | cpio archive                              |
| Extended <i>tar</i> format (see <b>ustar</b> in <i>pax</i> ) | tar archive                               |
| Shell script                                                 | commands text                             |
| C-language source                                            | c program text                            |
| FORTRAN source                                               | fortran program text                      |

17089

17090

17091

17092

17093

17094

17095

17096

17097

17098

17099

17100

17101

17102

17103

If *file* is identified as a symbolic link (see **-h**), the following alternative output format shall be used:

17104

17105

"%s: %s %s\n", <*file*>, <*type*>, <*contents of link*>"

17106

If the file named by the *file* operand does not exist or cannot be read, the string "cannot open" shall be included as part of the <*type*> field, but this shall not be considered an error that affects the exit status. If the type of the file named by the *file* operand cannot be determined, the string "data" shall be included as part of the <*type*> field, but this shall not be considered an error that affects the exit status.

17107

17108

17109

17110

17111 **STDERR**

17112

The standard error shall be used only for diagnostic messages.

17113 **OUTPUT FILES**

17114

None.

17115 **EXTENDED DESCRIPTION**

17116

A file specified as an option-argument to the **-m** or **-M** options shall contain one test per line, which shall be applied to the file. If the test succeeds, the message field of the line shall be printed and no further tests shall be applied, with the exception that tests on immediately following lines beginning with a single '**>**' character shall be applied.

17117

17118

17119

17120

Each line shall be composed of the following four <blank>-separated fields:

17121

*offset*

An unsigned number (optionally preceded by a single '**>**' character) specifying the *offset*, in bytes, of the value in the file that is to be compared against the *value* field of the line. If the file is shorter than the specified offset, the test shall fail.

17122

17123

17124

17125

17126

17127

17128

17129

If the *offset* begins with the character '**>**', the test contained in the line shall not be applied to the file unless the test on the last line for which the *offset* did not begin with a '**>**' was successful. By default, the *offset* shall be interpreted as an unsigned decimal number. With a leading 0x or 0X, the *offset* shall be interpreted as a hexadecimal number; otherwise, with a leading 0, the *offset* shall be interpreted as an octal number.

17130

*type*

The type of the value in the file to be tested. The type shall consist of the type specification characters c, d, f, s, and u, specifying character, signed decimal, floating point, string, and unsigned decimal, respectively.

17131

17132

17133 The *type* string shall be interpreted as the bytes from the file starting at the  
17134 specified *offset* and including the same number of bytes specified by the *value* field.  
17135 If insufficient bytes remain in the file past the *offset* to match the *value* field, the test  
17136 shall fail.

17137 The type specification characters *d*, *f*, and *u* can be followed by an optional |  
17138 unsigned decimal integer that specifies the number of bytes represented by the  
17139 type. The type specification character *f* can be followed by an optional *F*, *D*, or *L*, |  
17140 indicating that the value is of type **float**, **double**, or **long double**, respectively. The  
17141 type specification characters *d* and *u* can be followed by an optional *C*, *S*, *I*, or *L*, |  
17142 indicating that the value is of type **char**, **short**, **int**, or **long**, respectively.

17143 The default number of bytes represented by the type specifiers *d*, *f*, and *u* shall |  
17144 correspond to their respective C-language types as follows. If the system claims  
17145 conformance to the C-Language Development Utilities option, those specifiers  
17146 shall correspond to the default sizes used in the *c99* utility. Otherwise, the default  
17147 sizes shall be implementation-defined.

17148 For the type specifier characters *d* and *u*, the default number of bytes shall |  
17149 correspond to the size of a basic integer type of the implementation. For these  
17150 specifier characters, the implementation shall support values of the optional  
17151 number of bytes to be converted corresponding to the number of bytes in the C-  
17152 language types **char**, **short**, **int**, or **long**. These numbers can also be specified by an  
17153 application as the characters *C*, *S*, *I*, and *L*, respectively. The byte order used when |  
17154 interpreting numeric values is implementation-defined, but shall correspond to the  
17155 order in which a constant of the corresponding type is stored in memory on the  
17156 system.

17157 For the type specifier *f*, the default number of bytes shall correspond to the |  
17158 number of bytes in the basic double precision floating-point data type of the  
17159 underlying implementation. The implementation shall support values of the  
17160 optional number of bytes to be converted corresponding to the number of bytes in  
17161 the C-language types **float**, **double**, and **long double**. These numbers can also be  
17162 specified by an application as the characters *F*, *D*, and *L*, respectively. |

17163 All type specifiers, except for *s*, can be followed by a mask specifier of the form |  
17164 *&number*. The mask value shall be AND'ed with the value before the comparison  
17165 with the value from the file is made. By default, the mask shall be interpreted as an  
17166 unsigned decimal number. With a leading 0x or 0X, the mask shall be interpreted  
17167 as an unsigned hexadecimal number; otherwise, with a leading 0, the mask shall be  
17168 interpreted as an unsigned octal number.

17169 The strings **byte**, **short**, **long**, and **string** shall also be supported as type fields,  
17170 being interpreted as *dC*, *dS*, *dL*, and *s*, respectively. |

17171 *value* The *value* to be compared with the value from the file.

17172 If the specifier from the type field is *s* or **string**, then interpret the value as a string. |  
17173 Otherwise, interpret it as a number. If the value is a string, then the test shall  
17174 succeed only when a string value exactly matches the bytes from the file.

17175 If the *value* is a string, it can contain the following sequences:

17176 *\character* The backslash-escape sequences as specified in the Base  
17177 Definitions volume of IEEE Std 1003.1-200x, Table 5-1, Escape  
17178 Sequences and Associated Actions ('\\', '\a', '\b', '\f',  
17179 '\n', '\r', '\t', '\v'). The results of using any other

17180 character, other than an octal digit, following the backslash are  
 17181 unspecified.

17182            \*octal*            Octal sequences that can be used to represent characters with  
 17183 specific coded values. An octal sequence shall consist of a  
 17184 backslash followed by the longest sequence of one, two, or three  
 17185 octal-digit characters (01234567). If the size of a byte on the  
 17186 system is greater than 9 bits, the valid escape sequence used to  
 17187 represent a byte is implementation-defined.

17188            By default, any value that is not a string shall be interpreted as a signed decimal  
 17189 number. Any such value, with a leading 0x or 0X, shall be interpreted as an  
 17190 unsigned hexadecimal number; otherwise, with a leading zero, the value shall be  
 17191 interpreted as an unsigned octal number.

17192            If the value is not a string, it can be preceded by a character indicating the  
 17193 comparison to be performed. Permissible characters and the comparisons they  
 17194 specify are as follows:

17195            =    The test shall succeed if the value from the file equals the *value* field.  
 17196            <    The test shall succeed if the value from the file is less than the *value* field.  
 17197            >    The test shall succeed if the value from the file is greater than the *value* field.  
 17198            &    The test shall succeed if all of the bits in the *value* field are set in the value  
 17199 from the file.

17200            ^    The test shall succeed if at least one of the bits in the *value* field is not set in the  
 17201 value from the file.

17202            x    The test shall succeed if the file is large enough to contain a value of the type  
 17203 specified starting at the offset specified.

17204            *message*    The *message* to be printed if the test succeeds. The *message* shall be interpreted  
 17205 using the notation for the *printf* formatting specification; see *printf*. If the *value*  
 17206 field was a string, then the value from the file shall be the argument for the *printf*  
 17207 formatting specification; otherwise, the value from the file shall be the argument.

**17208 EXIT STATUS**

17209            The following exit values shall be returned:

17210            0    Successful completion.

17211            >0   An error occurred.

**17212 CONSEQUENCES OF ERRORS**

17213            Default.

**17214 APPLICATION USAGE**

17215            The *file* utility can only be required to guess at many of the file types because only exhaustive  
 17216 testing can determine some types with certainty. For example, binary data on some  
 17217 implementations might match the initial segment of an executable or a *tar* archive.

17218            Note that the table indicates that the output contains the stated string. Systems may add text  
 17219 before or after the string. For executables, as an example, the machine architecture and various  
 17220 facts about how the file was link-edited may be included.

17221 **EXAMPLES**

17222 Determine whether an argument is a binary executable file:

```
17223 file "$1" | grep -Fq executable &&
17224 printf "%s is executable.\n" "$1"
```

17225 **RATIONALE**

17226 The `-f` option was omitted because the same effect can (and should) be obtained using the *xargs*  
17227 utility.

17228 Historical versions of the *file* utility attempt to identify the following types of files: symbolic link,  
17229 directory, character special, block special, socket, *tar* archive, *cpio* archive, *SCCS* archive, archive  
17230 library, empty, *compress* output, *pack* output, binary data, C source, FORTRAN source, assembler  
17231 source, *nroff*/*troff*/*eqn*/*tbl* source *troff* output, shell script, C shell script, English text, ASCII text,  
17232 various executables, APL workspace, compiled terminfo entries, and *CURSES* screen images.  
17233 Only those types that are reasonably well specified in POSIX or are directly related to POSIX  
17234 utilities are listed in the table.

17235 Implementations that support symbolic links are encouraged to use the string "symbolic  
17236 link" to identify them.

17237 Historical systems have used a "magic file" named `/etc/magic` to help identify file types. Because  
17238 it is generally useful for users and scripts to be able to identify special file types, the `-m` flag and  
17239 a portable format for user-created magic files has been specified. No requirement is made that an  
17240 implementation of *file* use this method of identifying files, only that users be permitted to add  
17241 their own classifying tests.

17242 In addition, three options have been added to historical practice. The `-d` flag has been added to  
17243 permit users to cause their tests to follow any default system tests. The `-i` flag has been added to  
17244 permit users to test portably for regular files in shell scripts. The `-M` flag has been added to  
17245 permit users to ignore any default system tests.

17246 The historical `-c` option was omitted as not particularly useful to users or portable shell scripts.  
17247 In addition, a reasonable implementation of the *file* utility would report any errors found each  
17248 time the magic file is read.

17249 The historical format of the magic file was the same as that specified by the Rationale in the  
17250 previous version of IEEE Std 1003.1-200x for the *offset*, *value*, and *message* fields; however, it used  
17251 less precise type fields than the format specified by the current normative text. The new type  
17252 field values are a superset of the historical ones.

17253 The following is an example magic file:

```
17254 0 short      070707          cpio archive
17255 0 short      0143561        Byte-swapped cpio archive
17256 0 string     070707          ASCII cpio archive
17257 0 long       0177555        Very old archive
17258 0 short      0177545        Old archive
17259 0 short      017437         Old packed data
17260 0 string     \037\036       Packed data
17261 0 string     \377\037       Compacted data
17262 0 string     \037\235       Compressed data
17263 >2 byte&0x80 >0          Block compressed
17264 >2 byte&0x1f x           %d bits
17265 0 string     \032\001       Compiled Terminfo Entry
17266 0 short      0433          Curses screen image
17267 0 short      0434          Curses screen image
```

|       |   |        |                    |                                    |
|-------|---|--------|--------------------|------------------------------------|
| 17268 | 0 | string | <ar>               | System V Release 1 archive         |
| 17269 | 0 | string | !<arch>\n__.SYMDEF | Archive random library             |
| 17270 | 0 | string | !<arch>            | Archive                            |
| 17271 | 0 | string | ARF_BEGARF         | PHIGS clear text archive           |
| 17272 | 0 | long   | 0x137A2950         | Scalable OpenFont binary           |
| 17273 | 0 | long   | 0x137A2951         | Encrypted scalable OpenFont binary |

17274           The use of a basic integer data type is intended to allow the implementation to choose a word  
17275           size commonly used by applications on that architecture.

#### 17276 **FUTURE DIRECTIONS**

17277           None.

#### 17278 **SEE ALSO**

17279           *ls*

#### 17280 **CHANGE HISTORY**

17281           First released in Issue 4.

#### 17282 **Issue 6**

17283           This utility is now marked as part of the User Portability Utilities option.

17284           Options and an EXTENDED DESCRIPTION are added as specified in the IEEE P1003.2b draft  
17285           standard.

17286           IEEE PASC Interpretations 1003.2 #192 and #178 are applied.

17287 **NAME**

17288 find — find files

17289 **SYNOPSIS**17290 find [-H | -L] *path* ... [*operand\_expression* ...]17291 **DESCRIPTION**

17292 The *find* utility shall recursively descend the directory hierarchy from each file specified by *path*,  
 17293 evaluating a Boolean expression composed of the primaries described in the OPERANDS section  
 17294 for each file encountered.

17295 The *find* utility shall be able to descend to arbitrary depths in a file hierarchy and shall not fail  
 17296 due to path length limitations (unless a *path* operand specified by the application exceeds  
 17297 {PATH\_MAX} requirements).

17298 The *find* utility shall detect infinite loops; that is, entering a previously visited directory that is an  
 17299 ancestor of the last file encountered. When it detects an infinite loop, *find* shall write a  
 17300 diagnostic message to standard error and shall either recover its position in the hierarchy or  
 17301 terminate.

17302 **OPTIONS**

17303 The *find* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section  
 17304 12.2, Utility Syntax Guidelines.

17305 The following options shall be supported by the implementation:

17306 **-H** Cause the file information and file type evaluated for each symbolic link  
 17307 encountered on the command line to be those of the file referenced by the link, and  
 17308 not the link itself. If the referenced file does not exist, the file information and type  
 17309 shall be for the link itself. File information for all symbolic links not on the  
 17310 command line shall be that of the link itself.

17311 **-L** Cause the file information and file type evaluated for each symbolic link to be  
 17312 those of the file referenced by the link, and not the link itself.

17313 Specifying more than one of the mutually-exclusive options **-H** and **-L** shall not be considered  
 17314 an error. The last option specified shall determine the behavior of the utility.

17315 **OPERANDS**

17316 The following operands shall be supported:

17317 The *path* operand is a pathname of a starting point in the directory hierarchy.

17318 The first argument that starts with a '-', or is a '!' or a '(' , and all subsequent arguments  
 17319 shall be interpreted as an *expression* made up of the following primaries and operators. In the  
 17320 descriptions, wherever *n* is used as a primary argument, it shall be interpreted as a decimal  
 17321 integer optionally preceded by a plus ('+') or minus ('-') sign, as follows:

17322 **+n** More than *n*.

17323 **n** Exactly *n*.

17324 **-n** Less than *n*.

17325 The following primaries shall be supported:

17326 **-name** *pattern*

17327 The primary shall evaluate as true if the basename of the filename being examined  
 17328 matches *pattern* using the pattern matching notation described in Section 2.13 (on  
 17329 page 2264).

- 17330        **-nouser**        The primary shall evaluate as true if the file belongs to a user ID for which the  
17331                    *getpwuid()* function defined in the System Interfaces volume of  
17332                    IEEE Std 1003.1-200x (or equivalent) returns NULL.
- 17333        **-nogroup**        The primary shall evaluate as true if the file belongs to a group ID for which the  
17334                    *getgrgid()* function defined in the System Interfaces volume of  
17335                    IEEE Std 1003.1-200x (or equivalent) returns NULL.
- 17336        **-xdev**         The primary always shall evaluate as true; it shall cause *find* not to continue  
17337                    descending past directories that have a different device ID (*st\_dev*, see the *stat()*  
17338                    function defined in the System Interfaces volume of IEEE Std 1003.1-200x). If any  
17339                    **-xdev** primary is specified, it shall apply to the entire expression even if the **-xdev**  
17340                    primary would not normally be evaluated.
- 17341        **-prune**         The primary always shall evaluate as true; it shall cause *find* not to descend the  
17342                    current pathname if it is a directory. If the **-depth** primary is specified, the **-prune**  
17343                    primary shall have no effect.
- 17344        **-perm [-]mode**  
17345                    The *mode* argument is used to represent file mode bits. It shall be identical in  
17346                    format to the *symbolic\_mode* operand described in *chmod* (on page 2438), and shall  
17347                    be interpreted as follows. To start, a template shall be assumed with all file mode  
17348                    bits cleared. An *op* symbol of '+' shall set the appropriate mode bits in the  
17349                    template; '-' shall clear the appropriate bits; '=' shall set the appropriate mode  
17350                    bits, without regard to the contents of process' file mode creation mask. The *op*  
17351                    symbol of '-' cannot be the first character of *mode*; this avoids ambiguity with the  
17352                    optional leading hyphen. Since the initial mode is all bits off, there are not any  
17353                    symbolic modes that need to use '-' as the first character.
- 17354                    If the hyphen is omitted, the primary shall evaluate as true when the file  
17355                    permission bits exactly match the value of the resulting template.
- 17356                    Otherwise, if *mode* is prefixed by a hyphen, the primary shall evaluate as true if at  
17357                    least all the bits in the resulting template are set in the file permission bits.
- 17358        **-perm [-]onum**  
17359                    If the hyphen is omitted, the primary shall evaluate as true when the file  
17360                    permission bits exactly match the value of the octal number *onum* and only the bits  
17361                    corresponding to the octal mask 07777 shall be compared. (See the description of  
17362                    the octal *mode* in *chmod* (on page 2438).) Otherwise, if *onum* is prefixed by a  
17363                    hyphen, the primary shall evaluate as true if at least all of the bits specified in *onum*  
17364                    that are also set in the octal mask 07777 are set.
- 17365        **-type c**         The primary shall evaluate as true if the type of the file is *c*, where *c* is 'b', 'c',  
17366                    'd', 'l', 'p', 'f', or 's' for block special file, character special file, directory,  
17367                    symbolic link, FIFO, regular file, or socket, respectively.
- 17368        **-links n**         The primary shall evaluate as true if the file has *n* links.
- 17369        **-user uname**       The primary shall evaluate as true if the file belongs to the user *uname*. If *uname* is  
17370                    a decimal integer and the *getpwnam()* (or equivalent) function does not return a  
17371                    valid user name, *uname* shall be interpreted as a user ID.
- 17372        **-group gname**  
17373                    The primary shall evaluate as true if the file belongs to the group *gname*. If *gname*  
17374                    is a decimal integer and the *getgrnam()* (or equivalent) function does not return a  
17375                    valid group name, *gname* shall be interpreted as a group ID.

17376        **-size** *n*[*c*]     The primary shall evaluate as true if the file size in bytes, divided by 512 and  
17377                             rounded up to the next integer, is *n*. If *n* is followed by the character ' *c* ', the size  
17378                             shall be in bytes.

17379        **-atime** *n*       The primary shall evaluate as true if the file access time subtracted from the  
17380                             initialization time, divided by 86 400 (with any remainder discarded), is *n*.

17381        **-ctime** *n*       The primary shall evaluate as true if the time of last change of file status  
17382                             information subtracted from the initialization time, divided by 86 400 (with any  
17383                             remainder discarded), is *n*.

17384        **-mtime** *n*       The primary shall evaluate as true if the file modification time subtracted from the  
17385                             initialization time, divided by 86 400 (with any remainder discarded), is *n*.

17386        **-exec** *utility\_name* [*argument* . . . ] ;                             |  
17387        **-exec** *utility\_name* [*argument* . . . ] ; { } +                         |  
17388                             The end of the primary expression shall be punctuated by a semicolon or by a plus  
17389                             sign. Only a plus sign that follows an argument containing the two characters  
17390                             " { } " shall punctuate the end of the primary expression. Other uses of the plus  
17391                             sign shall not be treated as special.                         |

17392                             If the primary expression is punctuated by a semicolon, the utility *utility\_name*     |  
17393                             shall be invoked once for each pathname and the primary shall evaluate as true if     |  
17394                             the utility returns a zero value as exit status. A *utility\_name* or *argument* containing     |  
17395                             only the two characters " { } " shall be replaced by the current pathname.     |

17396                             If the primary expression is punctuated by a plus sign, the primary shall always     |  
17397                             evaluate as true, and the pathnames for which the primary is evaluated shall be     |  
17398                             aggregated into sets. The utility *utility\_name* shall be invoked once for each set of     |  
17399                             aggregated pathnames. Each invocation shall begin after the last pathname in the     |  
17400                             set is aggregated, and shall be completed before the *find* utility exits and before the     |  
17401                             first pathname in the next set (if any) is aggregated for this primary, but it is     |  
17402                             otherwise unspecified whether the invocation occurs before, during, or after the     |  
17403                             evaluations of other primaries. If any invocation returns a non-zero value as exit     |  
17404                             status, the *find* utility shall return a non-zero exit status. An argument containing     |  
17405                             only the two characters " { } " shall be replaced by the set of aggregated     |  
17406                             pathnames, with each pathname passed as a separate argument to the invoked     |  
17407                             utility in the same order that it was aggregated. The size of any set of two or more     |  
17408                             pathnames shall be limited such that execution of the utility does not cause the     |  
17409                             system's {ARG\_MAX} limit to be exceeded. If more than one argument containing     |  
17410                             only the two characters " { } " is present, the behavior is unspecified.     |

17411                             If a *utility\_name* or *argument* string contains the two characters " { } ", but not just     |  
17412                             the two characters " { } ", it is implementation-defined whether *find* replaces those     |  
17413                             two characters or uses the string without change. The current directory for the     |  
17414                             invocation of *utility\_name* shall be the same as the current directory when the *find*     |  
17415                             utility was started. If the *utility\_name* names any of the special built-in utilities (see     |  
17416                             Section 2.14 (on page 2266)), the results are undefined.     |

17417        **-ok** *utility\_name* [*argument* . . . ] ;                             |  
17418                             The **-ok** primary shall be equivalent to **-exec**, except that the use of a plus sign to     |  
17419                             punctuate the end of the primary expression need not be supported, and *find* shall     |  
17420                             request affirmation of the invocation of *utility\_name* using the current file as an     |  
17421                             argument by writing to standard error as described in the STDERR section. If the     |  
17422                             response on standard input is affirmative, the utility shall be invoked. Otherwise,     |  
17423                             the command shall not be invoked and the value of the **-ok** operand shall be false.     |

- 17424        **-print**        The primary always shall evaluate as true; it shall cause the current pathname to  
17425                    be written to standard output.
- 17426        **-newer file**    The primary shall evaluate as true if the modification time of the current file is  
17427                    more recent than the modification time of the file named by the pathname *file*.
- 17428        **-depth**        The primary shall always evaluate as true; it shall cause descent of the directory  
17429                    hierarchy to be done so that all entries in a directory are acted on before the  
17430                    directory itself. If a **-depth** primary is not specified, all entries in a directory shall  
17431                    be acted on after the directory itself. If any **-depth** primary is specified, it shall  
17432                    apply to the entire expression even if the **-depth** primary would not normally be  
17433                    evaluated.
- 17434        The primaries can be combined using the following operators (in order of decreasing  
17435                    precedence):
- 17436        (*expression*)    True if *expression* is true.
- 17437        !*expression*    Negation of a primary; the unary NOT operator.
- 17438        *expression* [**-a**] *expression*  
17439                    Conjunction of primaries; the AND operator is implied by the juxtaposition of two  
17440                    primaries or made explicit by the optional **-a** operator. The second expression  
17441                    shall not be evaluated if the first expression is false.
- 17442        *expression* **-o** *expression*  
17443                    Alternation of primaries; the OR operator. The second expression shall not be  
17444                    evaluated if the first expression is true.
- 17445        If no *expression* is present, **-print** shall be used as the expression. Otherwise, if the given  
17446                    expression does not contain any of the primaries **-exec**, **-ok**, or **-print**, the given expression shall  
17447                    be effectively replaced by:
- 17448        ( *given\_expression* ) **-print**
- 17449        The **-user**, **-group**, and **-newer** primaries each shall evaluate their respective arguments only  
17450                    once.
- 17451        **STDIN**
- 17452        If the **-ok** primary is used, the response shall be read from the standard input. An entire line  
17453                    shall be read as the response. Otherwise, the standard input shall not be used.
- 17454        **INPUT FILES**
- 17455                    None.
- 17456        **ENVIRONMENT VARIABLES**
- 17457        The following environment variables shall affect the execution of *find*:
- 17458        **LANG**        Provide a default value for the internationalization variables that are unset or null.  
17459                    (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
17460                    Internationalization Variables for the precedence of internationalization variables  
17461                    used to determine the values of locale categories.)
- 17462        **LC\_ALL**        If set to a non-empty string value, override the values of all the other  
17463                    internationalization variables.
- 17464        **LC\_COLLATE**  
17465                    Determine the locale for the behavior of ranges, equivalence classes and multi-  
17466                    character collating elements used in the pattern matching notation for the **-n**  
17467                    option and in the extended regular expression defined for the **yesexpr** locale

- 17468 keyword in the *LC\_MESSAGES* category.
- 17469 *LC\_CTYPE* This variable determines the locale for the interpretation of sequences of bytes of  
 17470 text data as characters (for example, single-byte as opposed to multi-byte  
 17471 characters in arguments), the behavior of character classes within the pattern  
 17472 matching notation used for the *-n* option, and the behavior of character classes  
 17473 within regular expressions used in the extended regular expression defined for the  
 17474 *yesexpr* locale keyword in the *LC\_MESSAGES* category.
- 17475 *LC\_MESSAGES*  
 17476 Determine the locale for the processing of affirmative responses that should be  
 17477 used to affect the format and contents of diagnostic messages written to standard  
 17478 error.
- 17479 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 17480 *PATH* Determine the location of the *utility\_name* for the *-exec* and *-ok* primaries, as  
 17481 described in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 8,  
 17482 Environment Variables.
- 17483 **ASYNCHRONOUS EVENTS**
- 17484 Default.
- 17485 **STDOUT**
- 17486 The *-print* primary shall cause the current pathnames to be written to standard output. The  
 17487 format shall be:  
 17488 "%s\n", <path>
- 17489 **STDERR**
- 17490 The *-ok* primary shall write a prompt to standard error containing at least the *utility\_name* to be  
 17491 invoked and the current pathname. In the POSIX locale, the last non-<blank> in the prompt shall  
 17492 be ' ? '. The exact format used is unspecified.
- 17493 Otherwise, the standard error shall be used only for diagnostic messages.
- 17494 **OUTPUT FILES**
- 17495 None.
- 17496 **EXTENDED DESCRIPTION**
- 17497 None.
- 17498 **EXIT STATUS**
- 17499 The following exit values shall be returned:
- 17500 0 All *path* operands were traversed successfully.
- 17501 >0 An error occurred.
- 17502 **CONSEQUENCES OF ERRORS**
- 17503 Default.

17504 **APPLICATION USAGE**

17505 When used in operands, pattern matching notation, semicolons, opening parentheses, and  
 17506 closing parentheses are special to the shell and must be quoted (see Section 2.2 (on page 2232)).

17507 The bit that is traditionally used for sticky (historically 01000) is specified in the **-perm** primary  
 17508 using the octal number argument form. Since this bit is not defined by this volume of  
 17509 IEEE Std 1003.1-200x, applications must not assume that it actually refers to the traditional  
 17510 sticky bit.

17511 **EXAMPLES**

17512 1. The following commands are equivalent:

```
17513 find .
17514 find . -print
```

17515 They both write out the entire directory hierarchy from the current directory.

17516 2. The following command:

```
17517 find / \( -name tmp -o -name '*.xx' \) -atime +7 -exec rm {} \;
```

17518 removes all files named **tmp** or ending in **.xx** that have not been accessed for seven or more  
 17519 24-hour periods.

17520 3. The following command:

```
17521 find . -perm -o+w,+s
```

17522 prints (**-print** is assumed) the names of all files in or below the current directory, with all  
 17523 of the file permission bits **S\_ISUID**, **S\_ISGID**, and **S\_IWOTH** set.

17524 4. The following command:

```
17525 find . -name SCCS -prune -o -print
```

17526 recursively prints pathnames of all files in the current directory and below, but skips  
 17527 directories named **SCCS** and files in them.

17528 5. The following command:

```
17529 find . -print -name SCCS -prune
```

17530 behaves as in the previous example, but prints the names of the **SCCS** directories.

17531 6. The following command is roughly equivalent to the **-nt** extension to *test*:

```
17532 if [ -n "$(find file1 -prune -newer file2)" ]; then
17533     printf %s\n "file1 is newer than file2"
17534 fi
```

17535 7. The descriptions of **-atime**, **-ctime**, and **-mtime** use the terminology *n* “86 400 second  
 17536 periods (days)”. For example, a file accessed at 23:59 is selected by:

```
17537 find . -atime -1 -print
```

17538 at 00:01 the next day (less than 24 hours later, not more than one day ago); the midnight  
 17539 boundary between days has no effect on the 24-hour calculation.

17540 **RATIONALE**

17541 The **-a** operator was retained as an optional operator for compatibility with historical shell  
 17542 scripts, even though it is redundant with expression concatenation.

17543 The descriptions of the `'-'` modifier on the `mode` and `onum` arguments to the `-perm` primary  
17544 agree with historical practice on BSD and System V implementations. System V and BSD  
17545 documentation both describe it in terms of checking additional bits; in fact, it uses the same bits,  
17546 but checks for having at least all of the matching bits set instead of having exactly the matching  
17547 bits set.

17548 The exact format of the interactive prompts is unspecified. Only the general nature of the  
17549 contents of prompts are specified because:

17550 • Implementations may desire more descriptive prompts than those used on historical  
17551 implementations.

17552 • Since the historical prompt strings do not terminate with `<newline>`s, there is no portable  
17553 way for another program to interact with the prompts of this utility via pipes.

17554 Therefore, an application using this prompting option relies on the system to provide the most  
17555 suitable dialog directly with the user, based on the general guidelines specified.

17556 The `-name file` operand was changed to use the shell pattern matching notation so that `find` is  
17557 consistent with other utilities using pattern matching.

17558 The `-size` operand refers to the size of a file, rather than the number of blocks it may occupy in  
17559 the file system. The intent is that the `st_size` field defined in the System Interfaces volume of  
17560 IEEE Std 1003.1-200x should be used, not the `st_blocks` found in historical implementations.  
17561 There are at least two reasons for this:

17562 1. In both System V and BSD, `find` only uses `st_size` in size calculations for the operands  
17563 specified by this volume of IEEE Std 1003.1-200x. (BSD uses `st_blocks` only when processing  
17564 the `-ls` primary.)

17565 2. Users usually think of file size in terms of bytes, which is also the unit used by the `ls` utility  
17566 for the output from the `-l` option. (In both System V and BSD, `ls` uses `st_size` for the `-l`  
17567 option size field and uses `st_blocks` for the `ls -s` calculations. This volume of  
17568 IEEE Std 1003.1-200x does not specify `ls -s`.)

17569 The descriptions of `-atime`, `-ctime`, and `-mtime` were changed from the SVID description of `n`  
17570 “days” to “24-hour periods”. The description is also different in terms of the exact timeframe for  
17571 the `n` case (*versus* the `+n` or `-n`), but it matches all known historical implementations. It refers to  
17572 one 86 400 second period in the past, not any time from the beginning of that period to the  
17573 current time. For example, `-atime 3` is true if the file was accessed any time in the period from 72  
17574 hours to 48 hours ago.

17575 Historical implementations do not modify `"{}"` when it appears as a substring of an `-exec` or  
17576 `-ok utility_name` or argument string. There have been numerous user requests for this extension,  
17577 so this volume of IEEE Std 1003.1-200x allows the desired behavior. At least one recent  
17578 implementation does support this feature, but encountered several problems in managing  
17579 memory allocation and dealing with multiple occurrences of `"{}"` in a string while it was being  
17580 developed, so it is not yet required behavior.

17581 Assuming the presence of `-print` was added to correct a historical pitfall that plagues novice  
17582 users, it is entirely upward-compatible from the historical System V `find` utility. In its simplest  
17583 form (*find directory*), it could be confused with the historical BSD fast `find`. The BSD developers  
17584 agreed that adding `-print` as a default expression was the correct decision and have added the  
17585 fast `find` functionality within a new utility called `locate`.

17586 Historically, the `-L` option was implemented using the primary `-follow`. The `-H` and `-L` options  
17587 were added for two reasons. First, they offer a finer granularity of control and consistency with  
17588 other programs that walk file hierarchies. Second, the `-follow` primary always evaluated to true.

17589 As they were historically really global variables that took effect before the traversal began, some  
 17590 valid expressions had unexpected results. An example is the expression **-print -o -follow**.  
 17591 Because **-print** always evaluates to true, the standard order of evaluation implies that **-follow**  
 17592 would never be evaluated. This was never the case. Historical practice for the **-follow** primary,  
 17593 however, is not consistent. Some implementations always follow symbolic links on the  
 17594 command line whether **-follow** is specified or not. Others follow symbolic links on the  
 17595 command line only if **-follow** is specified. Both behaviors are provided by the **-H** and **-L**  
 17596 options, but scripts using the current **-follow** primary would be broken if the **-follow** option is  
 17597 specified to work either way.

17598 Since the **-L** option resolves all symbolic links and the **-type l** primary is true for symbolic links  
 17599 that still exist after symbolic links have been resolved, the command:

```
17600 find -L . -type l
```

17601 prints a list of symbolic links reachable from the current directory that do not resolve to  
 17602 accessible files.

17603 A feature of SVR4's *find* utility was the **-exec** primary's + terminator. This allowed filenames  
 17604 containing special characters (especially <newline>s) to be grouped together without the  
 17605 problems that occur if such filenames are piped to *xargs*. Other implementations have added  
 17606 other ways to get around this problem, notably a **-print0** primary that wrote filenames with a  
 17607 null byte terminator. This was considered here, but not adopted. Using a null terminator meant  
 17608 that any utility that was going to process *find*'s **-print0** output had to add a new option to parse  
 17609 the null terminators it would now be reading.

17610 The "**-exec ... {} +**" syntax adopted was a result of IEEE PASC Interpretation 1003.2 #210.  
 17611 It should be noted that this is an incompatible change to the ISO/IEC 9899:1999 standard. For  
 17612 example, the following command prints all files with a '-' after their name if they are regular  
 17613 files, and a '+' otherwise:

```
17614 find / -type f -exec echo {} - ';' -o -exec echo {} + ';' |
```

17615 The change invalidates usage like this. Even though the previous standard stated that this usage  
 17616 would work, in practice many did not support it and the standard developers felt it better to  
 17617 now state that this was not allowable.

#### 17618 FUTURE DIRECTIONS

17619 None.

#### 17620 SEE ALSO

17621 *chmod*, *pax*, *sh*, *test*, the System Interfaces volume of IEEE Std 1003.1-200x, *stat()*

#### 17622 CHANGE HISTORY

17623 First released in Issue 2.

#### 17624 Issue 5

17625 FUTURE DIRECTIONS section added.

#### 17626 Issue 6

17627 The following new requirements on POSIX implementations derive from alignment with the  
 17628 Single UNIX Specification:

- 17629 • The **-perm [-]onum** primary is supported.

17630 The *find* utility is aligned with the IEEE P1003.2b draft standard, to include processing of  
 17631 symbolic links and changes to the description of the **atime**, **ctime**, and **mtime** operands.

17632 IEEE PASC Interpretation 1003.2 #210 is applied, extending the **-exec** operand.

## 17633 NAME

17634 fold — filter for folding lines

## 17635 SYNOPSIS

17636 fold [-bs][-w *width*][*file...*]

## 17637 DESCRIPTION

17638 The *fold* utility is a filter that shall fold lines from its input files, breaking the lines to have a  
 17639 maximum of *width* column positions (or bytes, if the **-b** option is specified). Lines shall be  
 17640 broken by the insertion of a <newline> such that each output line (referred to later in this section  
 17641 as a *segment*) is the maximum width possible that does not exceed the specified number of  
 17642 column positions (or bytes). A line shall not be broken in the middle of a character. The behavior  
 17643 is undefined if *width* is less than the number of columns any single character in the input would  
 17644 occupy.

17645 If the <carriage-return>s, <backspace>s, or <tab>s are encountered in the input, and the **-b**  
 17646 option is not specified, they shall be treated specially:

17647 <backspace> The current count of line width shall be decremented by one, although the count  
 17648 never shall become negative. The *fold* utility shall not insert a <newline>  
 17649 immediately before or after any <backspace>.

17650 <carriage-return>

17651 The current count of line width shall be set to zero. The *fold* utility shall not insert a  
 17652 <newline> immediately before or after any <carriage-return>.

17653 <tab> Each <tab> encountered shall advance the column position pointer to the next tab  
 17654 stop. Tab stops shall be at each column position *n* such that *n* modulo 8 equals 1.

## 17655 OPTIONS

17656 The *fold* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section  
 17657 12.2, Utility Syntax Guidelines.

17658 The following options shall be supported:

17659 **-b** Count *width* in bytes rather than column positions.

17660 **-s** If a segment of a line contains a <blank> within the first *width* column positions (or  
 17661 bytes), break the line after the last such <blank> meeting the width constraints. If  
 17662 there is no <blank> meeting the requirements, the **-s** option shall have no effect for  
 17663 that output segment of the input line.

17664 **-w *width*** Specify the maximum line length, in column positions (or bytes if **-b** is specified).  
 17665 The results are unspecified if *width* is not a positive decimal number. The default  
 17666 value shall be 80.

## 17667 OPERANDS

17668 The following operand shall be supported:

17669 *file* A pathname of a text file to be folded. If no *file* operands are specified, the standard  
 17670 input shall be used.

## 17671 STDIN

17672 The standard input shall be used only if no *file* operands are specified. See the INPUT FILES  
 17673 section.

17674 **INPUT FILES**

17675 If the **-b** option is specified, the input files shall be text files except that the lines are not limited  
 17676 to {LINE\_MAX} bytes in length. If the **-b** option is not specified, the input files shall be text files.

17677 **ENVIRONMENT VARIABLES**

17678 The following environment variables shall affect the execution of *fold*:

17679 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 17680 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
 17681 Internationalization Variables for the precedence of internationalization variables  
 17682 used to determine the values of locale categories.)

17683 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 17684 internationalization variables.

17685 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 17686 characters (for example, single-byte as opposed to multi-byte characters in  
 17687 arguments and input files), and for the determination of the width in column  
 17688 positions each character would occupy on a constant-width font output device.

17689 **LC\_MESSAGES**

17690 Determine the locale that should be used to affect the format and contents of  
 17691 diagnostic messages written to standard error.

17692 XSI **NLSPATH** Determine the location of message catalogs for the processing of **LC\_MESSAGES**.

17693 **ASYNCHRONOUS EVENTS**

17694 Default.

17695 **STDOUT**

17696 The standard output shall be a file containing a sequence of characters whose order shall be  
 17697 preserved from the input files, possibly with inserted <newline>s.

17698 **STDERR**

17699 The standard error shall be used only for diagnostic messages. |

17700 **OUTPUT FILES**

17701 None.

17702 **EXTENDED DESCRIPTION**

17703 None.

17704 **EXIT STATUS**

17705 The following exit values shall be returned:

17706 0 All input files were processed successfully.

17707 >0 An error occurred.

17708 **CONSEQUENCES OF ERRORS**

17709 Default.

**17710 APPLICATION USAGE**

17711 The *cut* and *fold* utilities can be used to create text files out of files with arbitrary line lengths. The  
17712 *cut* utility should be used when the number of lines (or records) needs to remain constant. The  
17713 *fold* utility should be used when the contents of long lines need to be kept contiguous.

17714 The *fold* utility is frequently used to send text files to printers that truncate, rather than fold, lines  
17715 wider than the printer is able to print (usually 80 or 132 column positions).

**17716 EXAMPLES**

17717 An example invocation that submits a file of possibly long lines to the printer (under the  
17718 assumption that the user knows the line width of the printer to be assigned by *lp*):

```
17719 fold -w 132 bigfile | lp
```

**17720 RATIONALE**

17721 Although terminal input in canonical processing mode requires the erase character (frequently  
17722 set to <backspace>) to erase the previous character (not byte or column position), terminal  
17723 output is not buffered and is extremely difficult, if not impossible, to parse correctly; the  
17724 interpretation depends entirely on the physical device that actually displays/prints/stores the  
17725 output. In all known internationalized implementations, the utilities producing output for mixed  
17726 column-width output assume that a <backspace> backs up one column position and outputs  
17727 enough <backspace>s to return to the start of the character when <backspace> is used to  
17728 provide local line motions to support underlining and emboldening operations. Since *fold*  
17729 without the **-b** option is dealing with these same constraints, <backspace> is always treated as  
17730 backing up one column position rather than backing up one character.

17731 Historical versions of the *fold* utility assumed 1 byte was one character and occupied one column  
17732 position when written out. This is no longer always true. Since the most common usage of *fold* is  
17733 believed to be folding long lines for output to limited-length output devices, this capability was  
17734 preserved as the default case. The **-b** option was added so that applications could *fold* files with  
17735 arbitrary length lines into text files that could then be processed by the standard utilities. Note  
17736 that although the width for the **-b** option is in bytes, a line is never split in the middle of a  
17737 character. (It is unspecified what happens if a width is specified that is too small to hold a single  
17738 character found in the input followed by a <newline>.)

17739 The tab stops are hardcoded to be every eighth column to meet historical practice. No new  
17740 method of specifying other tab stops was invented.

**17741 FUTURE DIRECTIONS**

17742 None.

**17743 SEE ALSO**

17744 *cut*

**17745 CHANGE HISTORY**

17746 First released in Issue 4.

**17747 Issue 6**

17748 The normative text is reworded to avoid use of the term “must” for application requirements.

17749 **NAME**17750 fort77 — FORTRAN compiler (**FORTRAN**)17751 **SYNOPSIS**

```
17752 FD fort77 [-c][-g][-L directory]... [-O optlevel][-o outfile][-s][-w]
17753 operand...
```

17754

17755 **DESCRIPTION**

17756 The *fort77* utility is the interface to the FORTRAN compilation system; it shall accept the full  
 17757 FORTRAN-77 language defined by the ANSI X3.9-1978 standard. The system conceptually  
 17758 consists of a compiler and link editor. The files referenced by *operands* are compiled and linked  
 17759 to produce an executable file. It is unspecified whether the linking occurs entirely within the  
 17760 operation of *fort77*; some implementations may produce objects that are not fully resolved until  
 17761 the file is executed.

17762 If the **-c** option is present, for all pathname operands of the form *file.f*, the files:

17763 \$(basename *pathname.f*).o

17764 shall be created or overwritten as the result of successful compilation. If the **-c** option is not  
 17765 specified, it is unspecified whether such *.o* files are created or deleted for the *file.f* operands.

17766 If there are no options that prevent link editing (such as **-c**) and all operands compile and link  
 17767 without error, the resulting executable file shall be written into the file named by the **-o** option  
 17768 (if present) or to the file **a.out**. The executable file shall be created as specified in the System  
 17769 Interfaces volume of IEEE Std 1003.1-200x, except that the file permissions shall be set to:

17770 S\_IRWXO | S\_IRWXG | S\_IRWXU

17771 and that the bits specified by the *umask* of the process shall be cleared.

17772 **OPTIONS**

17773 The *fort77* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section  
 17774 12.2, Utility Syntax Guidelines, except that:

- 17775 • The **-l** *library* operands have the format of options, but their position within a list of  
 17776 operands affects the order in which libraries are searched.
- 17777 • The order of specifying the multiple **-L** options is significant.
- 17778 • Conforming applications shall specify each option separately; that is, grouping option letters  
 17779 (for example, **-cg**) need not be recognized by all implementations.

17780 The following options shall be supported:

17781 **-c** Suppress the link-edit phase of the compilation, and do not remove any object files  
 17782 that are produced.

17783 **-g** Produce symbolic information in the object or executable files; the nature of this  
 17784 information is unspecified, and may be modified by implementation-defined  
 17785 interactions with other options.

17786 **-s** Produce object or executable files, or both, from which symbolic and other  
 17787 information not required for proper execution using the *exec* family of functions  
 17788 defined in the System Interfaces volume of IEEE Std 1003.1-200x has been removed  
 17789 (stripped). If both **-g** and **-s** options are present, the action taken is unspecified.

17790 **-o** *outfile* Use the pathname *outfile*, instead of the default **a.out**, for the executable file  
 17791 produced. If the **-o** option is present with **-c**, the result is unspecified.

17792        **-L *directory***   Change the algorithm of searching for the libraries named in **-I** operands to look in  
 17793                    the directory named by the *directory* pathname before looking in the usual places.  
 17794                    Directories named in **-L** options shall be searched in the specified order. At least  
 17795                    ten instances of this option shall be supported in a single *fort77* command  
 17796                    invocation. If a directory specified by a **-L** option contains a file named **libf.a**, the  
 17797                    results are unspecified.

17798        **-O *optlevel***   Specify the level of code optimization. If the *optlevel* option-argument is the digit  
 17799                    '0', all special code optimizations shall be disabled. If it is the digit '1', the  
 17800                    nature of the optimization is unspecified. If the **-O** option is omitted, the nature of  
 17801                    the system's default optimization is unspecified. It is unspecified whether code  
 17802                    generated in the presence of the **-O 0** option is the same as that generated when  
 17803                    **-O** is omitted. Other *optlevel* values may be supported.

17804        **-w**                Suppress warnings.

17805        Multiple instances of **-L** options can be specified.

#### 17806 OPERANDS

17807        An *operand* is either in the form of a pathname or the form **-I *library***. At least one operand of the  
 17808        pathname form shall be specified. The following operands shall be supported:

17809        ***file.f***            The pathname of a FORTRAN source file to be compiled and optionally passed to  
 17810                    the link editor. The filename operand shall be of this form if the **-c** option is used.

17811        ***file.a***            A library of object files typically produced by *ar*, and passed directly to the link  
 17812                    editor. Implementations may recognize implementation-defined suffixes other  
 17813                    than **.a** as denoting object file libraries.

17814        ***file.o***            An object file produced by *fort77 -c* and passed directly to the link editor.  
 17815                    Implementations may recognize implementation-defined suffixes other than **.o**  
 17816                    as denoting object files.

17817        The processing of other files is implementation-defined.

17818        **-I *library***        (The letter ell.) Search the library named:

17819                    *liblibrary.a*

17820                    A library is searched when its name is encountered, so the placement of a **-I**  
 17821                    operand is significant. Several standard libraries can be specified in this manner, as  
 17822                    described in the EXTENDED DESCRIPTION section. Implementations may  
 17823                    recognize implementation-defined suffixes other than **.a** as denoting libraries.

#### 17824 STDIN

17825        Not used.

#### 17826 INPUT FILES

17827        The input file shall be one of the following: a text file containing FORTRAN source code; an  
 17828        object file in the format produced by *fort77 -c*; or a library of object files, in the format produced  
 17829        by archiving zero or more object files, using *ar*. Implementations may supply additional utilities  
 17830        that produce files in these formats. Additional input files are implementation-defined.

17831        A **<tab>** encountered within the first six characters on a line of source code shall cause the  
 17832        compiler to interpret the following character as if it were the seventh character on the line (that  
 17833        is, in column 7).

17834 **ENVIRONMENT VARIABLES**

17835 The following environment variables shall affect the execution of *fort77*:

17836 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 17837 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
 17838 Internationalization Variables for the precedence of internationalization variables  
 17839 used to determine the values of locale categories.)

17840 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 17841 internationalization variables.

17842 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 17843 characters (for example, single-byte as opposed to multi-byte characters in  
 17844 arguments and input files).

17845 *LC\_MESSAGES*

17846 Determine the locale that should be used to affect the format and contents of  
 17847 diagnostic messages written to standard error.

17848 *XSI* *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

17849 *TMPDIR* Determine the pathname that should override the default directory for temporary  
 17850 files, if any.

17851 **ASYNCHRONOUS EVENTS**

17852 Default.

17853 **STDOUT**

17854 Not used.

17855 **STDERR**

17856 The standard error shall be used only for diagnostic messages. If more than one *file* operand  
 17857 ending in *.f* (or possibly other unspecified suffixes) is given, for each such file:

17858 "%s:\n", *<file>*

17859 may be written to allow identification of the diagnostic message with the appropriate input file.

17860 This utility may produce warning messages about certain conditions that do not warrant  
 17861 returning an error (non-zero) exit value.

17862 **OUTPUT FILES**

17863 Object files, listing files and executable files shall be produced in unspecified formats.

17864 **EXTENDED DESCRIPTION**17865 **Standard Libraries**

17866 The *fort77* utility shall recognize the following *-l* operand for the standard library:

17867 *-l f* This library contains all functions referenced in the ANSI X3.9-1978 standard. This  
 17868 operand shall not be required to be present to cause a search of this library.

17869 In the absence of options that inhibit invocation of the link editor, such as *-c*, the *fort77* utility  
 17870 shall cause the equivalent of a *-l f* operand to be passed to the link editor as the last *-l* operand,  
 17871 causing it to be searched after all other object files and libraries are loaded.

17872 It is unspecified whether the library *libf.a* exists as a regular file. The implementation may  
 17873 accept as *-l* operands names of objects that do not exist as regular files.

17874 **External Symbols**

17875 The FORTRAN compiler and link editor shall support the significance of external symbols up to  
17876 a length of at least 31 bytes; case folding is permitted. The action taken upon encountering  
17877 symbols exceeding the implementation-defined maximum symbol length is unspecified.

17878 The compiler and link editor shall support a minimum of 511 external symbols per source or  
17879 object file, and a minimum of 4 095 external symbols total. A diagnostic message is written to  
17880 standard output if the implementation-defined limit is exceeded; other actions are unspecified.

17881 **EXIT STATUS**

17882 The following exit values shall be returned:

17883 0 Successful compilation or link edit.

17884 >0 An error occurred.

17885 **CONSEQUENCES OF ERRORS**

17886 When *fort77* encounters a compilation error, it shall write a diagnostic to standard error and  
17887 continue to compile other source code operands. It shall return a non-zero exit status, but it is  
17888 implementation-defined whether an object module is created. If the link edit is unsuccessful, a  
17889 diagnostic message shall be written to standard error, and *fort77* shall exit with a non-zero  
17890 status.

17891 **APPLICATION USAGE**

17892 None.

17893 **EXAMPLES**

17894 The following usage example compiles **xyz.f** and creates the executable file **foo**:

```
17895 fort77 -o foo xyz.f
```

17896 The following example compiles **xyz.f** and creates the object file **xyz.o**:

```
17897 fort77 -c xyz.f
```

17898 The following example compiles **xyz.f** and creates the executable file **a.out**:

```
17899 fort77 xyz.f
```

17900 The following example compiles **xyz.f**, links it with **b.o**, and creates the executable **a.out**:

```
17901 fort77 xyz.f b.o
```

17902 **RATIONALE**

17903 The name of this utility was chosen as *fort77* to parallel the renaming of the C compiler. The  
17904 name *f77* was not chosen to avoid problems with historical implementations. The  
17905 ANSI X3.9-1978 standard was selected as a normative reference because the ISO/IEC version of  
17906 FORTRAN-77 has been superseded by the ISO/IEC 1539: 1990 standard (Fortran-90).

17907 The file inclusion and symbol definition **#define** mechanisms used by the *c99* utility were not  
17908 included in this volume of IEEE Std 1003.1-200x—even though they are commonly  
17909 implemented—since there is no requirement that the FORTRAN compiler use the C  
17910 preprocessor.

17911 The **-onetrip** option was not included in this volume of IEEE Std 1003.1-200x, even though  
17912 many historical compilers support it, because it is derived from FORTRAN-66; it is an  
17913 anachronism that should not be perpetuated.

17914 Some implementations produce compilation listings. This aspect of FORTRAN has been left  
17915 unspecified because there was controversy concerning the various methods proposed for  
17916 implementing it: a **-V** option overlapped with historical vendor practice and a naming

- 17917 convention of creating files with `.l` suffixes collided with historical *lex* file naming practice.
- 17918 There is no `-I` option in this version of this volume of IEEE Std 1003.1-200x to specify a directory  
17919 for file inclusion. An `INCLUDE` directive has been a part of the Fortran-90 discussions, but an  
17920 interface supporting that standard is not in the current scope.
- 17921 It is noted that many FORTRAN compilers produce an object module even when compilation  
17922 errors occur; during a subsequent compilation, the compiler may patch the object module rather  
17923 than recompiling all the code. Consequently, it is left to the implementor whether or not an  
17924 object file is created.
- 17925 A reference to MIL-STD-1753 was removed from an early proposal in response to a request from  
17926 the POSIX FORTRAN-binding standard developers. It was not the intention of the standard  
17927 developers to require certification of the FORTRAN compiler, and IEEE Std 1003.9-1992 does not  
17928 specify the military standard or any special preprocessing requirements. Furthermore, use of  
17929 that document would have been inappropriate for an international standard.
- 17930 The specification of optimization has been subject to changes through early proposals. At one  
17931 time, `-O` and `-N` were Booleans: optimize and do not optimize (with an unspecified default).  
17932 Some historical practice lead this to be changed to:
- 17933 `-O 0` No optimization.
- 17934 `-O 1` Some level of optimization.
- 17935 `-O n` Other, unspecified levels of optimization.
- 17936 It is not always clear whether “good code generation” is the same thing as optimization. Simple  
17937 optimizations of local actions do not usually affect the semantics of a program. The `-O 0` option  
17938 has been included to accommodate the very particular nature of scientific calculations in a  
17939 highly optimized environment; compilers make errors. Some degree of optimization is expected,  
17940 even if it is not documented here, and the ability to shut it off completely could be important  
17941 when porting an application. An implementation may treat `-O 0` as “do less than normal” if it  
17942 wishes, but this is only meaningful if any of the operations it performs can affect the semantics  
17943 of a program. It is highly dependent on the implementation whether doing less than normal is  
17944 logical. It is not the intent of the `-O 0` option to ask for inefficient code generation, but rather to  
17945 assure that any semantically visible optimization is suppressed.
- 17946 The specification of standard library access is consistent with the C compiler specification.  
17947 Implementations are not required to have `/usr/lib/libf.a`, as many historical implementations do,  
17948 but if not they are required to recognize `f` as a token.
- 17949 External symbol size limits are in normative text; conforming applications need to know these |  
17950 limits. However, the minimum maximum symbol length should be taken as a constraint on a |  
17951 conforming application, not on an implementation, and consequently the action taken for a |  
17952 symbol exceeding the limit is unspecified. The minimum size for the external symbol table was |  
17953 added for similar reasons.
- 17954 The CONSEQUENCES OF ERRORS section clearly specifies the behavior of the compiler when  
17955 compilation or link-edit errors occur. The behavior of several historical implementations was  
17956 examined, and the choice was made to be silent on the status of the executable, or `a.out`, file in  
17957 the face of compiler or linker errors. If a linker writes the executable file, then links it on disk  
17958 with `lseek()`s and `write()`s, the partially linked executable file can be left on disk and its execute  
17959 bits turned off if the link edit fails. However, if the linker links the image in memory before  
17960 writing the file to disk, it need not touch the executable file (if it already exists) because the link  
17961 edit fails. Since both approaches are historical practice, a conforming application shall rely on |  
17962 the exit status of *fort77*, rather than on the existence or mode of the executable file.

17963 The `-g` and `-s` options are not specified as mutually-exclusive. Historically these two options  
17964 have been mutually-exclusive, but because both are so loosely specified, it seemed appropriate  
17965 to leave their interaction unspecified.

17966 The requirement that conforming applications specify compiler options separately is to reserve  
17967 the multi-character option name space for vendor-specific compiler options, which are known to  
17968 exist in many historical implementations. Implementations are not required to recognize, for  
17969 example, `-gc` as if it were `-g -c`; nor are they forbidden from doing so. The SYNOPSIS shows all  
17970 of the options separately to highlight this requirement on applications.

17971 Echoing filenames to standard error is considered a diagnostic message because it would  
17972 otherwise be difficult to associate an error message with the erring file. They are described with  
17973 “may” to allow implementations to use other methods of identifying files and to parallel the  
17974 description in *c99*.

#### 17975 **FUTURE DIRECTIONS**

17976 A compilation system based on the ISO/IEC 1539:1990 standard (Fortran-90) may be considered  
17977 for a future issue; it may have a different utility name from *fort77*.

#### 17978 **SEE ALSO**

17979 *ar, asa, c99, umask*

#### 17980 **CHANGE HISTORY**

17981 First released in Issue 4.

#### 17982 **Issue 6**

17983 This utility is now marked as part of the FORTRAN Development Utilities option.

17984 The normative text is reworded to avoid use of the term “must” for application requirements.

17985 **NAME**

17986 fuser — list process IDs of all processes that have one or more files open

17987 **SYNOPSIS**

17988 xSI fuser [ -cfu ] file ...

17989

17990 **DESCRIPTION**

17991 The *fuser* utility shall write to standard output the process IDs of processes running on the local  
 17992 system that have one or more named files open. For block special devices, all processes using  
 17993 any file on that device are listed.

17994 The *fuser* utility shall write to standard error additional information about the named files  
 17995 indicating how the file is being used.

17996 Any output for processes running on remote systems that have a named file open is unspecified.

17997 A user may need appropriate privilege to invoke the *fuser* utility.

17998 **OPTIONS**

17999 The *fuser* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section  
 18000 12.2, Utility Syntax Guidelines.

18001 The following options shall be supported:

18002 **-c** The file is treated as a mount point and the utility shall report on any files open in  
 18003 the file system.

18004 **-f** The report shall be only for the named files.

18005 **-u** The user name, in parentheses, associated with each process ID written to standard  
 18006 output shall be written to standard error.

18007 **OPERANDS**

18008 The following operand shall be supported:

18009 *file* A pathname on which the file or file system is to be reported.

18010 **STDIN**

18011 Not used.

18012 **INPUT FILES**

18013 The user database.

18014 **ENVIRONMENT VARIABLES**

18015 The following environment variables shall affect the execution of *fuser*:

18016 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 18017 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
 18018 Internationalization Variables for the precedence of internationalization variables  
 18019 used to determine the values of locale categories.)

18020 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 18021 internationalization variables.

18022 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 18023 characters (for example, single-byte as opposed to multi-byte characters in  
 18024 arguments).

18025 **LC\_MESSAGES**

18026 Determine the locale that should be used to affect the format and contents of  
 18027 diagnostic messages written to standard error.

- 18028            *NLSPATH*    Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 18029 **ASYNCHRONOUS EVENTS**
- 18030            Default.
- 18031 **STDOUT**
- 18032            The *fuser* utility shall write the process ID for each process using each file given as an operand to  
18033            standard output in the following format:
- 18034            "%d", <*process\_id*>
- 18035 **STDERR**
- 18036            The *fuser* utility shall write diagnostic messages to standard error.
- 18037            The *fuser* utility also shall write the following to standard error:
- 18038            • The pathname of each named file is written followed immediately by a colon.
- 18039            • For each process ID written to standard output, the character 'c' shall be written to  
18040            standard error if the process is using the file as its current directory and the character 'r'  
18041            shall be written to standard error if the process is using the file as its root directory.  
18042            Implementations may write other alphabetic characters to indicate other uses of files.
- 18043            • When the *-u* option is specified, characters indicating the use of the file shall be followed  
18044            immediately by the user name, in parentheses, corresponding to the process' real user ID. If  
18045            the user name cannot be resolved from the process' real user ID, the process' real user ID  
18046            shall be written instead of the user name.
- 18047            When standard output and standard error are directed to the same file, the output shall be |  
18048            interleaved so that the filename appears at the start of each line, followed by the process ID and |  
18049            characters indicating the use of the file. Then, if the *-u* option is specified, the user name or user  
18050            ID for each process using that file shall be written.
- 18051            A <newline> shall be written to standard error after the last output described above for each *file*  
18052            operand.
- 18053 **OUTPUT FILES**
- 18054            None.
- 18055 **EXTENDED DESCRIPTION**
- 18056            None.
- 18057 **EXIT STATUS**
- 18058            The following exit values shall be returned:
- 18059            0    Successful completion.
- 18060            >0   An error occurred.
- 18061 **CONSEQUENCES OF ERRORS**
- 18062            Default.

18063 **APPLICATION USAGE**

18064           None.

18065 **EXAMPLES**

18066           The command:

18067           fuser -fu .

18068           writes to standard output the process IDs of processes that are using the current directory and

18069           writes to standard error an indication of how those processes are using the directory and the

18070           user names associated with the processes that are using the current directory.

18071 **RATIONALE**18072           The definition of the *fuser* utility follows existing practice.18073 **FUTURE DIRECTIONS**

18074           None.

18075 **SEE ALSO**

18076           None.

18077 **CHANGE HISTORY**

18078           First released in Issue 5.

18079 **NAME**

18080 gencat — generate a formatted message catalog

18081 **SYNOPSIS**18082 XSI gencat *catfile* *msgfile*...

18083

18084 **DESCRIPTION**

18085 The *gencat* utility shall merge the message text source files *msgfile* into a formatted message  
18086 catalog *catfile*. The file *catfile* shall be created if it does not already exist. If *catfile* does exist, its  
18087 messages shall be included in the new *catfile*. If set and message numbers collide, the new  
18088 message text defined in *msgfile* shall replace the old message text currently contained in *catfile*.

18089 **OPTIONS**

18090 None.

18091 **OPERANDS**

18092 The following operands shall be supported:

18093 *catfile* A pathname of the formatted message catalog. If '-' is specified, standard output  
18094 shall be used. The format of the message catalog produced is unspecified.

18095 *msgfile* A pathname of a message text source file. If '-' is specified for an instance of  
18096 *msgfile*, standard input shall be used. The format of message text source files is  
18097 defined in the EXTENDED DESCRIPTION section.

18098 **STDIN**18099 The standard input shall not be used unless a *msgfile* operand is specified as '-'.18100 **INPUT FILES**

18101 The input files shall be text files.

18102 **ENVIRONMENT VARIABLES**18103 The following environment variables shall affect the execution of *gencat*:

18104 *LANG* Provide a default value for the internationalization variables that are unset or null.  
18105 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
18106 Internationalization Variables for the precedence of internationalization variables  
18107 used to determine the values of locale categories.)

18108 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
18109 internationalization variables.

18110 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
18111 characters (for example, single-byte as opposed to multi-byte characters in  
18112 arguments and input files).

18113 *LC\_MESSAGES*

18114 Determine the locale that should be used to affect the format and contents of  
18115 diagnostic messages written to standard error.

18116 *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

18117 **ASYNCHRONOUS EVENTS**

18118 Default.

18119 **STDOUT**18120 The standard output shall not be used unless the *catfile* operand is specified as '-'.

18121 **STDERR**

18122 The standard error shall be used only for diagnostic messages. |

18123 **OUTPUT FILES**

18124 None.

18125 **EXTENDED DESCRIPTION**

18126 The content of a message text file shall be in the format defined as follows. Note that the fields of |  
 18127 a message text source line are separated by a single <blank>. Any other <blank>s are considered |  
 18128 as being part of the subsequent field.

18129 **\$set** *n comment*

18130 This line specifies the set identifier of the following messages until the next **\$set** or  
 18131 end-of-file appears. The *n* denotes the set identifier, which is defined as a number  
 18132 in the range [1, {NL\_SETMAX}] (see the <limits.h> header defined in the System  
 18133 Interfaces volume of IEEE Std 1003.1-200x). The application shall ensure that set  
 18134 identifiers are presented in ascending order within a single source file, but need  
 18135 not be contiguous. Any string following the set identifier shall be treated as a  
 18136 comment. If no **\$set** directive is specified in a message text source file, all messages  
 18137 shall be located in an implementation-defined default message set NL\_SETD (see  
 18138 the <nl\_types.h> header defined in the System Interfaces volume of  
 18139 IEEE Std 1003.1-200x).

18140 **\$delsset** *n comment*

18141 This line deletes message set *n* from an existing message catalog. The *n* denotes the  
 18142 set number [1, {NL\_SETMAX}]. Any string following the set number shall be  
 18143 treated as a comment.

18144 **\$ comment** A line beginning with ' \$ ' followed by a <blank> shall be treated as a comment.

18145 *m message-text*

18146 The *m* denotes the message identifier, which is defined as a number in the range [1,  
 18147 {NL\_MSGMAX}] (see the <limits.h> header defined in the System Interfaces  
 18148 volume of IEEE Std 1003.1-200x). The *message-text* shall be stored in the message  
 18149 catalog with the set identifier specified by the last **\$set** directive, and with message  
 18150 identifier *m*. If the *message-text* is empty, and a <blank> field separator is present,  
 18151 an empty string shall be stored in the message catalog. If a message source line has  
 18152 a message number, but neither a field separator nor *message-text*, the existing  
 18153 message with that number (if any) shall be deleted from the catalog. The  
 18154 application shall ensure that message identifiers are in ascending order within a  
 18155 single set, but need not be contiguous. The application shall ensure that the length  
 18156 of *message-text* is in the range [0, {NL\_TEXTMAX}] (see the <limits.h> header  
 18157 defined in the System Interfaces volume of IEEE Std 1003.1-200x).

18158 **\$quote** *n*

18159 This line specifies an optional quote character *c*, which can be used to surround  
 18160 *message-text* so that trailing spaces or null (empty) messages are visible in a  
 18161 message source line. By default, or if an empty **\$quote** directive is supplied, no  
 quoting of *message-text* shall be recognized.

18162 Empty lines in a message text source file shall be ignored. The effects of lines starting with any  
 18163 character other than those defined above are implementation-defined.

18164 Text strings can contain the special characters and escape sequences defined in the following  
 18165 table:

18166  
18167  
18168  
18169  
18170  
18171  
18172  
18173  
18174  
18175

| Description       | Symbol | Sequence |
|-------------------|--------|----------|
| <newline>         | NL(LF) | \n       |
| Horizontal tab    | HT     | \t       |
| <vertical-tab>    | VT     | \v       |
| <backspace>       | BS     | \b       |
| <carriage-return> | CR     | \r       |
| <form-feed>       | FF     | \f       |
| Backslash         | \      | \\       |
| Bit pattern       | ddd    | \ddd     |

18176 The escape sequence "\ddd" consists of backslash followed by one, two, or three octal digits,  
18177 which shall be taken to specify the value of the desired character. If the character following a  
18178 backslash is not one of those specified, the backslash shall be ignored.

18179 Backslash ('\') followed by a <newline> is also used to continue a string on the following line.  
18180 Thus, the following two lines describe a single message string:

18181 1 This line continues \  
18182 to the next line

18183 which shall be equivalent to:

18184 1 This line continues to the next line

#### 18185 EXIT STATUS

18186 The following exit values shall be returned:

18187 0 Successful completion.

18188 >0 An error occurred.

#### 18189 CONSEQUENCES OF ERRORS

18190 Default.

#### 18191 APPLICATION USAGE

18192 Message catalogs produced by *gencat* are binary encoded, meaning that their portability cannot  
18193 be guaranteed between different types of machine. Thus, just as C programs need to be  
18194 recompiled for each type of machine, so message catalogs must be recreated via *gencat*.

#### 18195 EXAMPLES

18196 None.

#### 18197 RATIONALE

18198 None.

#### 18199 FUTURE DIRECTIONS

18200 None.

#### 18201 SEE ALSO

18202 *iconv*, the System Interfaces volume of IEEE Std 1003.1-200x, <limits.h>

#### 18203 CHANGE HISTORY

18204 First released in Issue 3.

#### 18205 Issue 6

18206 The normative text is reworded to avoid use of the term “must” for application requirements.

## 18207 NAME

18208 `get` — get a version of an SCCS file (**DEVELOPMENT**)

## 18209 SYNOPSIS

```
18210 xSI get [-begkmnlpst][-c cutoff][-i list][-r SID][-x list] file...
```

18211

## 18212 DESCRIPTION

18213 The `get` utility shall generate a text file from each named SCCS *file* according to the specifications  
18214 given by its options.

18215 The generated text shall normally be written into a file called the **g-file** whose name is derived  
18216 from the SCCS filename by simply removing the leading "s. ".

## 18217 OPTIONS

18218 The `get` utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2,  
18219 Utility Syntax Guidelines.

18220 The following options shall be supported:

18221 **-r SID** Indicate the SCCS Identification String (SID) of the version (delta) of an SCCS file  
18222 to be retrieved. The table shows, for the most useful cases, what version of an  
18223 SCCS file is retrieved (as well as the SID of the version to be eventually created by  
18224 *delta* if the **-e** option is also used), as a function of the SID specified.

18225 **-c cutoff** Indicate the *cutoff* date-time, in the form:

```
18226 YY[MM[DD[HH[MM[SS]]]]]
```

18227 For the YY component, values in the range [69,99] shall refer to years 1969 to 1999  
18228 inclusive, and values in the range [00,68] shall refer to years 2000 to 2068 inclusive.

18229 **Note:** It is expected that in a future version of IEEE Std 1003.1-200x the default  
18230 century inferred from a 2-digit year will change. (This would apply to all  
18231 commands accepting a 2-digit year as input.)

18232 No changes (deltas) to the SCCS file that were created after the specified *cutoff*  
18233 date-time shall be included in the generated text file. Units omitted from the date-  
18234 time default to their maximum possible values; for example, **-c 7502** is equivalent  
18235 to **-c 750228235959**.

18236 Any number of non-numeric characters may separate the various 2-digit pieces of  
18237 the *cutoff* date-time. This feature allows the user to specify a *cutoff* date in the form:  
18238 **-c "77/2/2 9:22:25"**.

18239 **-e** Indicate that the `get` is for the purpose of editing or making a change (delta) to the  
18240 SCCS file via a subsequent use of *delta*. The **-e** option used in a `get` for a particular  
18241 version (SID) of the SCCS file shall prevent further `get` commands from editing on  
18242 the same SID until *delta* is executed or the **j** (joint edit) flag is set in the SCCS file.  
18243 Concurrent use of `get -e` for different SIDs is always allowed.

18244 If the **g-file** generated by `get` with a **-e** option is accidentally ruined in the process  
18245 of editing, it may be regenerated by re-executing the `get` command with the **-k**  
18246 option in place of the **-e** option.

18247 SCCS file protection specified via the ceiling, floor, and authorized user list stored  
18248 in the SCCS file shall be enforced when the **-e** option is used.

18249 **-b** Use with the **-e** option to indicate that the new delta should have an SID in a new  
18250 branch as shown in the table below. This option shall be ignored if the **b** flag is not  
18251 present in the file or if the retrieved delta is not a leaf delta. (A leaf delta is one that

18252 has no successors on the SCCS file tree.)

18253 **Note:** A branch delta may always be created from a non-leaf delta.

18254 **-i list** Indicate a *list* of deltas to be included (forced to be applied) in the creation of the  
 18255 generated file. The *list* has the following syntax:

18256 `<list> ::= <range> | <list> , <range>`  
 18257 `<range> ::= SID | SID - SID`

18258 SID, the SCCS Identification of a delta, may be in any form shown in the "SID  
 18259 Specified" column of the table in the EXTENDED DESCRIPTION section, except |  
 18260 that the result of supplying a partial SID is unspecified. A diagnostic message shall |  
 18261 be written if the first SID in the range is not an ancestor of the second SID in the |  
 18262 range. |

18263 **-x list** Indicate a *list* of deltas to be excluded (forced not to be applied) in the creation of  
 18264 the generated file. See the **-i** option for the *list* format.

18265 **-k** Suppress replacement of identification keywords (see below) in the retrieved text  
 18266 by their value. The **-k** option shall be implied by the **-e** option. |

18267 **-l** Write a delta summary into an **l-file**.

18268 **-L** Write a delta summary to standard output. All informative output that normally is  
 18269 written to standard output shall be written to standard error instead, unless the **-s**  
 18270 option is used, in which case it shall be suppressed.

18271 **-p** Write the text retrieved from the SCCS file to the standard output. No **g-file** shall  
 18272 be created. All informative output that normally goes to the standard output shall  
 18273 go to standard error instead, unless the **-s** option is used, in which case it shall |  
 18274 disappear. |

18275 **-s** Suppress all informative output normally written to standard output. However,  
 18276 fatal error messages (which shall always be written to the standard error) shall |  
 18277 remain unaffected. |

18278 **-m** Precede each text line retrieved from the SCCS file by the SID of the delta that |  
 18279 inserted the text line in the SCCS file. The format shall be: |

18280 `"%s\t%s", <SID>, <text line>`

18281 **-n** Precede each generated text line with the **%M%** identification keyword value (see |  
 18282 below). The format shall be: |

18283 `"%s\t%s", <%M% value>, <text line>` |

18284 When both the **-m** and **-n** options are used, the `<text line>` shall be replaced by the |  
 18285 **-m** option-generated format. |

18286 **-g** Suppress the actual retrieval of text from the SCCS file. It is primarily used to  
 18287 generate an **l-file**, or to verify the existence of a particular SID.

18288 **-t** Use to access the most recently created (top) delta in a given release (for example,  
 18289 **-r 1**), or release and level (for example, **-r 1.2**).

## 18290 OPERANDS

18291 The following operands shall be supported:

18292 **file** A pathname of an existing SCCS file or a directory. If *file* is a directory, the *get*  
 18293 utility shall behave as though each file in the directory were specified as a named  
 18294 file, except that non-SCCS files (last component of the pathname does not begin

- 18295 with **s**.) and unreadable files shall be silently ignored.
- 18296 If exactly one *file* operand appears, and it is `'-'`, the standard input shall be read;  
 18297 each line of the standard input is taken to be the name of an SCCS file to be  
 18298 processed. Non-SCCS files and unreadable files shall be silently ignored.
- 18299 **STDIN**
- 18300 The standard input shall be a text file used only if the *file* operand is specified as `'-'`. Each line  
 18301 of the text file shall be interpreted as an SCCS pathname.
- 18302 **INPUT FILES**
- 18303 The SCCS files shall be files of an unspecified format. |
- 18304 **ENVIRONMENT VARIABLES**
- 18305 The following environment variables shall affect the execution of *get*:
- 18306 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 18307 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
 18308 Internationalization Variables for the precedence of internationalization variables  
 18309 used to determine the values of locale categories.)
- 18310 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 18311 internationalization variables.
- 18312 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 18313 characters (for example, single-byte as opposed to multi-byte characters in  
 18314 arguments and input files).
- 18315 **LC\_MESSAGES**
- 18316 Determine the locale that should be used to affect the format and contents of  
 18317 diagnostic messages written to standard error, and informative messages written  
 18318 to standard output (or standard error, if the `-p` option is used).
- 18319 **NLSPATH** Determine the location of message catalogs for the processing of *LC\_MESSAGES*. |
- 18320 **TZ** Determine the timezone in which the times and dates are written in the SCCS file |  
 18321 are evaluated. If the *TZ* variable is unset or NULL, an unspecified system default |  
 18322 timezone is used. |
- 18323 **ASYNCHRONOUS EVENTS**
- 18324 Default.
- 18325 **STDOUT**
- 18326 For each file processed, *get* shall write to standard output the SID being accessed and the number  
 18327 of lines retrieved from the SCCS file, in the following format:
- 18328 `"%s\n%d lines\n", <SID>, <number of lines>`
- 18329 If the `-e` option is used, the SID of the delta to be made shall appear after the SID accessed and  
 18330 before the number of lines generated, in the POSIX locale:
- 18331 `"%s\nnew delta %s\n%d lines\n", <SID accessed>,  
 18332 <SID to be made>, <number of lines>` |
- 18333 If there is more than one named file or if a directory or standard input is named, each pathname |  
 18334 shall be written before each of the lines shown in one of the preceding formats:
- 18335 `"\n%s:\n", <pathname>`
- 18336 If the `-L` option is used, a delta summary shall be written following the format specified below  
 18337 for **l-files**.

18338 If the **-i** option is used, included deltas shall be listed following the notation, in the POSIX locale: |  
 18339 "Included:\n" |

18340 If the **-x** option is used, excluded deltas shall be listed following the notation, in the POSIX |  
 18341 locale: |

18342 "Excluded:\n" |

18343 If the **-p** or **-L** options are specified, the standard output shall consist of the text retrieved from |  
 18344 the SCCS file. |

18345 **STDERR** |

18346 The standard error shall be used only for diagnostic messages, except if the **-p** or **-L** options are |  
 18347 specified, it shall include all informative messages normally sent to standard output. |

18348 **OUTPUT FILES** |

18349 Several auxiliary files may be created by *get*. These files are known generically as the **g-file**, **l-** |  
 18350 **file**, **p-file**, and **z-file**. The letter before the hyphen is called the *tag*. An auxiliary filename shall |  
 18351 be formed from the SCCS filename: the application shall ensure that the last component of all |  
 18352 SCCS filenames is of the form *s.module-name*; the auxiliary files shall be named by replacing the |  
 18353 leading *s* with the tag. The **g-file** shall be an exception to this scheme: the **g-file** is named by |  
 18354 removing the *s*. prefix. For example, for *s.xyz.c*, the auxiliary filenames would be *xyz.c*, *l.xyz.c*, |  
 18355 *p.xyz.c*, and *z.xyz.c*, respectively. |

18356 The **g-file**, which contains the generated text, shall be created in the current directory (unless the |  
 18357 **-p** option is used). A **g-file** shall be created in all cases, whether or not any lines of text were |  
 18358 generated by the *get*. It shall be owned by the real user. If the **-k** option is used or implied, the |  
 18359 **g-file** shall be writable by the owner only (read-only for everyone else); otherwise, it shall be |  
 18360 read-only. Only the real user need have write permission in the current directory. |

18361 The **l-file** shall contain a table showing which deltas were applied in generating the retrieved |  
 18362 text. The **l-file** shall be created in the current directory if the **-l** option is used; it shall be read- |  
 18363 only and it is owned by the real user. Only the real user need have write permission in the |  
 18364 current directory. |

18365 Lines in the **l-file** shall have the following format: |

18366 "%c%c%cΔ%s\t%sΔ%s\n", <code1>, <code2>, <code3>, |  
 18367 <SID>, <date-time>, <login> |

18368 where the entries are: |

18369 <code1> A <space> if the delta was applied; '\*' otherwise. |

18370 <code2> A <space> if the delta was applied or was not applied and ignored; '\*' if the delta |  
 18371 was not applied and was not ignored. |

18372 <code3> A character indicating a special reason why the delta was or was not applied: |

18373 **I** Included. |

18374 **X** Excluded. |

18375 **C** Cut off (by a **-c** option). |

18376 <date-time> Date and time (using the format of the *date* utility's %Y/%m/%d %T conversion |  
 18377 specification format) of creation. |

18378 <login> Login name of person who created *delta*. |

18379           The comments and MR data shall follow on subsequent lines, indented one <tab>. A blank line |  
18380 shall terminate each entry. |

18381           The **p-file** shall be used to pass information resulting from a *get* with a **-e** option along to *delta*. |  
18382 Its contents shall also be used to prevent a subsequent execution of *get* with a **-e** option for the |  
18383 same SID until *delta* is executed or the joint edit flag, **j**, is set in the SCCS file. The **p-file** shall be |  
18384 created in the directory containing the SCCS file and the application shall ensure that the |  
18385 effective user has write permission in that directory. It shall be writable by owner only, and |  
18386 owned by the effective user. Each line in the **p-file** shall have the following format: |

```
18387           "%sΔ%sΔ%sΔ%s%s\n", <g-file SID>,
18388                    <SID of new delta>, <login-name of real user>,
18389                    <date-time>, <i-value>, <x-value>
```

18390           where <i-value> uses the format " " if no **-i** option was specified, and shall use the format: |

```
18391           "Δ-i%s", <-i option option-argument>
```

18392           if a **-i** option was specified and <x-value> uses the format " " if no **-x** option was specified, and |  
18393 shall use the format: |

```
18394           "Δ-x%s", <-x option option-argument>
```

18395           if a **-x** option was specified. There can be an arbitrary number of lines in the **p-file** at any time; |  
18396 no two lines shall have the same new delta SID. |

18397           The **z-file** shall serve as a lock-out mechanism against simultaneous updates. Its contents shall |  
18398 be the binary process ID of the command (that is, *get*) that created it. The **z-file** shall be created |  
18399 in the directory containing the SCCS file for the duration of *get*. The same protection restrictions |  
18400 as those for the **p-file** shall apply for the **z-file**. The **z-file** shall be created read-only. |

## 18401 EXTENDED DESCRIPTION

| Determination of SCCS Identification String |                       |                     |                                                |                               |                |
|---------------------------------------------|-----------------------|---------------------|------------------------------------------------|-------------------------------|----------------|
| SID*<br>Specified                           | -b Keyletter<br>Used† | Other<br>Conditions | SID<br>Retrieved                               | SID of Delta<br>to be Created |                |
| 18402                                       | none‡                 | no                  | R defaults to mR                               | mR.mL                         | mR.(mL+1)      |
| 18403                                       | none‡                 | yes                 | R defaults to mR                               | mR.mL                         | mR.mL.(mB+1).1 |
| 18404                                       | R                     | no                  | R > mR                                         | mR.mL                         | R.1***         |
| 18405                                       | R                     | no                  | R = mR                                         | mR.mL                         | mR.(mL+1)      |
| 18406                                       | R                     | yes                 | R > mR                                         | mR.mL                         | mR.mL.(mB+1).1 |
| 18407                                       | R                     | yes                 | R = mR                                         | mR.mL                         | mR.mL.(mB+1).1 |
| 18408                                       | R                     | -                   | R < mR and<br>R does not exist                 | hR.mL**                       | hR.mL.(mB+1).1 |
| 18409                                       | R                     | -                   | Trunk successor in release > R<br>and R exists | R.mL                          | R.mL.(mB+1).1  |
| 18410                                       | R.L                   | no                  | No trunk successor                             | R.L                           | R.(L+1)        |
| 18411                                       | R.L                   | yes                 | No trunk successor                             | R.L                           | R.L.(mB+1).1   |
| 18412                                       | R.L                   | -                   | Trunk successor<br>in release ≥ R              | R.L                           | R.L.(mB+1).1   |
| 18413                                       | R.L.B                 | no                  | No branch successor                            | R.L.B.mS                      | R.L.B.(mS+1)   |
| 18414                                       | R.L.B                 | yes                 | No branch successor                            | R.L.B.mS                      | R.L.(mB+1).1   |
| 18415                                       | R.L.B.S               | no                  | No branch successor                            | R.L.B.S                       | R.L.B.(S+1)    |
| 18416                                       | R.L.B.S               | yes                 | No branch successor                            | R.L.B.S                       | R.L.(mB+1).1   |
| 18417                                       | R.L.B.S               | -                   | Branch successor                               | R.L.B.S                       | R.L.(mB+1).1   |

18424 \* R, L, B, and S are the release, level, branch, and sequence components of the SID,  
 18425 respectively; m means maximum. Thus, for example, R.mL means “the maximum level  
 18426 number within release R”; R.L.(mB+1).1 means “the first sequence number on the new  
 18427 branch (that is, maximum branch number plus one) of level L within release R”. Note  
 18428 that if the SID specified is of the form R.L, R.L.B, or R.L.B.S, each of the specified  
 18429 components shall exist.

18430 \*\* hR is the highest existing release that is lower than the specified, nonexistent, release R.

18431 \*\*\* This is used to force creation of the first delta in a new release.

18432 † The -b option is effective only if the b flag is present in the file. An entry of ‘-’ means  
 18433 “irrelevant”.

18434 ‡ This case applies if the d (default SID) flag is not present in the file. If the d flag is  
 18435 present in the file, then the SID obtained from the d flag is interpreted as if it had been  
 18436 specified on the command line. Thus, one of the other cases in this table applies.

18437 **System Date and Time**

18438 When a **g-file** is generated, the creation time of deltas in the SCCS file may be taken into  
 18439 account. If any of these times are apparently in the future, the behavior is unspecified.

18440 **Identification Keywords**

18441 Identifying information shall be inserted into the text retrieved from the SCCS file by replacing  
 18442 identification keywords with their value wherever they occur. The following keywords may be  
 18443 used in the text stored in an SCCS file:

18444 **%M%** Module name: either the value of the **m** flag in the file, or if absent, the name of the  
 18445 SCCS file with the leading **s.** removed.

18446 **%I%** SCCS identification (SID) (**%R%.%L%** or **%R%.%L%.%B%.%S%**) of the retrieved  
 18447 text.

18448 **%R%** Release.

18449 **%L%** Level.

18450 **%B%** Branch.

18451 **%S%** Sequence.

18452 **%D%** Current date (**YY/MM/DD**).

18453 **%H%** Current date (**MM/DD/YY**).

18454 **%T%** Current time (**HH:MM:SS**).

18455 **%E%** Date newest applied delta was created (**YY/MM/DD**).

18456 **%G%** Date newest applied delta was created (**MM/DD/YY**).

18457 **%U%** Time newest applied delta was created (**HH:MM:SS**).

18458 **%Y%** Module type: value of the **t** flag in the SCCS file.

18459 **%F%** SCCS filename.

18460 **%P%** SCCS absolute pathname.

18461 **%Q%** The value of the **q** flag in the file.

18462 **%C%** Current line number. This keyword is intended for identifying messages output by  
 18463 the program, such as "this should not have happened" type errors. It is not  
 18464 intended to be used on every line to provide sequence numbers.

18465 **%Z%** The four-character string "@(#)" recognizable by *what*.

18466 **%W%** A shorthand notation for constructing *what* strings:

18467 `%W%=%Z%%M%<tab>%I%`

18468 **%A%** Another shorthand notation for constructing *what* strings:

18469 `%A%=%Z%%Y%%M%%I%%Z%`

18470 **EXIT STATUS**

18471 The following exit values shall be returned:

18472 **0** Successful completion.

18473 **>0** An error occurred.

18474 **CONSEQUENCES OF ERRORS**

18475 Default.

18476 **APPLICATION USAGE**

18477 Problems can arise if the system date and time have been modified (for example, put forward |  
18478 and then back again, or unsynchronized clocks across a network) and can also arise when |  
18479 different values of the *TZ* environment variable are used. |

18480 Problems of a similar nature can also arise for the operation of the *delta* utility, which compares |  
18481 the previous file body against the working file as part of its normal operation. |

18482 **EXAMPLES**

18483 None.

18484 **RATIONALE**

18485 None.

18486 **FUTURE DIRECTIONS**18487 The **-lp** option may be withdrawn in a future issue.18488 **SEE ALSO**18489 *admin, delta, prs, what*18490 **CHANGE HISTORY**

18491 First released in Issue 2.

18492 **Issue 5**

18493 Correction to the first format string in STDOUT.

18494 The interpretation of the *YY* component of the **-c cutoff** argument is noted.18495 **Issue 6**18496 The obsolescent SYNOPSIS is removed, removing the **-lp** option.

18497 The Open Group Corrigendum U025/5 is applied, correcting text in the OPTIONS section.

18498 The normative text is reworded to avoid use of the term “must” for application requirements.

18499 The normative text is reworded to emphasize the term “shall” for implementation requirements.

18500 The Open Group Corrigendum U048/1 is applied. |

18501 The EXTENDED DESCRIPTION section is updated to make partial SID handling unspecified, |  
18502 reflecting common usage, and to clarify SID ranges as per The Open Group Base Resolution |  
18503 bwg2001-007. |

18504 The Open Group Interpretation PIN4C.00014 is applied. |

18505 New text is added to the EXTENDED DESCRIPTION and APPLICATION USAGE sections |  
18506 regarding how the system date and time may be taken into account, and the *TZ* environment |  
18507 variable is added to the ENVIRONMENT VARIABLES section as per The Open Group Base |  
18508 Resolution bwg2001-007. |

18509 **NAME**

18510 `getconf` — get configuration values

18511 **SYNOPSIS**

18512 `getconf` [ `-v` *specification* ] *system\_var*

18513 `getconf` [ `-v` *specification* ] *path\_var pathname*

18514 **DESCRIPTION**

18515 In the first synopsis form, the *getconf* utility shall write to the standard output the value of the  
18516 variable specified by the *system\_var* operand.

18517 In the second synopsis form, the *getconf* utility shall write to the standard output the value of the  
18518 variable specified by the *path\_var* operand for the path specified by the *pathname* operand.

18519 The value of each configuration variable shall be determined as if it were obtained by calling the  
18520 function from which it is defined to be available by this volume of IEEE Std 1003.1-200x or by the  
18521 System Interfaces volume of IEEE Std 1003.1-200x (see the OPERANDS section). The value shall  
18522 reflect conditions in the current operating environment.

18523 **OPTIONS**

18524 The *getconf* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section  
18525 12.2, Utility Syntax Guidelines.

18526 The following option shall be supported:

18527 `-v` *specification*

18528 Indicate a specific specification and version for which configuration variables shall  
18529 be determined. If this option is not specified, the values returned correspond to an  
18530 implementation default conforming compilation environment.

18531 If the command:

18532 `getconf` `_POSIX_V6_ILP32_OFF32`

18533 does not write "`-1\n`" or "`undefined\n`" to standard output, then commands of  
18534 the form:

18535 `getconf` `-v` `POSIX_V6_ILP32_OFF32` ...

18536 determine values for configuration variables corresponding to the  
18537 `POSIX_V6_ILP32_OFF32` compilation environment specified in *c99* (on page 2413),  
18538 EXTENDED DESCRIPTION.

18539 If the command:

18540 `getconf` `_POSIX_V6_ILP32_OFFBIG`

18541 does not write "`-1\n`" or "`undefined\n`" to standard output, then commands of  
18542 the form:

18543 `getconf` `-v` `POSIX_V6_ILP32_OFFBIG` ...

18544 determine values for configuration variables corresponding to the  
18545 `POSIX_V6_ILP32_OFFBIG` compilation environment specified in *c99* (on page  
18546 2413), EXTENDED DESCRIPTION.

18547 If the command:

18548 `getconf` `_POSIX_V6_LP64_OFF64`

18549 does not write "`-1\n`" or "`undefined\n`" to standard output, then commands of  
18550 the form:

18551 `getconf -v POSIX_V6_LP64_OFF64 ...`

18552 determine values for configuration variables corresponding to the  
18553 `POSIX_V6_LP64_OFF64` compilation environment specified in *c99* (on page 2413),  
18554 EXTENDED DESCRIPTION.

18555 If the command:

18556 `getconf _POSIX_V6_LPBIG_OFFBIG`

18557 does not write "-1\n" or "undefined\n" to standard output, then commands of  
18558 the form:

18559 `getconf -v POSIX_V6_LPBIG_OFFBIG ...`

18560 determine values for configuration variables corresponding to the  
18561 `POSIX_V6_LPBIG_OFFBIG` compilation environment specified in *c99* (on page  
18562 2413), EXTENDED DESCRIPTION.

### 18563 OPERANDS

18564 The following operands shall be supported:

18565 *path\_var* A name of a configuration variable. All of the variables in the *pathconf()* function  
18566 defined in the System Interfaces volume of IEEE Std 1003.1-200x are supported and  
18567 the implementation may add other local variables.

18568 *pathname* A pathname for which the variable specified by *path\_var* is to be determined.

18569 *system\_var* A name of a configuration variable. All of the variables in the *confstr()* and  
18570 *sysconf()* functions defined in the System Interfaces volume of  
18571 IEEE Std 1003.1-200x shall be supported and the implementation may add other  
18572 local values.

18573 When the symbol listed in the first column of the following table is used as the  
18574 *system\_var* operand, *getconf* yields the same value as *confstr()* when called with the  
18575 value in the second column:

| <i>system_var</i>                     | <i>confstr()</i> Name Value         |
|---------------------------------------|-------------------------------------|
| 18578 PATH                            | _CS_PATH                            |
| 18579 POSIX_V6_ILP32_OFF32_CFLAGS     | _CS_POSIX_V6_ILP32_OFF32_CFLAGS     |
| 18580 POSIX_V6_ILP32_OFF32_LDFLAGS    | _CS_POSIX_V6_ILP32_OFF32_LDFLAGS    |
| 18581 POSIX_V6_ILP32_OFF32_LIBS       | _CS_POSIX_V6_ILP32_OFF32_LIBS       |
| 18582 POSIX_V6_ILP32_OFF32_LINTFLAGS  | _CS_POSIX_V6_ILP32_OFF32_LINTFLAGS  |
| 18583 POSIX_V6_ILP32_OFFBIG_CFLAGS    | _CS_POSIX_V6_ILP32_OFFBIG_CFLAGS    |
| 18584 POSIX_V6_ILP32_OFFBIG_LDFLAGS   | _CS_POSIX_V6_ILP32_OFFBIG_LDFLAGS   |
| 18585 POSIX_V6_ILP32_OFFBIG_LIBS      | _CS_POSIX_V6_ILP32_OFFBIG_LIBS      |
| 18586 POSIX_V6_ILP32_OFFBIG_LINTFLAGS | _CS_POSIX_V6_ILP32_OFFBIG_LINTFLAGS |
| 18587 POSIX_V6_LP64_OFF64_CFLAGS      | _CS_POSIX_V6_LP64_OFF64_CFLAGS      |
| 18588 POSIX_V6_LP64_OFF64_LDFLAGS     | _CS_POSIX_V6_LP64_OFF64_LDFLAGS     |
| 18589 POSIX_V6_LP64_OFF64_LIBS        | _CS_POSIX_V6_LP64_OFF64_LIBS        |
| 18590 POSIX_V6_LP64_OFF64_LINTFLAGS   | _CS_POSIX_V6_LP64_OFF64_LINTFLAGS   |
| 18591 POSIX_V6_LPBIG_OFFBIG_CFLAGS    | _CS_POSIX_V6_LPBIG_OFFBIG_CFLAGS    |
| 18592 POSIX_V6_LPBIG_OFFBIG_LDFLAGS   | _CS_POSIX_V6_LPBIG_OFFBIG_LDFLAGS   |
| 18593 POSIX_V6_LPBIG_OFFBIG_LIBS      | _CS_POSIX_V6_LPBIG_OFFBIG_LIBS      |

18594

18595

18596

18597

18598 XSI

18599

18600

18601

18602

18603

18604

18605

18606

18607

18608

18609

18610

18611

18612

18613

| <i>system_var</i>                    | <i>confstr() Name Value</i>         |
|--------------------------------------|-------------------------------------|
| POSIX_V6_LPBIG_OFFBIG_LINTFLAGS      | _CS_POSIX_V6_LPBIG_OFFBIG_LINTFLAGS |
| POSIX_V6_WIDTH_RESTRICTED_ENVS       | _CS_POSIX_V6_WIDTH_RESTRICTED_ENVS  |
| XBS5_ILP32_OFF32_CFLAGS (LEGACY)     | _CS_XBS5_ILP32_OFF32_CFLAGS         |
| XBS5_ILP32_OFF32_LDFLAGS (LEGACY)    | _CS_XBS5_ILP32_OFF32_LDFLAGS        |
| XBS5_ILP32_OFF32_LIBS (LEGACY)       | _CS_XBS5_ILP32_OFF32_LIBS           |
| XBS5_ILP32_OFF32_LINTFLAGS (LEGACY)  | _CS_XBS5_ILP32_OFF32_LINTFLAGS      |
| XBS5_ILP32_OFFBIG_CFLAGS (LEGACY)    | _CS_XBS5_ILP32_OFFBIG_CFLAGS        |
| XBS5_ILP32_OFFBIG_LDFLAGS (LEGACY)   | _CS_XBS5_ILP32_OFFBIG_LDFLAGS       |
| XBS5_ILP32_OFFBIG_LIBS (LEGACY)      | _CS_XBS5_ILP32_OFFBIG_LIBS          |
| XBS5_ILP32_OFFBIG_LINTFLAGS (LEGACY) | _CS_XBS5_ILP32_OFFBIG_LINTFLAGS     |
| XBS5_LP64_OFF64_CFLAGS (LEGACY)      | _CS_XBS5_LP64_OFF64_CFLAGS          |
| XBS5_LP64_OFF64_LDFLAGS (LEGACY)     | _CS_XBS5_LP64_OFF64_LDFLAGS         |
| XBS5_LP64_OFF64_LIBS (LEGACY)        | _CS_XBS5_LP64_OFF64_LIBS            |
| XBS5_LP64_OFF64_LINTFLAGS (LEGACY)   | _CS_XBS5_LP64_OFF64_LINTFLAGS       |
| XBS5_LPBIG_OFFBIG_CFLAGS (LEGACY)    | _CS_XBS5_LPBIG_OFFBIG_CFLAGS        |
| XBS5_LPBIG_OFFBIG_LDFLAGS (LEGACY)   | _CS_XBS5_LPBIG_OFFBIG_LDFLAGS       |
| XBS5_LPBIG_OFFBIG_LIBS (LEGACY)      | _CS_XBS5_LPBIG_OFFBIG_LIBS          |
| XBS5_LPBIG_OFFBIG_LINTFLAGS (LEGACY) | _CS_XBS5_LPBIG_OFFBIG_LINTFLAGS     |

18614 **STDIN**

18615 Not used.

18616 **INPUT FILES**

18617 None.

18618 **ENVIRONMENT VARIABLES**18619 The following environment variables shall affect the execution of *getconf*:18620 *LANG*

Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)

18624 *LC\_ALL*

If set to a non-empty string value, override the values of all the other internationalization variables.

18625

18626 *LC\_CTYPE*

Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

18627

18628

18629 *LC\_MESSAGES*

Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

18630

18631

18632 XSI

*NLSPATH*

Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

18633 **ASYNCHRONOUS EVENTS**

18634 Default.

18635 **STDOUT**

If the specified variable is defined on the system and its value is described to be available from the *confstr()* function defined in the System Interfaces volume of IEEE Std 1003.1-200x, its value shall be written in the following format:

18638

18639

"%s\n", &lt;value&gt;

18640 Otherwise, if the specified variable is defined on the system, its value shall be written in the  
18641 following format:

18642 "%d\n", <value>

18643 If the specified variable is valid, but is undefined on the system, *getconf* shall write using the  
18644 following format:

18645 "undefined\n"

18646 If the variable name is invalid or an error occurs, nothing shall be written to standard output.

#### 18647 **STDERR**

18648 The standard error shall be used only for diagnostic messages. |

#### 18649 **OUTPUT FILES**

18650 None.

#### 18651 **EXTENDED DESCRIPTION**

18652 None.

#### 18653 **EXIT STATUS**

18654 The following exit values shall be returned:

18655 0 The specified variable is valid and information about its current state was written  
18656 successfully.

18657 >0 An error occurred.

#### 18658 **CONSEQUENCES OF ERRORS**

18659 Default.

#### 18660 **APPLICATION USAGE**

18661 None.

#### 18662 **EXAMPLES**

18663 The following example illustrates the value of {NGROUPS\_MAX}:

18664 `getconf NGROUPS_MAX`

18665 The following example illustrates the value of {NAME\_MAX} for a specific directory:

18666 `getconf NAME_MAX /usr`

18667 The following example shows how to deal more carefully with results that might be unspecified:

```
18668 if value=$(getconf PATH_MAX /usr); then
18669     if [ "$value" = "undefined" ]; then
18670         echo PATH_MAX in /usr is infinite.
18671     else
18672         echo PATH_MAX in /usr is $value.
18673     fi
18674 else
18675     echo Error in getconf.
18676 fi
```

18677 Note that:

18678 `sysconf(_SC_POSIX_C_BIND);`

18679 and:

18680 `system("getconf POSIX2_C_BIND");`

18681 in a C program could give different answers. The `sysconf()` call supplies a value that corresponds  
 18682 to the conditions when the program was either compiled or executed, depending on the  
 18683 implementation; the `system()` call to `getconf` always supplies a value corresponding to conditions  
 18684 when the program is executed.

#### 18685 RATIONALE

18686 The original need for this utility, and for the `confstr()` function, was to provide a way of finding  
 18687 the configuration-defined default value for the `PATH` environment variable. Since `PATH` can be  
 18688 modified by the user to include directories that could contain utilities replacing the standard  
 18689 utilities, shell scripts need a way to determine the system-supplied `PATH` environment variable  
 18690 value that contains the correct search path for the standard utilities. It was later suggested that  
 18691 access to the other variables described in this volume of IEEE Std 1003.1-200x could also be  
 18692 useful to applications.

18693 This functionality of `getconf` would not be adequately subsumed by another command such as:

18694 `grep var /etc/conf`

18695 because such a strategy would provide correct values for neither those variables that can vary at  
 18696 runtime, nor those that can vary depending on the path.

18697 Early proposal versions of `getconf` specified exit status 1 when the specified variable was valid,  
 18698 but not defined on the system. The output string "undefined" is now used to specify this case  
 18699 with exit code 0 because so many things depend on an exit code of zero when an invoked utility  
 18700 is successful.

#### 18701 FUTURE DIRECTIONS

18702 None.

#### 18703 SEE ALSO

18704 `c99`, the System Interfaces volume of IEEE Std 1003.1-200x, `confstr()`, `pathconf()`, `sysconf()`

#### 18705 CHANGE HISTORY

18706 First released in Issue 4.

#### 18707 Issue 5

18708 In the OPERANDS section:

- 18709 • {NL\_MAX} is changed to {NL\_NMAX}.
- 18710 • Entries beginning NL\_ are deleted from the list of standard configuration variables.
- 18711 • The list of variables previously marked UX is merged with the list marked EX.
- 18712 • Operands are added to support new Option Groups.
- 18713 • Operands are added so that `getconf` can determine supported programming environments.

#### 18714 Issue 6

18715 The Open Group Corrigendum U029/4 is applied, correcting the example command in the last  
 18716 paragraph of the OPTIONS section.

18717 The following new requirements on POSIX implementations derive from alignment with the  
 18718 Single UNIX Specification:

- 18719 • Operands are added to determine supported programming environments.

18720 This reference page is updated for alignment with the ISO/IEC 9899:1999 standard. Specifically,  
 18721 new macros for `c99` programming environments are introduced.

18722

XSI marked *system\_var* (XBS5\_\*) values are marked LEGACY.

18723 **NAME**

18724           getopts — parse utility options

18725 **SYNOPSIS**18726           getopts *optstring name* [*arg...*]18727 **DESCRIPTION**

18728           The *getopts* utility shall retrieve options and option-arguments from a list of parameters. It shall  
 18729 support the Utility Syntax Guidelines 3 to 10, inclusive, described in the Base Definitions volume  
 18730 of IEEE Std 1003.1-200x, Section 12.2, Utility Syntax Guidelines.

18731           Each time it is invoked, the *getopts* utility shall place the value of the next option in the shell  
 18732 variable specified by the *name* operand and the index of the next argument to be processed in the  
 18733 shell variable *OPTIND*. Whenever the shell is invoked, *OPTIND* shall be initialized to 1.

18734           When the option requires an option-argument, the *getopts* utility shall place it in the shell  
 18735 variable *OPTARG*. If no option was found, or if the option that was found does not have an  
 18736 option-argument, *OPTARG* shall be unset.

18737           If an option character not contained in the *optstring* operand is found where an option character  
 18738 is expected, the shell variable specified by *name* shall be set to the question-mark ('?') character.  
 18739 In this case, if the first character in *optstring* is a colon (':'), the shell variable *OPTARG* shall be  
 18740 set to the option character found, but no output shall be written to standard error; otherwise, the  
 18741 shell variable *OPTARG* shall be unset and a diagnostic message shall be written to standard  
 18742 error. This condition shall be considered to be an error detected in the way arguments were  
 18743 presented to the invoking application, but shall be not an error in *getopts* processing.

18744           If an option-argument is missing:

- 18745           • If the first character of *optstring* is a colon, the shell variable specified by *name* shall be set to  
 18746           the colon character and the shell variable *OPTARG* shall be set to the option character found.
- 18747           • Otherwise, the shell variable specified by *name* shall be set to the question-mark character,  
 18748           the shell variable *OPTARG* shall be unset, and a diagnostic message shall be written to  
 18749           standard error. This condition shall be considered to be an error detected in the way  
 18750           arguments were presented to the invoking application, but shall not be an error in *getopts*  
 18751           processing; a diagnostic message shall be written as stated, but the exit status shall be zero.

18752           When the end of options is encountered, the *getopts* utility shall exit with a return value greater  
 18753 than zero; the shell variable *OPTIND* shall be set to the index of the first non-option-argument,  
 18754 where the first "--" argument is considered to be an option-argument if there are no other  
 18755 non-option-arguments appearing before it, or the value "\$#+1" if there are no non-option-  
 18756 arguments; the *name* variable shall be set to the question-mark character. Any of the following  
 18757 shall identify the end of options: the special option "--", finding an argument that does not  
 18758 begin with a '-', or encountering an error.

18759           The shell variables *OPTIND* and *OPTARG* shall be local to the caller of *getopts* and shall not be  
 18760 exported by default.

18761           The shell variable specified by the *name* operand, *OPTIND* and *OPTARG* shall affect the current  
 18762 shell execution environment; see Section 2.12 (on page 2263).

18763           If the application sets *OPTIND* to the value 1, a new set of parameters can be used: either the  
 18764 current positional parameters or new *arg* values. Any other attempt to invoke *getopts* multiple  
 18765 times in a single shell execution environment with parameters (positional parameters or *arg*  
 18766 operands) that are not the same in all invocations, or with an *OPTIND* value modified to be a  
 18767 value other than 1, produces unspecified results.

18768 **OPTIONS**

18769       None.

18770 **OPERANDS**

18771       The following operands shall be supported:

18772       *optstring*     A string containing the option characters recognized by the utility invoking *getopts*.  
 18773                     If a character is followed by a colon, the option shall be expected to have an  
 18774                     argument, which should be supplied as a separate argument. Applications should  
 18775                     specify an option character and its option-argument as separate arguments, but  
 18776                     *getopts* shall interpret the characters following an option character requiring  
 18777                     arguments as an argument whether or not this is done. An explicit null option-  
 18778                     argument need not be recognized if it is not supplied as a separate argument when  
 18779                     *getopts* is invoked. (See also the *getopt()* function defined in the System Interfaces  
 18780                     volume of IEEE Std 1003.1-200x.) The characters question-mark and colon shall not  
 18781                     be used as option characters by an application. The use of other option characters  
 18782                     that are not alphanumeric produces unspecified results. If the option-argument is  
 18783                     not supplied as a separate argument from the option character, the value in  
 18784                     *OPTARG* shall be stripped of the option character and the '-'. The first character  
 18785                     in *optstring* determines how *getopts* behaves if an option character is not known or  
 18786                     an option-argument is missing.

18787       *name*         The name of a shell variable that shall be set by the *getopts* utility to the option  
 18788                     character that was found.

18789       The *getopts* utility by default shall parse positional parameters passed to the invoking shell  
 18790       procedure. If *args* are given, they shall be parsed instead of the positional parameters.

18791 **STDIN**

18792       Not used.

18793 **INPUT FILES**

18794       None.

18795 **ENVIRONMENT VARIABLES**18796       The following environment variables shall affect the execution of *getopts*:

18797       *LANG*         Provide a default value for the internationalization variables that are unset or null.  
 18798                     (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
 18799                     Internationalization Variables for the precedence of internationalization variables  
 18800                     used to determine the values of locale categories.)

18801       *LC\_ALL*       If set to a non-empty string value, override the values of all the other  
 18802                     internationalization variables.

18803       *LC\_CTYPE*     Determine the locale for the interpretation of sequences of bytes of text data as  
 18804                     characters (for example, single-byte as opposed to multi-byte characters in  
 18805                     arguments and input files).

18806       *LC\_MESSAGES*  
 18807                     Determine the locale that should be used to affect the format and contents of  
 18808                     diagnostic messages written to standard error.

18809 *XSI*       *NLSPATH*     Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

18810       *OPTIND*       This variable shall be used by the *getopts* utility as the index of the next argument  
 18811                     to be processed.

18812 **ASYNCHRONOUS EVENTS**

18813 Default.

18814 **STDOUT**

18815 Not used.

18816 **STDERR**

18817 Whenever an error is detected and the first character in the *optstring* operand is not a colon  
 18818 (' : '), a diagnostic message shall be written to standard error with the following information in  
 18819 an unspecified format:

18820 • The invoking program name shall be identified in the message. The invoking program name  
 18821 shall be the value of the shell special parameter 0 (see Section 2.5.2 (on page 2235)) at the time  
 18822 the *getopts* utility is invoked. A name equivalent to:

18823 basename "\$0"

18824 may be used.

18825 • If an option is found that was not specified in *optstring*, this error is identified and the invalid  
 18826 option character shall be identified in the message.

18827 • If an option requiring an option-argument is found, but an option-argument is not found,  
 18828 this error shall be identified and the invalid option character shall be identified in the  
 18829 message.

18830 **OUTPUT FILES**

18831 None.

18832 **EXTENDED DESCRIPTION**

18833 None.

18834 **EXIT STATUS**

18835 The following exit values shall be returned:

18836 0 An option, specified or unspecified by *optstring*, was found.

18837 &gt;0 The end of options was encountered or an error occurred.

18838 **CONSEQUENCES OF ERRORS**

18839 Default.

18840 **APPLICATION USAGE**

18841 Since *getopts* affects the current shell execution environment, it is generally provided as a shell  
 18842 regular built-in. If it is called in a subshell or separate utility execution environment, such as one  
 18843 of the following:

18844 (*getopts* abc value "\$@")18845 nohup *getopts* ...18846 find . -exec *getopts* ... \;

18847 it does not affect the shell variables in the caller's environment.

18848 Note that shell functions share *OPTIND* with the calling shell even though the positional  
 18849 parameters are changed. If the calling shell and any of its functions uses *getopts* to parse  
 18850 arguments, the results are unspecified.

18851 **EXAMPLES**

18852 The following example script parses and displays its arguments:

18853 aflag=

18854 bflag=

```

18855     while getopts ab: name
18856     do
18857         case $name in
18858             a)     aflag=1;;
18859             b)     bflag=1
18860                   bval="$OPTARG";;
18861             ?)     printf "Usage: %s: [-a] [-b value] args\n" $0
18862                   exit 2;;
18863         esac
18864     done
18865     if [ ! -z "$aflag" ]; then
18866         printf "Option -a specified\n"
18867     fi
18868     if [ ! -z "$bflag" ]; then
18869         printf 'Option -b "%s" specified\n' "$bval"
18870     fi
18871     shift $(( $OPTIND - 1 ))
18872     printf "Remaining arguments are: %s\n" "$*"

```

### 18873 RATIONALE

18874 The *getopts* utility was chosen in preference to the System V *getopt* utility because *getopts* handles  
 18875 option-arguments containing <blank>s.

18876 The *OPTARG* variable is not mentioned in the ENVIRONMENT VARIABLES section because it  
 18877 does not affect the execution of *getopts*; it is one of the few “output-only” variables used by the  
 18878 standard utilities.

18879 The colon is not allowed as an option character because that is not historical behavior, and it  
 18880 violates the Utility Syntax Guidelines. The colon is now specified to behave as in the KornShell  
 18881 version of the *getopts* utility; when used as the first character in the *optstring* operand, it disables  
 18882 diagnostics concerning missing option-arguments and unexpected option characters. This  
 18883 replaces the use of the *OPTERR* variable that was specified in an early proposal.

18884 The formats of the diagnostic messages produced by the *getopts* utility and the *getopt()* function  
 18885 are not fully specified because implementations with superior (“friendlier”) formats objected to  
 18886 the formats used by some historical implementations. The standard developers considered it  
 18887 important that the information in the messages used be uniform between *getopts* and *getopt()*.  
 18888 Exact duplication of the messages might not be possible, particularly if a utility is built on  
 18889 another system that has a different *getopt()* function, but the messages must have specific  
 18890 information included so that the program name, invalid option character, and type of error can  
 18891 be distinguished by a user.

18892 Only a rare application program intercepts a *getopts* standard error message and wants to parse  
 18893 it. Therefore, implementations are free to choose the most usable messages they can devise. The  
 18894 following formats are used by many historical implementations:

18895 "%s: illegal option -- %c\n", <program name>, <option character>

18896 "%s: option requires an argument -- %c\n", <program name>, \  
 18897 <option character>

18898 Historical shells with built-in versions of *getopt()* or *getopts* have used different formats,  
 18899 frequently not even indicating the option character found in error.

18900 **FUTURE DIRECTIONS**

18901 None.

18902 **SEE ALSO**18903 The System Interfaces volume of IEEE Std 1003.1-200x, *getopt()*18904 **CHANGE HISTORY**

18905 First released in Issue 4.

18906 **Issue 6**

18907 The normative text is reworded to avoid use of the term “must” for application requirements.

## 18908 NAME

18909 grep — search a file for a pattern

## 18910 SYNOPSIS

18911 grep [-E| -F][-c| -l| -q][-insvx] -e *pattern\_list*...  
18912 [-f *pattern\_file*]...[*file*...]18913 grep [-E| -F][-c| -l| -q][-insvx][*-e pattern\_list*]...  
18914 -f *pattern\_file*...[*file*...]18915 grep [-E| -F][-c| -l| -q][-insvx] *pattern\_list*[*file*...]

## 18916 DESCRIPTION

18917 The *grep* utility shall search the input files, selecting lines matching one or more patterns; the  
 18918 types of patterns are controlled by the options specified. The patterns are specified by the *-e*  
 18919 option, *-f* option, or the *pattern\_list* operand. The *pattern\_list*'s value shall consist of one or more  
 18920 patterns separated by <newline>s; the *pattern\_file*'s contents shall consist of one or more  
 18921 patterns terminated by <newline>. By default, an input line shall be selected if any pattern,  
 18922 treated as an entire basic regular expression (BRE) as described in the Base Definitions volume of  
 18923 IEEE Std 1003.1-200x, Section 9.3, Basic Regular Expressions, matches any part of the line  
 18924 excluding the terminating <newline>; a null BRE shall match every line. By default, each selected  
 18925 input line shall be written to the standard output.

18926 Regular expression matching shall be based on text lines. Since a <newline> separates or  
 18927 terminates patterns (see the *-e* and *-f* options below), regular expressions cannot contain a  
 18928 <newline>. Similarly, since patterns are matched against individual lines (excluding the  
 18929 terminating <newline>s) of the input, there is no way for a pattern to match a <newline> found  
 18930 in the input.

## 18931 OPTIONS

18932 The *grep* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section  
18933 12.2, Utility Syntax Guidelines.

18934 The following options shall be supported:

18935 **-E** Match using extended regular expressions. Treat each pattern specified as an ERE,  
 18936 as described in the Base Definitions volume of IEEE Std 1003.1-200x, Section 9.4,  
 18937 Extended Regular Expressions. If any entire ERE pattern matches some part of an  
 18938 input line excluding the terminating <newline>, the line shall be matched. A null  
 18939 ERE shall match every line.

18940 **-F** Match using fixed strings. Treat each pattern specified as a string instead of a  
 18941 regular expression. If an input line contains any of the patterns as a contiguous  
 18942 sequence of bytes, the line shall be matched. A null string shall match every line.

18943 **-c** Write only a count of selected lines to standard output.18944 **-e *pattern\_list***

18945 Specify one or more patterns to be used during the search for input. The  
 18946 application shall ensure that patterns in *pattern\_list* are separated by a <newline>.  
 18947 A null pattern can be specified by two adjacent <newline>s in *pattern\_list*. Unless  
 18948 the **-E** or **-F** option is also specified, each pattern shall be treated as a BRE, as  
 18949 described in the Base Definitions volume of IEEE Std 1003.1-200x, Section 9.3, Basic  
 18950 Regular Expressions. Multiple **-e** and **-f** options shall be accepted by the *grep*  
 18951 utility. All of the specified patterns shall be used when matching lines, but the  
 18952 order of evaluation is unspecified.

- 18953        **-f *pattern\_file***  
 18954            Read one or more patterns from the file named by the pathname *pattern\_file*.  
 18955            Patterns in *pattern\_file* shall be terminated by a <newline>. A null pattern can be  
 18956            specified by an empty line in *pattern\_file*. Unless the **-E** or **-F** option is also  
 18957            specified, each pattern shall be treated as a BRE, as described in the Base  
 18958            Definitions volume of IEEE Std 1003.1-200x, Section 9.3, Basic Regular Expressions.
- 18959        **-i**            Perform pattern matching in searches without regard to case; see the Base  
 18960            Definitions volume of IEEE Std 1003.1-200x, Section 9.2, Regular Expression  
 18961            General Requirements.
- 18962        **-l**            (The letter ell.) Write only the names of files containing selected lines to standard  
 18963            output. Pathnames shall be written once per file searched. If the standard input is  
 18964            searched, a pathname of "(standard input)" shall be written, in the POSIX  
 18965            locale. In other locales, "standard input" may be replaced by something more  
 18966            appropriate in those locales.
- 18967        **-n**            Precede each output line by its relative line number in the file, each file starting at  
 18968            line 1. The line number counter shall be reset for each file processed.
- 18969        **-q**            Quiet. Nothing shall be written to the standard output, regardless of matching  
 18970            lines. Exit with zero status if an input line is selected.
- 18971        **-s**            Suppress the error messages ordinarily written for nonexistent or unreadable files.  
 18972            Other error messages shall not be suppressed.
- 18973        **-v**            Select lines not matching any of the specified patterns. If the **-v** option is not  
 18974            specified, selected lines shall be those that match any of the specified patterns.
- 18975        **-x**            Consider only input lines that use all characters in the line excluding the  
 18976            terminating <newline> to match an entire fixed string or regular expression to be  
 18977            matching lines.

**18978 OPERANDS**

18979        The following operands shall be supported:

- 18980        ***pattern\_list***   Specify one or more patterns to be used during the search for input. This operand  
 18981            shall be treated as if it were specified as **-e *pattern\_list***.
- 18982        ***file***            A pathname of a file to be searched for the patterns. If no *file* operands are  
 18983            specified, the standard input shall be used.

**18984 STDIN**

18985        The standard input shall be used only if no *file* operands are specified. See the INPUT FILES  
 18986        section.

**18987 INPUT FILES**

18988        The input files shall be text files.

**18989 ENVIRONMENT VARIABLES**

18990        The following environment variables shall affect the execution of *grep*:

- 18991        ***LANG***            Provide a default value for the internationalization variables that are unset or null.  
 18992            (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
 18993            Internationalization Variables for the precedence of internationalization variables  
 18994            used to determine the values of locale categories.)
- 18995        ***LC\_ALL***          If set to a non-empty string value, override the values of all the other  
 18996            internationalization variables.

- 18997 *LC\_COLLATE*  
 18998 Determine the locale for the behavior of ranges, equivalence classes and multi-  
 18999 character collating elements within regular expressions.
- 19000 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 19001 characters (for example, single-byte as opposed to multi-byte characters in  
 19002 arguments and input files) and the behavior of character classes within regular  
 19003 expressions.
- 19004 *LC\_MESSAGES*  
 19005 Determine the locale that should be used to affect the format and contents of  
 19006 diagnostic messages written to standard error.
- 19007 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 19008 **ASYNCHRONOUS EVENTS**  
 19009 Default.
- 19010 **STDOUT**  
 19011 If the **-l** option is in effect, and the **-q** option is not, the following shall be written for each file  
 19012 containing at least one selected input line:  
 19013 `"%s\n", <file>`
- 19014 Otherwise, if more than one *file* argument appears, and **-q** is not specified, the *grep* utility shall  
 19015 prefix each output line by:  
 19016 `"%s: ", <file>`
- 19017 The remainder of each output line shall depend on the other options specified:
- 19018 • If the **-c** option is in effect, the remainder of each output line shall contain:  
 19019 `"%d\n", <count>`
  - 19020 • Otherwise, if **-c** is not in effect and the **-n** option is in effect, the following shall be written to  
 19021 standard output:  
 19022 `"%d: ", <line number>`
  - 19023 • Finally, the following shall be written to standard output:  
 19024 `"%s", <selected-line contents>`
- 19025 **STDERR**  
 19026 The standard error shall be used only for diagnostic messages. |
- 19027 **OUTPUT FILES**  
 19028 None.
- 19029 **EXTENDED DESCRIPTION**  
 19030 None.
- 19031 **EXIT STATUS**  
 19032 The following exit values shall be returned:
- 19033 0 One or more lines were selected.
  - 19034 1 No lines were selected.
  - 19035 >1 An error occurred.

## 19036 CONSEQUENCES OF ERRORS

19037 If the `-q` option is specified, the exit status shall be zero if an input line is selected, even if an  
 19038 error was detected. Otherwise, default actions shall be performed.

## 19039 APPLICATION USAGE

19040 Care should be taken when using characters in *pattern\_list* that may also be meaningful to the  
 19041 command interpreter. It is safest to enclose the entire *pattern\_list* argument in single quotes:

19042 '...'

19043 The `-e pattern_list` option has the same effect as the *pattern\_list* operand, but is useful when  
 19044 *pattern\_list* begins with the hyphen delimiter. It is also useful when it is more convenient to  
 19045 provide multiple patterns as separate arguments.

19046 Multiple `-e` and `-f` options are accepted and *grep* uses all of the patterns it is given while  
 19047 matching input text lines. (Note that the order of evaluation is not specified. If an  
 19048 implementation finds a null string as a pattern, it is allowed to use that pattern first, matching  
 19049 every line, and effectively ignore any other patterns.)

19050 The `-q` option provides a means of easily determining whether or not a pattern (or string) exists  
 19051 in a group of files. When searching several files, it provides a performance improvement  
 19052 (because it can quit as soon as it finds the first match) and requires less care by the user in  
 19053 choosing the set of files to supply as arguments (because it exits zero if it finds a match even if  
 19054 *grep* detected an access or read error on earlier *file* operands).

## 19055 EXAMPLES

19056 1. To find all uses of the word "Posix" (in any case) in file **text.mm** and write with line  
 19057 numbers:

19058 `grep -i -n posix text.mm`

19059 2. To find all empty lines in the standard input:

19060 `grep ^$`

19061 or:

19062 `grep -v .`

19063 3. Both of the following commands print all lines containing strings "abc" or "def" or both:

19064 `grep -E 'abc|def'` |

19065 `grep -F 'abc|def'` |

19066 4. Both of the following commands print all lines matching exactly "abc" or "def": |

19067 `grep -E '^abc$|^def$'` |

19068 `grep -F -x 'abc|def'` |

## 19069 RATIONALE

19070 This *grep* has been enhanced in an upward-compatible way to provide the exact functionality of  
 19071 the historical *egrep* and *fgrep* commands as well. It was the clear intention of the standard  
 19072 developers to consolidate the three *greps* into a single command.

19073 The old *egrep* and *fgrep* commands are likely to be supported for many years to come as  
 19074 implementation extensions, allowing historical applications to operate unmodified.

19075 Historical implementations usually silently ignored all but one of multiply-specified `-e` and `-f`  
 19076 options, but were not consistent as to which specification was actually used.

- 19077 The **-b** option was omitted from the OPTIONS section because block numbers are  
19078 implementation-defined.
- 19079 The System V restriction on using **-** to mean standard input was omitted.
- 19080 A definition of action taken when given a null BRE or ERE is specified. This is an error condition  
19081 in some historical implementations.
- 19082 The **-I** option previously indicated that its use was undefined when no files were explicitly  
19083 named. This behavior was historical and placed an unnecessary restriction on future  
19084 implementations. It has been removed.
- 19085 The historical BSD *grep* **-s** option practice is easily duplicated by redirecting standard output to  
19086 **/dev/null**. The **-s** option required here is from System V.
- 19087 The **-x** option, historically available only with *fgrep*, is available here for all of the non-  
19088 obsolescent versions.
- 19089 **FUTURE DIRECTIONS**
- 19090 None.
- 19091 **SEE ALSO**
- 19092 *sed*
- 19093 **CHANGE HISTORY**
- 19094 First released in Issue 2.
- 19095 **Issue 6**
- 19096 The Open Group Corrigendum U029/5 is applied, correcting the SYNOPSIS.
- 19097 The normative text is reworded to avoid use of the term “must” for application requirements.

19098 **NAME**

19099 hash — remember or report utility locations

19100 **SYNOPSIS**19101 xSI hash [*utility...*]

19102 hash -r

19103

19104 **DESCRIPTION**

19105 The *hash* utility shall affect the way the current shell environment remembers the locations of  
 19106 utilities found as described in Section 2.9.1.1 (on page 2249). Depending on the arguments  
 19107 specified, it shall add utility locations to its list of remembered locations or it shall purge the  
 19108 contents of the list. When no arguments are specified, it shall report on the contents of the list.

19109 Utilities provided as built-ins to the shell shall not be reported by *hash*.19110 **OPTIONS**

19111 The *hash* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section  
 19112 12.2, Utility Syntax Guidelines.

19113 The following option shall be supported:

19114 -r Forget all previously remembered utility locations.

19115 **OPERANDS**

19116 The following operand shall be supported:

19117 *utility* The name of a utility to be searched for and added to the list of remembered  
 19118 locations. If *utility* contains one or more slashes, the results are unspecified.

19119 **STDIN**

19120 Not used.

19121 **INPUT FILES**

19122 None.

19123 **ENVIRONMENT VARIABLES**19124 The following environment variables shall affect the execution of *hash*:

19125 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 19126 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
 19127 Internationalization Variables for the precedence of internationalization variables  
 19128 used to determine the values of locale categories.)

19129 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 19130 internationalization variables.

19131 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 19132 characters (for example, single-byte as opposed to multi-byte characters in  
 19133 arguments).

19134 *LC\_MESSAGES*

19135 Determine the locale that should be used to affect the format and contents of  
 19136 diagnostic messages written to standard error.

19137 *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

19138 *PATH* Determine the location of *utility*, as described in the Base Definitions volume of  
 19139 IEEE Std 1003.1-200x, Chapter 8, Environment Variables.

19140 **ASYNCHRONOUS EVENTS**

19141 Default.

19142 **STDOUT**

19143 The standard output of *hash* shall be used when no arguments are specified. Its format is  
19144 unspecified, but includes the pathname of each utility in the list of remembered locations for the  
19145 current shell environment. This list shall consist of those utilities named in previous *hash*  
19146 invocations that have been invoked, and may contain those invoked and found through the  
19147 normal command search process.

19148 **STDERR**

19149 The standard error shall be used only for diagnostic messages.

19150 **OUTPUT FILES**

19151 None.

19152 **EXTENDED DESCRIPTION**

19153 None.

19154 **EXIT STATUS**

19155 The following exit values shall be returned:

19156 0 Successful completion.

19157 &gt;0 An error occurred.

19158 **CONSEQUENCES OF ERRORS**

19159 Default.

19160 **APPLICATION USAGE**

19161 Since *hash* affects the current shell execution environment, it is always provided as a shell  
19162 regular built-in. If it is called in a separate utility execution environment, such as one of the  
19163 following:

19164 `nohup hash -r`19165 `find . -type f | xargs hash`

19166 it does not affect the command search process of the caller's environment.

19167 The *hash* utility may be implemented as an alias—for example, *alias -t -*, in which case utilities  
19168 found through normal command search are not listed by the *hash* command.

19169 The effects of *hash -r* can also be achieved portably by resetting the value of *PATH*; in the  
19170 simplest form, this can be:

19171 `PATH="$PATH"`

19172 The use of *hash* with *utility* names is unnecessary for most applications, but may provide a  
19173 performance improvement on a few implementations; normally, the hashing process is included  
19174 by default.

19175 **EXAMPLES**

19176 None.

19177 **RATIONALE**

19178 None.

19179 **FUTURE DIRECTIONS**

19180 None.

19181 **SEE ALSO**

19182           Section 2.9.1.1 (on page 2249)

19183 **CHANGE HISTORY**

19184           First released in Issue 2.

19185 **NAME**

19186 head — copy the first part of files

19187 **SYNOPSIS**

19188 head [-n *number*][*file...*]

19189 **DESCRIPTION**

19190 The *head* utility shall copy its input files to the standard output, ending the output for each file at  
19191 a designated point.

19192 Copying shall end at the point in each input file indicated by the **-n *number*** option. The option-  
19193 argument *number* shall be counted in units of lines.

19194 **OPTIONS**

19195 The *head* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section  
19196 12.2, Utility Syntax Guidelines.

19197 The following option shall be supported:

19198 **-n *number*** The first *number* lines of each input file shall be copied to standard output. The  
19199 application shall ensure that the *number* option-argument is a positive decimal  
19200 integer.

19201 If no options are specified, *head* shall act as if **-n 10** had been specified.

19202 **OPERANDS**

19203 The following operand shall be supported:

19204 *file* A pathname of an input file. If no *file* operands are specified, the standard input  
19205 shall be used.

19206 **STDIN**

19207 The standard input shall be used only if no *file* operands are specified. See the INPUT FILES  
19208 section.

19209 **INPUT FILES**

19210 Input files shall be text files, but the line length is not restricted to {LINE\_MAX} bytes.

19211 **ENVIRONMENT VARIABLES**

19212 The following environment variables shall affect the execution of *head*:

19213 **LANG** Provide a default value for the internationalization variables that are unset or null.  
19214 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
19215 Internationalization Variables for the precedence of internationalization variables  
19216 used to determine the values of locale categories.)

19217 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
19218 internationalization variables.

19219 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
19220 characters (for example, single-byte as opposed to multi-byte characters in  
19221 arguments and input files).

19222 **LC\_MESSAGES**

19223 Determine the locale that should be used to affect the format and contents of  
19224 diagnostic messages written to standard error.

19225 **XSI** **NLS\_PATH** Determine the location of message catalogs for the processing of **LC\_MESSAGES**.

19226 **ASYNCHRONOUS EVENTS**

19227 Default.

19228 **STDOUT**

19229 The standard output shall contain designated portions of the input files.

19230 If multiple *file* operands are specified, *head* shall precede the output for each with the header:

19231 "\n==&gt; %s &lt;==\n", &lt;pathname&gt;

19232 except that the first header written shall not include the initial &lt;newline&gt;.

19233 **STDERR**

19234 The standard error shall be used only for diagnostic messages. |

19235 **OUTPUT FILES**

19236 None.

19237 **EXTENDED DESCRIPTION**

19238 None.

19239 **EXIT STATUS**

19240 The following exit values shall be returned:

19241 0 Successful completion.

19242 &gt;0 An error occurred.

19243 **CONSEQUENCES OF ERRORS**

19244 Default.

19245 **APPLICATION USAGE**19246 The obsolescent *-number* form is withdrawn in this version. Applications should use the *-n*  
19247 *number* option.19248 **EXAMPLES**

19249 To write the first ten lines of all files (except those with a leading period) in the directory:

19250 head \*

19251 **RATIONALE**19252 Although it is possible to simulate *head* with *sed* 10q for a single file, the standard developers  
19253 decided that the popularity of *head* on historical BSD systems warranted its inclusion alongside  
19254 *tail*.19255 This standard version of *head* follows the Utility Syntax Guidelines. The *-n* option was added to  
19256 this new interface so that *head* and *tail* would be more logically related.19257 There is no *-c* option (as there is in *tail*) because it is not historical practice and because other  
19258 utilities in this volume of IEEE Std 1003.1-200x provide similar functionality.19259 **FUTURE DIRECTIONS**

19260 None.

19261 **SEE ALSO**19262 *sed*, *tail*19263 **CHANGE HISTORY**

19264 First released in Issue 4.

19265 **Issue 6**

19266 The obsolescent –**number** form is withdrawn.

19267 The normative text is reworded to avoid use of the term “must” for application requirements.

19268 **NAME**

19269 iconv — codeset conversion

19270 **SYNOPSIS**19271 iconv [-cs] -f *fromcode* -t *toctype* [*file* ...]

19272 iconv -l

19273 **DESCRIPTION**19274 The *iconv* utility shall convert the encoding of characters in *file* from one codeset to another and  
19275 write the results to standard output.19276 When the options indicate that charmap files are used to specify the codesets (see **OPTIONS**),  
19277 the codeset conversion shall be accomplished by performing a logical join on the symbolic  
19278 character names in the two charmaps. The implementation need not support the use of charmap  
19279 files for codeset conversion unless the `POSIX2_LOCALEDEF` symbol is defined on the system.19280 **OPTIONS**19281 The *iconv* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section  
19282 12.2, Utility Syntax Guidelines.

19283 The following options shall be supported:

19284 **-c** Omit any invalid characters from the output. When **-c** is not used, the results of  
19285 encountering invalid characters in the input stream (either those that are not valid  
19286 members of the *fromcode* or those that have no corresponding value in *toctype*) shall  
19287 be specified in the system documentation. The presence or absence of **-c** shall not  
19288 affect the exit status of *iconv*.19289 **-f *fromcode*** Identify the codeset of the input file. If the option-argument contains a slash  
19290 character, *iconv* shall attempt to use it as the pathname of a charmap file, as  
19291 defined in the Base Definitions volume of IEEE Std 1003.1-200x, Section 6.4,  
19292 Character Set Description File. If the pathname does not represent a valid, readable  
19293 charmap file, the results are undefined. If the option-argument does not contain a  
19294 slash, it shall be considered the name of one of the codeset descriptions provided  
19295 by the system, in an unspecified format. The valid values of the option-argument  
19296 without a slash are implementation-defined. If this option is omitted, the codeset  
19297 of the current locale shall be used.19298 **-l** Write all supported *fromcode* and *toctype* values to standard output in an unspecified  
19299 format.19300 **-s** Suppress any messages written to standard error concerning invalid characters.  
19301 When **-s** is not used, the results of encountering invalid characters in the input  
19302 stream (either those that are not valid members of the *fromcode* or those that have  
19303 no corresponding value in *toctype*) shall be specified in the system documentation.  
19304 The presence or absence of **-s** shall not affect the exit status of *iconv*.19305 **-t *toctype*** Identify the codeset to be used for the output file. The semantics shall be |  
19306 equivalent to the **-f *fromcode*** option. |19307 If either **-f** or **-t** represents a charmap file, but the other does not (or is omitted), or both **-f** and  
19308 **-t** are omitted, the results are undefined.19309 **OPERANDS**

19310 The following operand shall be supported:

19311 ***file*** A pathname of an input file. If no *file* operands are specified, or if a *file* operand is  
19312 '-', the standard input shall be used.

19313 **STDIN**

19314 The standard input shall be used only if no *file* operands are specified, or if a *file* operand is '- '.

19315 **INPUT FILES**

19316 The input file shall be a text file.

19317 **ENVIRONMENT VARIABLES**

19318 The following environment variables shall affect the execution of *iconv*:

19319 *LANG* Provide a default value for the internationalization variables that are unset or null.  
19320 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
19321 Internationalization Variables for the precedence of internationalization variables  
19322 used to determine the values of locale categories.)

19323 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
19324 internationalization variables.

19325 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
19326 characters (for example, single-byte as opposed to multi-byte characters in  
19327 arguments). During translation of the file, this variable is superseded by the use of  
19328 the *fromcode* option-argument.

19329 *LC\_MESSAGES*

19330 Determine the locale that should be used to affect the format and contents of  
19331 diagnostic messages written to standard error.

19332 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

19333 **ASYNCHRONOUS EVENTS**

19334 Default.

19335 **STDOUT**

19336 When the *-l* option is used, the standard output shall contain all supported *fromcode* and *to*  
19337 *code* values, written in an unspecified format.

19338 When the *-l* option is not used, the standard output shall contain the sequence of characters  
19339 read from the input files, translated to the specified codeset. Nothing else shall be written to the  
19340 standard output.

19341 **STDERR**

19342 The standard error shall be used only for diagnostic messages. |

19343 **OUTPUT FILES**

19344 None.

19345 **EXTENDED DESCRIPTION**

19346 None.

19347 **EXIT STATUS**

19348 The following exit values shall be returned:

19349 0 Successful completion.

19350 >0 An error occurred.

19351 **CONSEQUENCES OF ERRORS**

19352 Default.

19353 **APPLICATION USAGE**

19354           The user must ensure that both charmap files use the same symbolic names for characters the  
19355           two codesets have in common.

19356 **EXAMPLES**

19357           The following example converts the contents of file **mail.x400** from the ISO/IEC 6937:1994  
19358           standard codeset to the ISO/IEC 8859-1:1998 standard codeset, and stores the results in file  
19359           **mail.local**:

```
19360           iconv -f IS6937 -t IS8859 mail.x400 > mail.local
```

19361 **RATIONALE**

19362           The *iconv* utility can be used portably only when the user provides two charmap files as option-  
19363           arguments. This is because a single charmap provided by the user cannot reliably be joined with  
19364           the names in a system-provided character set description. The valid values for *fromcode* and  
19365           *tocode* are implementation-defined and do not have to have any relation to the charmap  
19366           mechanisms. As an aid to interactive users, the **-I** option was adopted from the Plan 9 operating  
19367           system. It writes information concerning these implementation-defined values. The format is  
19368           unspecified because there are many possible useful formats that could be chosen, such as a  
19369           matrix of valid combinations of *fromcode* and *tocode*. The **-I** option is not intended for shell script  
19370           usage; conforming applications will have to use charmaps.

19371 **FUTURE DIRECTIONS**

19372           None.

19373 **SEE ALSO**

19374           *gencat*

19375 **CHANGE HISTORY**

19376           First released in Issue 3.

19377 **Issue 6**

19378           This utility has been rewritten to align with the IEEE P1003.2b draft standard. Specifically, the  
19379           ability to use charmap files for conversion has been added.

19380 **NAME**

19381           id — return user identity

19382 **SYNOPSIS**

19383           id [*user*]

19384           id -G[-n] [*user*]

19385           id -g[-nr] [*user*]

19386           id -u[-nr] [*user*]

19387 **DESCRIPTION**

19388           If no *user* operand is provided, the *id* utility shall write the user and group IDs and the  
19389           corresponding user and group names of the invoking process to standard output. If the effective  
19390           and real IDs do not match, both shall be written. If multiple groups are supported by the  
19391           underlying system (see the description of {NGROUPS\_MAX} in the System Interfaces volume of  
19392           IEEE Std 1003.1-200x), the supplementary group affiliations of the invoking process shall also be  
19393           written.

19394           If a *user* operand is provided and the process has the appropriate privileges, the user and group  
19395           IDs of the selected user shall be written. In this case, effective IDs shall be assumed to be  
19396           identical to real IDs. If the selected user has more than one allowable group membership listed  
19397           in the group database, these shall be written in the same manner as the supplementary groups  
19398           described in the preceding paragraph.

19399 **OPTIONS**

19400           The *id* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2,  
19401           Utility Syntax Guidelines.

19402           The following options shall be supported:

19403           **-G**           Output all different group IDs (effective, real, and supplementary) only, using the  
19404           format "%u\n". If there is more than one distinct group affiliation, output each  
19405           such affiliation, using the format " %u", before the <newline> is output.

19406           **-g**           Output only the effective group ID, using the format "%u\n".

19407           **-n**           Output the name in the format "%s" instead of the numeric ID using the format  
19408           "%u".

19409           **-r**           Output the real ID instead of the effective ID.

19410           **-u**           Output only the effective user ID, using the format "%u\n".

19411 **OPERANDS**

19412           The following operand shall be supported:

19413           *user*           The login name for which information is to be written.

19414 **STDIN**

19415           Not used.

19416 **INPUT FILES**

19417           None.

19418 **ENVIRONMENT VARIABLES**

19419           The following environment variables shall affect the execution of *id*:

19420           **LANG**           Provide a default value for the internationalization variables that are unset or null.  
19421           (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
19422           Internationalization Variables for the precedence of internationalization variables

19423 used to determine the values of locale categories.)

19424 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
19425 internationalization variables.

19426 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
19427 characters (for example, single-byte as opposed to multi-byte characters in  
19428 arguments).

19429 **LC\_MESSAGES**  
19430 Determine the locale that should be used to affect the format and contents of  
19431 diagnostic messages written to standard error and informative messages written to  
19432 standard output.

19433 XSI **NLSPATH** Determine the location of message catalogs for the processing of **LC\_MESSAGES**.

19434 **ASYNCHRONOUS EVENTS**  
19435 Default.

19436 **STDOUT**  
19437 The following formats shall be used when the **LC\_MESSAGES** locale category specifies the  
19438 POSIX locale. In other locales, the strings *uid*, *gid*, *eid*, *egid*, and *groups* may be replaced with  
19439 more appropriate strings corresponding to the locale.

19440 "uid=%u(%s) gid=%u(%s)\n", <real user ID>, <user-name>,  
19441 <real group ID>, <group-name>

19442 If the effective and real user IDs do not match, the following shall be inserted immediately  
19443 before the '\n' character in the previous format:

19444 " eid=%u(%s) "

19445 with the following arguments added at the end of the argument list:  
19446 <effective user ID>, <effective user-name>

19447 If the effective and real group IDs do not match, the following shall be inserted directly before  
19448 the '\n' character in the format string (and after any addition resulting from the effective and  
19449 real user IDs not matching):

19450 " egid=%u(%s) "

19451 with the following arguments added at the end of the argument list:  
19452 <effective group-ID>, <effective group name>

19453 If the process has supplementary group affiliations or the selected user is allowed to belong to  
19454 multiple groups, the first shall be added directly before the <newline> in the format string:

19455 " groups=%u(%s) "

19456 with the following arguments added at the end of the argument list:  
19457 <supplementary group ID>, <supplementary group name>

19458 and the necessary number of the following added after that for any remaining supplementary  
19459 group IDs:

19460 " ,%u(%s) "

19461 and the necessary number of the following arguments added at the end of the argument list:  
19462 <supplementary group ID>, <supplementary group name>

19463 If any of the user ID, group ID, effective user ID, effective group ID, or supplementary/multiple  
19464 group IDs cannot be mapped by the system into printable user or group names, the  
19465 corresponding ("%s") and *name* argument shall be omitted from the corresponding format  
19466 string.

19467 When any of the options are specified, the output format shall be as described in the OPTIONS  
19468 section.

#### 19469 **STDERR**

19470 The standard error shall be used only for diagnostic messages.

#### 19471 **OUTPUT FILES**

19472 None.

#### 19473 **EXTENDED DESCRIPTION**

19474 None.

#### 19475 **EXIT STATUS**

19476 The following exit values shall be returned:

19477 0 Successful completion.

19478 >0 An error occurred.

#### 19479 **CONSEQUENCES OF ERRORS**

19480 Default.

#### 19481 **APPLICATION USAGE**

19482 Output produced by the **-G** option and by the default case could potentially produce very long  
19483 lines on systems that support large numbers of supplementary groups. (On systems with user  
19484 and group IDs that are 32-bit integers and with group names with a maximum of 8 bytes per  
19485 name, 93 supplementary groups plus distinct effective and real group and user IDs could  
19486 theoretically overflow the 2 048-byte {LINE\_MAX} text file line limit on the default output case.  
19487 It would take about 186 supplementary groups to overflow the 2 048-byte barrier using *id -G*).  
19488 This is not expected to be a problem in practice, but in cases where it is a concern, applications  
19489 should consider using *fold -s* before postprocessing the output of *id*.

#### 19490 **EXAMPLES**

19491 None.

#### 19492 **RATIONALE**

19493 The functionality provided by the 4 BSD *groups* utility can be simulated using:

19494 `id -Gn [ user ]`

19495 The 4 BSD command *groups* was considered, but it was not included because it did not provide  
19496 the functionality of the *id* utility of the SVID. Also, it was thought that it would be easier to  
19497 modify *id* to provide the additional functionality necessary to systems with multiple groups  
19498 than to invent another command.

19499 The options **-u**, **-g**, **-n**, and **-r** were added to ease the use of *id* with shell commands  
19500 substitution. Without these options it is necessary to use some preprocessor such as *sed* to select  
19501 the desired piece of information. Since output such as that produced by:

19502 `id -u -n`

19503 is frequently wanted, it seemed desirable to add the options.

19504 **FUTURE DIRECTIONS**

19505           None.

19506 **SEE ALSO**19507           *fold*, *logname*, *who*, the System Interfaces volume of IEEE Std 1003.1-200x, *getgid()*, *getgroups()*,19508           *getuid()*19509 **CHANGE HISTORY**

19510           First released in Issue 2.

19511 **NAME**

19512 ipcrm — remove an XSI message queue, semaphore set, or shared memory segment identifier

19513 **SYNOPSIS**

```
19514 xsi ipcrm [ -q msgid | -Q msgkey | -s semid | -S semkey |
19515 -m shmid | -M shmkey ] ...
```

19516

19517 **DESCRIPTION**

19518 The *ipcrm* utility shall remove zero or more message queues, semaphore sets, or shared memory  
 19519 segments. The interprocess communication facilities to be removed are specified by the options.

19520 Only a user with appropriate privilege shall be allowed to remove an interprocess  
 19521 communication facility that was not created by or owned by the user invoking *ipcrm*.

19522 **OPTIONS**

19523 The *ipcrm* facility supports the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2,  
 19524 Utility Syntax Guidelines.

19525 The following options shall be supported:

19526 **-q msgid** Remove the message queue identifier *msgid* from the system and destroy the  
 19527 message queue and data structure associated with it.

19528 **-m shmid** Remove the shared memory identifier *shmid* from the system. The shared memory  
 19529 segment and data structure associated with it shall be destroyed after the last  
 19530 detach.

19531 **-s semid** Remove the semaphore identifier *semid* from the system and destroy the set of  
 19532 semaphores and data structure associated with it.

19533 **-Q msgkey** Remove the message queue identifier, created with key *msgkey*, from the system  
 19534 and destroy the message queue and data structure associated with it.

19535 **-M shmkey** Remove the shared memory identifier, created with key *shmkey*, from the system.  
 19536 The shared memory segment and data structure associated with it shall be  
 19537 destroyed after the last detach.

19538 **-S semkey** Remove the semaphore identifier, created with key *semkey*, from the system and  
 19539 destroy the set of semaphores and data structure associated with it.

19540 **OPERANDS**

19541 None.

19542 **STDIN**

19543 Not used.

19544 **INPUT FILES**

19545 None.

19546 **ENVIRONMENT VARIABLES**

19547 The following environment variables shall affect the execution of *ipcrm*:

19548 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 19549 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
 19550 Internationalization Variables for the precedence of internationalization variables  
 19551 used to determine the values of locale categories.)

19552 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 19553 internationalization variables.

- 19554 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
19555 characters (for example, single-byte as opposed to multi-byte characters in  
19556 arguments).
- 19557 **LC\_MESSAGES**  
19558 Determine the locale that should be used to affect the format and contents of  
19559 diagnostic messages written to standard error.
- 19560 **NLSPATH**  
19561 Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 19562 **ASYNCHRONOUS EVENTS**  
19563 Default.
- 19564 **STDOUT**  
19565 Not used.
- 19566 **STDERR**  
19567 The standard error shall be used only for diagnostic messages. |
- 19568 **OUTPUT FILES**  
19569 None.
- 19570 **EXTENDED DESCRIPTION**  
19571 None.
- 19572 **EXIT STATUS**  
19573 The following exit values shall be returned:  
19574 0 Successful completion.  
19575 >0 An error occurred.
- 19576 **CONSEQUENCES OF ERRORS**  
19577 Default.
- 19578 **APPLICATION USAGE**  
19579 None.
- 19580 **EXAMPLES**  
19581 None.
- 19582 **RATIONALE**  
19583 None.
- 19584 **FUTURE DIRECTIONS**  
19585 None.
- 19586 **SEE ALSO**  
19587 *ipcs*, the System Interfaces volume of IEEE Std 1003.1-200x, *msgctl()*, *semctl()*, *shmctl()*
- 19588 **CHANGE HISTORY**  
19589 First released in Issue 5.

## 19590 NAME

19591 ipcs — report XSI interprocess communication facilities status

## 19592 SYNOPSIS

19593 xsi `ipcs [-qms] [-a | -bcopt]`

19594

## 19595 DESCRIPTION

19596 The *ipcs* utility shall write information about active interprocess communication facilities.19597 Without options, information shall be written in short format for message queues, shared  
19598 memory segments, and semaphores sets that are currently active in the system. Otherwise, the  
19599 information that is displayed is controlled by the options specified.

## 19600 OPTIONS

19601 The *ipcs* facility supports the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2,  
19602 Utility Syntax Guidelines.19603 The *ipcs* utility accepts the following options:19604 **-q** Write information about active message queues.19605 **-m** Write information about active shared memory segments.19606 **-s** Write information about active semaphores sets.19607 If **-q**, **-m**, or **-s** are specified, only information about those facilities shall be written. If none of  
19608 these three are specified, information about all three shall be written subject to the following  
19609 options:19610 **-a** Use all print options. (This is a shorthand notation for **-b**, **-c**, **-o**, **-p**, and **-t**.)19611 **-b** Write information on maximum allowable size. (Maximum number of bytes in  
19612 messages on queue for message queues, size of segments for shared memory, and  
19613 number of semaphores in each set for semaphores.)19614 **-c** Write creator's user name and group name; see below.19615 **-o** Write information on outstanding usage. (Number of messages on queue and total  
19616 number of bytes in messages on queue for message queues, and number of  
19617 processes attached to shared memory segments.)19618 **-p** Write process number information. (Process ID of last process to send a message  
19619 and process ID of last process to receive a message on message queues, process ID  
19620 of creating process, and process ID of last process to attach or detach on shared  
19621 memory segments.)19622 **-t** Write time information. (Time of the last control operation that changed the access  
19623 permissions for all facilities, time of last *msgsnd()* and *msgrcv()* operations on  
19624 message queues, time of last *shmat()* and *shmdt()* operations on shared memory,  
19625 and time of last *semop()* operation on semaphores.)

## 19626 OPERANDS

19627 None.

## 19628 STDIN

19629 Not used.

19630 **INPUT FILES**

- 19631           • The group database
- 19632           • The user database

19633 **ENVIRONMENT VARIABLES**

19634           The following environment variables shall affect the execution of *ipcs*:

- 19635           *LANG*       Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)
- 19636
- 19637
- 19638
- 19639           *LC\_ALL*      If set to a non-empty string value, override the values of all the other internationalization variables.
- 19640
- 19641           *LC\_CTYPE*   Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).
- 19642
- 19643
- 19644           *LC\_MESSAGES*   Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.
- 19645
- 19646
- 19647           *NLSPATH*     Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 19648           *TZ*           Determine the timezone for the date and time strings written by *ipcs*. If *TZ* is unset or null, an unspecified default timezone shall be used.
- 19649

19650 **ASYNCHRONOUS EVENTS**

19651           Default.

19652 **STDOUT**

19653           An introductory line shall be written with the format:

19654           "IPC status from %s as of %s\n", <source>, <date>

19655           where <source> indicates the source used to gather the statistics and <date> is the information that would be produced by the *date* command when invoked in the POSIX locale.

19656

19657           The *ipcs* utility then shall create up to three reports depending upon the *-q*, *-m*, and *-s* options. The first report shall indicate the status of message queues, the second report shall indicate the status of shared memory segments, and the third report shall indicate the status of semaphore sets.

19661           If the corresponding facility is not installed or has not been used since the last reboot, then the report shall be written out in the format:

19662

19663           "%s facility not in system.\n", <facility>

19664           where <facility> is *Message Queue*, *Shared Memory*, or *Semaphore*, as appropriate. If the facility has been installed and has been used since the last reboot, column headings separated by one or more spaces and followed by a <newline> shall be written as indicated below followed by the facility name written out using the format:

19665

19666

19667

19668           "%s:\n", <facility>

19669           where <facility> is *Message Queues*, *Shared Memory*, or *Semaphores*, as appropriate. On the second and third reports the column headings need not be written if the last column headings written already provide column headings for all information in that report.

19670

19671

|       |      |              |                                                                                                     |
|-------|------|--------------|-----------------------------------------------------------------------------------------------------|
| 19672 |      |              | The column headings provided in the first column below and the meaning of the information in        |
| 19673 |      |              | those columns shall be given in order below; the letters in parentheses indicate the options that   |
| 19674 |      |              | shall cause the corresponding column to appear; “all” means that the column shall always            |
| 19675 |      |              | appear. Each column is separated by one or more <space>s. Note that these options only              |
| 19676 |      |              | determine what information is provided for each report; they do not determine which reports         |
| 19677 |      |              | are written.                                                                                        |
| 19678 | T    | (all)        | Type of facility:                                                                                   |
| 19679 |      | q            | Message queue.                                                                                      |
| 19680 |      | m            | Shared memory segment.                                                                              |
| 19681 |      | s            | Semaphore.                                                                                          |
| 19682 |      |              | This field is a single character written using the format %c.                                       |
| 19683 | ID   | (all)        | The identifier for the facility entry. This field shall be written using the format                 |
| 19684 |      |              | %d.                                                                                                 |
| 19685 | KEY  | (all)        | The key used as an argument to <i>msgget()</i> , <i>semget()</i> , or <i>shmget()</i> to create the |
| 19686 |      |              | facility entry.                                                                                     |
| 19687 |      | <b>Note:</b> | The key of a shared memory segment is changed to IPC_PRIVATE when                                   |
| 19688 |      |              | the segment has been removed until all processes attached to the segment                            |
| 19689 |      |              | detach it.                                                                                          |
| 19690 |      |              | This field shall be written using the format 0x%x.                                                  |
| 19691 | MODE | (all)        | The facility access modes and flags. The mode shall consist of 11 characters                        |
| 19692 |      |              | that are interpreted as follows.                                                                    |
| 19693 |      |              | The first character shall be:                                                                       |
| 19694 |      | S            | If a process is waiting on a <i>msgsnd()</i> operation.                                             |
| 19695 |      | –            | If the above is not true.                                                                           |
| 19696 |      |              | The second character shall be:                                                                      |
| 19697 |      | R            | If a process is waiting on a <i>msgrcv()</i> operation.                                             |
| 19698 |      | C or –       | If the associated shared memory segment is to be cleared when the                                   |
| 19699 |      |              | first attach operation is executed.                                                                 |
| 19700 |      | –            | If none of the above is true.                                                                       |
| 19701 |      |              | The next nine characters shall be interpreted as three sets of three bits each.                     |
| 19702 |      |              | The first set refers to the owner’s permissions; the next to permissions of                         |
| 19703 |      |              | others in the usergroup of the facility entry; and the last to all others. Within                   |
| 19704 |      |              | each set, the first character indicates permission to read, the second character                    |
| 19705 |      |              | indicates permission to write or alter the facility entry, and the last character is                |
| 19706 |      |              | a minus sign ( ‘ – ‘ ).                                                                             |
| 19707 |      |              | The permissions shall be indicated as follows:                                                      |
| 19708 |      | r            | If read permission is granted.                                                                      |
| 19709 |      | w            | If write permission is granted.                                                                     |
| 19710 |      | a            | If alter permission is granted.                                                                     |
| 19711 |      | –            | If the indicated permission is not granted.                                                         |

|       |         |       |                                                                                   |
|-------|---------|-------|-----------------------------------------------------------------------------------|
| 19712 |         |       | The first character following the permissions specifies if there is an alternate  |
| 19713 |         |       | or additional access control method associated with the facility. If there is no  |
| 19714 |         |       | alternate or additional access control method associated with the facility, a     |
| 19715 |         |       | single <space> shall be written; otherwise, another printable character is        |
| 19716 |         |       | written.                                                                          |
| 19717 | OWNER   | (all) | The user name of the owner of the facility entry. If the user name of the owner   |
| 19718 |         |       | is found in the user database, at least the first eight column positions of the   |
| 19719 |         |       | name shall be written using the format %s. Otherwise, the user ID of the          |
| 19720 |         |       | owner shall be written using the format %d.                                       |
| 19721 | GROUP   | (all) | The group name of the owner of the facility entry. If the group name of the       |
| 19722 |         |       | owner is found in the group database, at least the first eight column positions   |
| 19723 |         |       | of the name shall be written using the format %s. Otherwise, the group ID of      |
| 19724 |         |       | the owner shall be written using the format %d.                                   |
| 19725 |         |       | The following nine columns shall be only written out for message queues:          |
| 19726 | CREATOR | (a,c) | The user name of the creator of the facility entry. If the user name of the       |
| 19727 |         |       | creator is found in the user database, at least the first eight column positions  |
| 19728 |         |       | of the name shall be written using the format %s. Otherwise, the user ID of       |
| 19729 |         |       | the creator shall be written using the format %d.                                 |
| 19730 | CGROUP  | (a,c) | The group name of the creator of the facility entry. If the group name of the     |
| 19731 |         |       | creator is found in the group database, at least the first eight column positions |
| 19732 |         |       | of the name shall be written using the format %s. Otherwise, the group ID of      |
| 19733 |         |       | the creator shall be written using the format %d.                                 |
| 19734 | CBYTES  | (a,o) | The number of bytes in messages currently outstanding on the associated           |
| 19735 |         |       | message queue. This field shall be written using the format %d.                   |
| 19736 | QNUM    | (a,o) | The number of messages currently outstanding on the associated message            |
| 19737 |         |       | queue. This field shall be written using the format %d.                           |
| 19738 | QBYTES  | (a,b) | The maximum number of bytes allowed in messages outstanding on the                |
| 19739 |         |       | associated message queue. This field shall be written using the format %d.        |
| 19740 | LSPID   | (a,p) | The process ID of the last process to send a message to the associated queue.     |
| 19741 |         |       | This field shall be written using the format:                                     |
| 19742 |         |       | "%d", <pid>                                                                       |
| 19743 |         |       | where <pid> is 0 if no message has been sent to the corresponding message         |
| 19744 |         |       | queue; otherwise, <pid> shall be the process ID of the last process to send a     |
| 19745 |         |       | message to the queue.                                                             |
| 19746 | LRPID   | (a,p) | The process ID of the last process to receive a message from the associated       |
| 19747 |         |       | queue. This field shall be written using the format:                              |
| 19748 |         |       | "%d", <pid>                                                                       |
| 19749 |         |       | where <pid> is 0 if no message has been received from the corresponding           |
| 19750 |         |       | message queue; otherwise, <pid> shall be the process ID of the last process to    |
| 19751 |         |       | receive a message from the queue.                                                 |
| 19752 | STIME   | (a,t) | The time the last message was sent to the associated queue. If a message has      |
| 19753 |         |       | been sent to the corresponding message queue, the hour, minute, and second        |
| 19754 |         |       | of the last time a message was sent to the queue shall be written using the       |
| 19755 |         |       | format %d:%2.2d:%2.2d. Otherwise, the format "no-entry" shall be                  |
| 19756 |         |       | written.                                                                          |

19757 RTIME (a,t) The time the last message was received from the associated queue. If a  
 19758 message has been received from the corresponding message queue, the hour,  
 19759 minute, and second of the last time a message was received from the queue  
 19760 shall be written using the format %d:%2.2d:%2.2d. Otherwise, the format  
 19761 " no-entry" shall be written.

19762 The following eight columns shall be only written out for shared memory segments.

19763 CREATOR (a,c) The user of the creator of the facility entry. If the user name of the creator is  
 19764 found in the user database, at least the first eight column positions of the  
 19765 name shall be written using the format %s. Otherwise, the user ID of the  
 19766 creator shall be written using the format %d.

19767 CGROUP (a,c) The group name of the creator of the facility entry. If the group name of the  
 19768 creator is found in the group database, at least the first eight column positions  
 19769 of the name shall be written using the format %s. Otherwise, the group ID of  
 19770 the creator shall be written using the format %d.

19771 NATTCH (a,o) The number of processes attached to the associated shared memory segment.  
 19772 This field shall be written using the format %d.

19773 SEGSZ (a,b) The size of the associated shared memory segment. This field shall be written  
 19774 using the format %d.

19775 CPID (a,p) The process ID of the creator of the shared memory entry. This field shall be  
 19776 written using the format %d.

19777 LPID (a,p) The process ID of the last process to attach or detach the shared memory  
 19778 segment. This field shall be written using the format:

19779 "%d", <pid>

19780 where <pid> is 0 if no process has attached the corresponding shared memory  
 19781 segment; otherwise, <pid> shall be the process ID of the last process to attach  
 19782 or detach the segment.

19783 ATIME (a,t) The time the last attach on the associated shared memory segment was  
 19784 completed. If the corresponding shared memory segment has ever been  
 19785 attached, the hour, minute, and second of the last time the segment was  
 19786 attached shall be written using the format %d:%2.2d:%2.2d. Otherwise, the  
 19787 format " no-entry" shall be written.

19788 DTIME (a,t) The time the last detach on the associated shared memory segment was  
 19789 completed. If the corresponding shared memory segment has ever been  
 19790 detached, the hour, minute, and second of the last time the segment was  
 19791 detached shall be written using the format %d:%2.2d:%2.2d. Otherwise, the  
 19792 format " no-entry" shall be written.

19793 The following four columns shall be only written out for semaphore sets:

19794 CREATOR (a,c) The user of the creator of the facility entry. If the user name of the creator is  
 19795 found in the user database, at least the first eight column positions of the  
 19796 name shall be written using the format %s. Otherwise, the user ID of the  
 19797 creator shall be written using the format %d.

19798 CGROUP (a,c) The group name of the creator of the facility entry. If the group name of the  
 19799 creator is found in the group database, at least the first eight column positions  
 19800 of the name shall be written using the format %s. Otherwise, the group ID of  
 19801 the creator shall be written using the format %d.

- 19802 NSEMS (a,b) The number of semaphores in the set associated with the semaphore entry.  
19803 This field shall be written using the format %d.
- 19804 OTIME (a,t) The time the last semaphore operation on the set associated with the  
19805 semaphore entry was completed. If a semaphore operation has ever been  
19806 performed on the corresponding semaphore set, the hour, minute, and second  
19807 of the last semaphore operation on the semaphore set shall be written using  
19808 the format %d:%2.2d:%2.2d. Otherwise, the format " no-entry" shall be  
19809 written.
- 19810 The following column shall be written for all three reports when it is requested:
- 19811 CTIME (a,t) The time the associated entry was created or changed. The hour, minute, and  
19812 second of the time when the associated entry was created shall be written  
19813 using the format %d:%2.2d:%2.2d.
- 19814 **STDERR**
- 19815 The standard error shall be used only for diagnostic messages.
- 19816 **OUTPUT FILES**
- 19817 None.
- 19818 **EXTENDED DESCRIPTION**
- 19819 None.
- 19820 **EXIT STATUS**
- 19821 The following exit values shall be returned:
- 19822 0 Successful completion.
- 19823 >0 An error occurred.
- 19824 **CONSEQUENCES OF ERRORS**
- 19825 Default.
- 19826 **APPLICATION USAGE**
- 19827 Things can change while *ipcs* is running; the information it gives is guaranteed to be accurate  
19828 only when it was retrieved.
- 19829 **EXAMPLES**
- 19830 None.
- 19831 **RATIONALE**
- 19832 None.
- 19833 **FUTURE DIRECTIONS**
- 19834 None.
- 19835 **SEE ALSO**
- 19836 The System Interfaces volume of IEEE Std 1003.1-200x, *msgop()*, *msgrcv()*, *msgsnd()*, *semget()*,  
19837 *semop()*, *shmat()*, *shmdt()*, *shmget()*, *shmop()*
- 19838 **CHANGE HISTORY**
- 19839 First released in Issue 5.
- 19840 **Issue 6**
- 19841 The Open Group Corrigendum U020/1 is applied, correcting the SYNOPSIS.
- 19842 The Open Group Corrigendum U032/1 and U032/2 are applied, clarifying the output format.
- 19843 The Open Group Base Resolution bwg98-004 is applied.

19844 **NAME**

19845 jobs — display status of jobs in the current session

19846 **SYNOPSIS**19847 UP jobs [-l | -p][*job\_id*...]

19848

19849 **DESCRIPTION**19850 The *jobs* utility shall display the status of jobs that were started in the current shell environment;  
19851 see Section 2.12 (on page 2263).19852 When *jobs* reports the termination status of a job, the shell shall remove its process ID from the  
19853 list of those “known in the current shell execution environment”; see Section 2.9.3.1 (on page  
19854 2252).19855 **OPTIONS**19856 The *jobs* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section  
19857 12.2, Utility Syntax Guidelines.

19858 The following options shall be supported:

19859 **-l** (The letter ell.) Provide more information about each job listed. This information  
19860 shall include the job number, current job, process group ID, state, and the  
19861 command that formed the job.19862 **-p** Display only the process IDs for the process group leaders of the selected jobs.19863 By default, the *jobs* utility shall display the status of all stopped jobs, running background jobs  
19864 and all jobs whose status has changed and have not been reported by the shell.19865 **OPERANDS**

19866 The following operand shall be supported:

19867 *job\_id* Specifies the jobs for which the status is to be displayed. If no *job\_id* is given, the  
19868 status information for all jobs shall be displayed. The format of *job\_id* is described  
19869 in the Base Definitions volume of IEEE Std 1003.1-200x, Section 3.203, Job Control  
19870 Job ID.19871 **STDIN**

19872 Not used.

19873 **INPUT FILES**

19874 None.

19875 **ENVIRONMENT VARIABLES**19876 The following environment variables shall affect the execution of *jobs*:19877 **LANG** Provide a default value for the internationalization variables that are unset or null.  
19878 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
19879 Internationalization Variables for the precedence of internationalization variables  
19880 used to determine the values of locale categories.)19881 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
19882 internationalization variables.19883 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
19884 characters (for example, single-byte as opposed to multi-byte characters in  
19885 arguments).19886 **LC\_MESSAGES**

19887 Determine the locale that should be used to affect the format and contents of

- 19888 diagnostic messages written to standard error and informative messages written to  
19889 standard output.
- 19890 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 19891 **ASYNCHRONOUS EVENTS**
- 19892 Default.
- 19893 **STDOUT**
- 19894 If the **-p** option is specified, the output shall consist of one line for each process ID:
- 19895 "%d\n", <process ID>
- 19896 Otherwise, if the **-l** option is not specified, the output shall be a series of lines of the form:
- 19897 "[%d] %c %s %s\n", <job-number>, <current>, <state>, <command>
- 19898 where the fields shall be as follows:
- 19899 <current> The character '+' identifies the job that would be used as a default for the *fg* or *bg*  
19900 utilities; this job can also be specified using the *job\_id* %+ or "%%". The character  
19901 '-' identifies the job that would become the default if the current default job were  
19902 to exit; this job can also be specified using the *job\_id* %-. For other jobs, this field is  
19903 a <space>. At most one job can be identified with '+' and at most one job can be  
19904 identified with '-'. If there is any suspended job, then the current job shall be a  
19905 suspended job. If there are at least two suspended jobs, then the previous job also  
19906 shall be a suspended job.
- 19907 <job-number> A number that can be used to identify the process group to the *wait*, *fg*, *bg*, and *kill*  
19908 utilities. Using these utilities, the job can be identified by prefixing the job number  
19909 with '% '.
- 19910 <state> One of the following strings (in the POSIX locale):
- 19911 **Running** Indicates that the job has not been suspended by a signal and has not  
19912 exited.
- 19913 **Done** Indicates that the job completed and returned exit status zero.
- 19914 **Done(code)** Indicates that the job completed normally and that it exited with the  
19915 specified non-zero exit status, *code*, expressed as a decimal number.
- 19916 **Stopped** Indicates that the job was suspended by the SIGTSTP signal.
- 19917 **Stopped (SIGTSTP)**
- 19918 Indicates that the job was suspended by the SIGTSTP signal.
- 19919 **Stopped (SIGSTOP)**
- 19920 Indicates that the job was suspended by the SIGSTOP signal.
- 19921 **Stopped (SIGTTIN)**
- 19922 Indicates that the job was suspended by the SIGTTIN signal.
- 19923 **Stopped (SIGTTOU)**
- 19924 Indicates that the job was suspended by the SIGTTOU signal.
- 19925 The implementation may substitute the string **Suspended** in place of **Stopped**. If  
19926 the job was terminated by a signal, the format of <state> is unspecified, but it shall  
19927 be visibly distinct from all of the other <state> formats shown here and shall  
19928 indicate the name or description of the signal causing the termination.

- 19929           <*command*> The associated command that was given to the shell.
- 19930           If the **-I** option is specified, a field containing the process group ID shall be inserted before the
- 19931           <*state*> field. Also, more processes in a process group may be output on separate lines, using
- 19932           only the process ID and <*command*> fields.
- 19933 **STDERR**
- 19934           The standard error shall be used only for diagnostic messages.
- 19935 **OUTPUT FILES**
- 19936           None.
- 19937 **EXTENDED DESCRIPTION**
- 19938           None.
- 19939 **EXIT STATUS**
- 19940           The following exit values shall be returned:
- 19941           0 Successful completion.
- 19942           >0 An error occurred.
- 19943 **CONSEQUENCES OF ERRORS**
- 19944           Default.
- 19945 **APPLICATION USAGE**
- 19946           The **-p** option is the only portable way to find out the process group of a job because different
- 19947           implementations have different strategies for defining the process group of the job. Usage such
- 19948           as  $\$(jobs -p)$  provides a way of referring to the process group of the job in an implementation-
- 19949           independent way.
- 19950           The *jobs* utility does not work as expected when it is operating in its own utility execution
- 19951           environment because that environment has no applicable jobs to manipulate. See the
- 19952           APPLICATION USAGE section for *bg* (on page 2410). For this reason, *jobs* is generally
- 19953           implemented as a shell regular built-in.
- 19954 **EXAMPLES**
- 19955           None.
- 19956 **RATIONALE**
- 19957           Both "%%" and "%+" are used to refer to the current job. Both forms are of equal validity—the
- 19958           "%%" mirroring "\$\$" and "%+" mirroring the output of *jobs*. Both forms reflect historical
- 19959           practice of the KornShell and the C shell with job control.
- 19960           The job control features provided by *bg*, *fg*, and *jobs* are based on the KornShell. The standard
- 19961           developers examined the characteristics of the C shell versions of these utilities and found that
- 19962           differences exist. Despite widespread use of the C shell, the KornShell versions were selected for
- 19963           this volume of IEEE Std 1003.1-200x to maintain a degree of uniformity with the rest of the
- 19964           KornShell features selected (such as the very popular command line editing features).
- 19965           The *jobs* utility is not dependent on the job control option, as are the seemingly related *bg* and *fg*
- 19966           utilities because *jobs* is useful for examining background jobs, regardless of the condition of job
- 19967           control. When the user has invoked a *set +m* command and job control has been turned off, *jobs*
- 19968           can still be used to examine the background jobs associated with that current session. Similarly,
- 19969           *kill* can then be used to kill background jobs with *kill% <background job number>*.
- 19970           The output for terminated jobs is left unspecified to accommodate various historical systems.
- 19971           The following formats have been witnessed:

19972 1. **Killed**(*signal name*)

19973 2. *signal name*

19974 3. *signal name*(**coredump**)

19975 4. *signal description*– **core dumped**

19976 Most users should be able to understand these formats, although it means that applications have  
19977 trouble parsing them.

19978 The calculation of job IDs was not described since this would suggest an implementation, which  
19979 may impose unnecessary restrictions.

19980 In an early proposal, a **-n** option was included to “Display the status of jobs that have changed,  
19981 exited, or stopped since the last status report”. It was removed because the shell always writes  
19982 any changed status of jobs before each prompt.

19983 **FUTURE DIRECTIONS**

19984 None.

19985 **SEE ALSO**

19986 *bg, fg, kill, wait*

19987 **CHANGE HISTORY**

19988 First released in Issue 4.

19989 **Issue 6**

19990 This utility is now marked as part of the User Portability Utilities option.

19991 The JC shading is removed as job control is mandatory in this issue.

## 19992 NAME

19993 join — relational database operator

## 19994 SYNOPSIS

19995 join [-a *file\_number* | -v *file\_number*][-e *string*][-o *list*][-t *char*]  
 19996 [-1 *field*][-2 *field*] *file1 file2*

## 19997 DESCRIPTION

19998 The *join* utility shall perform an equality join on the files *file1* and *file2*. The joined files shall be  
 19999 written to the standard output.

20000 The join field is a field in each file on which the files are compared. The *join* utility shall write |  
 20001 one line in the output for each pair of lines in *file1* and *file2* that have identical join fields. The |  
 20002 output line by default shall consist of the join field, then the remaining fields from *file1*, then the  
 20003 remaining fields from *file2*. This format can be changed by using the -o option (see below). The  
 20004 -a option can be used to add unmatched lines to the output. The -v option can be used to output  
 20005 only unmatched lines.

20006 The files *file1* and *file2* shall be ordered in the collating sequence of *sort -b* on the fields on which |  
 20007 they shall be joined, by default the first in each line. All selected output shall be written in the  
 20008 same collating sequence.

20009 The default input field separators shall be <blank>s. In this case, multiple separators shall count  
 20010 as one field separator, and leading separators shall be ignored. The default output field separator  
 20011 shall be a <space>.

20012 The field separator and collating sequence can be changed by using the -t option (see below).

20013 If the same key appears more than once in either file, all combinations of the set of remaining  
 20014 fields in *file1* and the set of remaining fields in *file2* are output in the order of the lines  
 20015 encountered.

20016 If the input files are not in the appropriate collating sequence, the results are unspecified.

## 20017 OPTIONS

20018 The *join* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section  
 20019 12.2, Utility Syntax Guidelines.

20020 The following options shall be supported:

20021 -a *file\_number*

20022 Produce a line for each unpairable line in file *file\_number*, where *file\_number* is 1 or  
 20023 2, in addition to the default output. If both -a1 and -a2 are specified, all unpairable  
 20024 lines shall be output.

20025 -e *string* Replace empty output fields in the list selected by -o with the string *string*.

20026 -o *list* Construct the output line to comprise the fields specified in *list*, each element of  
 20027 which shall have one of the following two forms:

20028 1. *file\_number.field*, where *file\_number* is a file number and *field* is a decimal  
 20029 integer field number

20030 2. 0 (zero), representing the join field

20031 The elements of *list* shall be either comma-separated or <blank>-separated, as  
 20032 specified in Guideline 8 of the Base Definitions volume of IEEE Std 1003.1-200x,  
 20033 Section 12.2, Utility Syntax Guidelines. The fields specified by *list* shall be written  
 20034 for all selected output lines. Fields selected by *list* that do not appear in the input  
 20035 shall be treated as empty output fields. (See the -e option.) Only specifically

20036 requested fields shall be written. The application shall ensure that *list* is a single  
20037 command line argument.

20038 **-t *char*** Use character *char* as a separator, for both input and output. Every appearance of  
20039 *char* in a line shall be significant. When this option is specified, the collating  
20040 sequence shall be the same as *sort* without the **-b** option.

20041 **-v *file\_number***  
20042 Instead of the default output, produce a line only for each unpairable line in  
20043 *file\_number*, where *file\_number* is 1 or 2. If both **-v1** and **-v2** are specified, all  
20044 unpairable lines shall be output.

20045 **-1 *field*** Join on the *field*th field of file 1. Fields are decimal integers starting with 1.

20046 **-2 *field*** Join on the *field*th field of file 2. Fields are decimal integers starting with 1.

#### 20047 OPERANDS

20048 The following operands shall be supported:

20049 *file1, file2*

20050 A pathname of a file to be joined. If either of the *file1* or *file2* operands is '-', the  
20051 standard input shall be used in its place.

#### 20052 STDIN

20053 The standard input shall be used only if the *file1* or *file2* operand is '-'. See the INPUT FILES  
20054 section.

#### 20055 INPUT FILES

20056 The input files shall be text files.

#### 20057 ENVIRONMENT VARIABLES

20058 The following environment variables shall affect the execution of *join*:

20059 **LANG** Provide a default value for the internationalization variables that are unset or null.  
20060 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
20061 Internationalization Variables for the precedence of internationalization variables  
20062 used to determine the values of locale categories.)

20063 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
20064 internationalization variables.

20065 **LC\_COLLATE**

20066 Determine the locale of the collating sequence *join* expects to have been used when  
20067 the input files were sorted.

20068 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
20069 characters (for example, single-byte as opposed to multi-byte characters in  
20070 arguments and input files).

20071 **LC\_MESSAGES**

20072 Determine the locale that should be used to affect the format and contents of  
20073 diagnostic messages written to standard error.

20074 XSI **NLSPATH** Determine the location of message catalogs for the processing of **LC\_MESSAGES**.

#### 20075 ASYNCHRONOUS EVENTS

20076 Default.

20077 **STDOUT**

20078 The *join* utility output shall be a concatenation of selected character fields. When the **-o** option  
 20079 is not specified, the output shall be:

20080 "%s%s%s\n", <join field>, <other file1 fields>,  
 20081 <other file2 fields>

20082 If the join field is not the first field in a file, the <other file fields> for that file shall be:

20083 <fields preceding join field>, <fields following join field>

20084 When the **-o** option is specified, the output format shall be:

20085 "%s\n", <concatenation of fields>

20086 where the concatenation of fields is described by the **-o** option, above.

20087 For either format, each field (except the last) shall be written with its trailing separator character.  
 20088 If the separator is the default (<blank>s), a single <space> shall be written after each field  
 20089 (except the last).

20090 **STDERR**

20091 The standard error shall be used only for diagnostic messages.

20092 **OUTPUT FILES**

20093 None.

20094 **EXTENDED DESCRIPTION**

20095 None.

20096 **EXIT STATUS**

20097 The following exit values shall be returned:

20098 0 All input files were output successfully.

20099 >0 An error occurred.

20100 **CONSEQUENCES OF ERRORS**

20101 Default.

20102 **APPLICATION USAGE**

20103 Pathnames consisting of numeric digits or of the form *string.string* should not be specified  
 20104 directly following the **-o** list.

20105 **EXAMPLES**

20106 The **-o 0** field essentially selects the union of the join fields. For example, given file **phone**:

|       |         |                 |
|-------|---------|-----------------|
| 20107 | !Name   | Phone Number    |
| 20108 | Don     | +1 123-456-7890 |
| 20109 | Hal     | +1 234-567-8901 |
| 20110 | Yasushi | +2 345-678-9012 |

20111 and file **fax**:

|       |         |                 |
|-------|---------|-----------------|
| 20112 | !Name   | Fax Number      |
| 20113 | Don     | +1 123-456-7899 |
| 20114 | Keith   | +1 456-789-0122 |
| 20115 | Yasushi | +2 345-678-9011 |

20116 (where the large expanses of white space are meant to each represent a single <tab>), the  
 20117 command:

20118 `join -t "<tab>" -a 1 -a 2 -e '(unknown)' -o 0,1.2,2.2 phone fax`

20119 would produce:

| 20120 | !Name   | Phone Number    | Fax Number      |
|-------|---------|-----------------|-----------------|
| 20121 | Don     | +1 123-456-7890 | +1 123-456-7899 |
| 20122 | Hal     | +1 234-567-8901 | (unknown)       |
| 20123 | Keith   | (unknown)       | +1 456-789-0122 |
| 20124 | Yasushi | +2 345-678-9012 | +2 345-678-9011 |

20125 Multiple instances of the same key will produce combinatorial results. The following:

20126 fa:  
 20127 a x  
 20128 a y  
 20129 a z  
 20130 fb:  
 20131 a p

20132 will produce:

20133 a x p  
 20134 a y p  
 20135 a z p

20136 And the following:

20137 fa:  
 20138 a b c  
 20139 a d e  
 20140 fb:  
 20141 a w x  
 20142 a y z  
 20143 a o p

20144 will produce:

20145 a b c w x  
 20146 a b c y z  
 20147 a b c o p  
 20148 a d e w x  
 20149 a d e y z  
 20150 a d e o p

#### 20151 RATIONALE

20152 The `-e` option is only effective when used with `-o` because, unless specific fields are identified  
 20153 using `-o`, *join* is not aware of what fields might be empty. The exception to this is the join field,  
 20154 but identifying an empty join field with the `-e` string is not historical practice and some scripts  
 20155 might break if this were changed.

20156 The 0 field in the `-o` list was adopted from the Tenth Edition version of *join* to satisfy  
 20157 international objections that the *join* in the base documents do not support the “full join” or  
 20158 “outer join” described in relational database literature. Although it has been possible to include  
 20159 a join field in the output (by default, or by field number using `-o`), the join field could not be  
 20160 included for an unpaired line selected by `-a`. The `-o 0` field essentially selects the union of the  
 20161 join fields.

20162 This sort of outer join was not possible with the *join* commands in the base documents. The `-o 0`  
 20163 field was chosen because it is an upward-compatible change for applications. An alternative was

- 20164 considered: have the join field represent the union of the fields in the files (where they are  
20165 identical for matched lines, and one or both are null for unmatched lines). This was not adopted  
20166 because it would break some historical applications.
- 20167 The ability to specify *file2* as – is not historical practice; it was added for completeness.
- 20168 The –v option is not historical practice, but was considered necessary because it permitted the  
20169 writing of *only* those lines that do not match on the join field, as opposed to the –a option, which  
20170 prints both lines that do and do not match. This additional facility is parallel with the –v option  
20171 of *grep*.
- 20172 Some historical implementations have been encountered where a blank line in one of the input  
20173 files was considered to be the end of the file; the description in this volume of  
20174 IEEE Std 1003.1-200x does not cite this as an allowable case.
- 20175 **FUTURE DIRECTIONS**
- 20176 None.
- 20177 **SEE ALSO**
- 20178 *awk, comm, sort, uniq*
- 20179 **CHANGE HISTORY**
- 20180 First released in Issue 2.
- 20181 **Issue 6**
- 20182 The obsolescent –j options and the multi-argument –o option are withdrawn in this issue.
- 20183 The normative text is reworded to avoid use of the term “must” for application requirements.

## 20184 NAME

20185 kill — terminate or signal processes

## 20186 SYNOPSIS

20187 kill -s *signal\_name* *pid* ...

20188 kill -l [*exit\_status*]

20189 XSI kill [-*signal\_name*] *pid* ...

20190 kill [-*signal\_number*] *pid* ...

20191

## 20192 DESCRIPTION

20193 The *kill* utility shall send a signal to the process or processes specified by each *pid* operand.

20194 For each *pid* operand, the *kill* utility shall perform actions equivalent to the *kill()* function  
20195 defined in the System Interfaces volume of IEEE Std 1003.1-200x called with the following  
20196 arguments:

- 20197 • The value of the *pid* operand shall be used as the *pid* argument.
- 20198 • The *sig* argument is the value specified by the *-s* option, *-signal\_number* option, or the  
20199 *-signal\_name* option, or by SIGTERM, if none of these options is specified.

## 20200 OPTIONS

20201 The *kill* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section  
20202 XSI 12.2, Utility Syntax Guidelines, except that in the last two SYMOPSIS forms, the *-signal\_number*  
20203 and *-signal\_name* options are usually more than a single character.

20204 The following options shall be supported:

20205 -l (The letter ell.) Write all values of *signal\_name* supported by the implementation, if  
20206 no operand is given. If an *exit\_status* operand is given and it is a value of the ' ? '  
20207 shell special parameter (see Section 2.5.2 (on page 2235) and *wait* (on page 3239))  
20208 corresponding to a process that was terminated by a signal, the *signal\_name*  
20209 corresponding to the signal that terminated the process shall be written. If an  
20210 *exit\_status* operand is given and it is the unsigned decimal integer value of a signal  
20211 number, the *signal\_name* (the symbolic constant name without the **SIG** prefix  
20212 defined in the Base Definitions volume of IEEE Std 1003.1-200x) corresponding to  
20213 that signal shall be written. Otherwise, the results are unspecified.

20214 -s *signal\_name*

20215 Specify the signal to send, using one of the symbolic names defined in the  
20216 <**signal.h**> header. Values of *signal\_name* shall be recognized in a case-independent  
20217 fashion, without the **SIG** prefix. In addition, the symbolic name 0 shall be  
20218 recognized, representing the signal value zero. The corresponding signal shall be  
20219 sent instead of SIGTERM.

20220 XSI -*signal\_name*

20221 Equivalent to *-s signal\_name*.

20222 XSI -*signal\_number*

20223 Specify a non-negative decimal integer, *signal\_number*, representing the signal to  
20224 be used instead of SIGTERM, as the *sig* argument in the effective call to *kill()*. The  
20225 correspondence between integer values and the *sig* value used is shown in the  
20226 following table.

20227 The effects of specifying any *signal\_number* other than those listed in the table are  
20228 undefined.

20229

20230

20231 XSI

20232

20233

20234

20235

20236

20237

20238

| <i>signal_number</i> | <i>sig Value</i> |
|----------------------|------------------|
| 0                    | 0                |
| 1                    | SIGHUP           |
| 2                    | SIGINT           |
| 3                    | SIGQUIT          |
| 6                    | SIGABRT          |
| 9                    | SIGKILL          |
| 14                   | SIGALRM          |
| 15                   | SIGTERM          |

20239

20240

If the first argument is a negative integer, it shall be interpreted as a *-signal\_number* option, not as a negative *pid* operand specifying a process group.

20241 **OPERANDS**

20242

The following operands shall be supported:

20243

*pid*

One of the following:

20244

20245

20246

20247

20248

20249

20250

20251

1. A decimal integer specifying a process or process group to be signaled. The process or processes selected by positive, negative and zero values of the *pid* operand shall be as described for the *kill()* function defined in the System Interfaces volume of IEEE Std 1003.1-200x. If process number 0 is specified, all processes in the current process group shall be signaled. For the effects of negative *pid* numbers, see the *kill()* function defined in the System Interfaces volume of IEEE Std 1003.1-200x. If the first *pid* operand is negative, it should be preceded by "--" to keep it from being interpreted as an option.

20252

20253

20254

20255

20256

2. A job control job ID (see the Base Definitions volume of IEEE Std 1003.1-200x, Section 3.203, Job Control Job ID) that identifies a background process group to be signaled. The job control job ID notation is applicable only for invocations of *kill* in the current shell execution environment; see Section 2.12 (on page 2263).

20257

20258

*exit\_status*

A decimal integer specifying a signal number or the exit status of a process terminated by a signal.

20259 **STDIN**

20260

Not used.

20261 **INPUT FILES**

20262

None.

20263 **ENVIRONMENT VARIABLES**

20264

The following environment variables shall affect the execution of *kill*:

20265

20266

20267

20268

*LANG*

Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)

20269

20270

*LC\_ALL*

If set to a non-empty string value, override the values of all the other internationalization variables.

20271

20272

20273

*LC\_CTYPE*

Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

20274 **LC\_MESSAGES**  
 20275 Determine the locale that should be used to affect the format and contents of  
 20276 diagnostic messages written to standard error.

20277 XSI **NLSPATH** Determine the location of message catalogs for the processing of **LC\_MESSAGES**.

20278 **ASYNCHRONOUS EVENTS**  
 20279 Default.

20280 **STDOUT**  
 20281 When the **-I** option is not specified, the standard output shall not be used.  
 20282 When the **-I** option is specified, the symbolic name of each signal shall be written in the  
 20283 following format:  
 20284 "%s%c", <signal\_name>, <separator>  
 20285 where the <signal\_name> is in uppercase, without the **SIG** prefix, and the <separator> shall be  
 20286 either a <newline> or a <space>. For the last signal written, <separator> shall be a <newline>.  
 20287 When both the **-I** option and *exit\_status* operand are specified, the symbolic name of the  
 20288 corresponding signal shall be written in the following format:  
 20289 "%s\n", <signal\_name>

20290 **STDERR**  
 20291 The standard error shall be used only for diagnostic messages.

20292 **OUTPUT FILES**  
 20293 None.

20294 **EXTENDED DESCRIPTION**  
 20295 None.

20296 **EXIT STATUS**  
 20297 The following exit values shall be returned:  
 20298 0 At least one matching process was found for each *pid* operand, and the specified signal was  
 20299 successfully processed for at least one matching process.  
 20300 >0 An error occurred.

20301 **CONSEQUENCES OF ERRORS**  
 20302 Default.

20303 **APPLICATION USAGE**  
 20304 Process numbers can be found by using *ps*.  
 20305 The job control job ID notation is not required to work as expected when *kill* is operating in its  
 20306 own utility execution environment. In either of the following examples:  
 20307 nohup kill %1 &  
 20308 system("kill %1");  
 20309 the *kill* operates in a different environment and does not share the shell's understanding of job  
 20310 numbers.

20311 **EXAMPLES**  
 20312 Any of the commands:  
 20313 kill -9 100 -165  
 20314 kill -s kill 100 -165  
 20315 kill -s KILL 100 -165

20316 sends the SIGKILL signal to the process whose process ID is 100 and to all processes whose  
 20317 process group ID is 165, assuming the sending process has permission to send that signal to the  
 20318 specified processes, and that they exist.

20319 The System Interfaces volume of IEEE Std 1003.1-200x and this volume of IEEE Std 1003.1-200x  
 20320 do not require specific signal numbers for any *signal\_names*. Even the *-signal\_number* option  
 20321 provides symbolic (although numeric) names for signals. If a process is terminated by a signal,  
 20322 its exit status indicates the signal that killed it, but the exact values are not specified. The *kill -l*  
 20323 option, however, can be used to map decimal signal numbers and exit status values into the  
 20324 name of a signal. The following example reports the status of a terminated job:

```
20325 job
20326 stat=$?
20327 if [ $stat -eq 0 ]
20328 then
20329     echo job completed successfully.
20330 elif [ $stat -gt 128 ]
20331 then
20332     echo job terminated by signal SIG$(kill -l $stat).
20333 else
20334     echo job terminated with error code $stat.
20335 fi
```

20336 To send the default signal to a process group (say 123), an application should use a command |  
 20337 similar to one of the following: |

```
20338 kill -TERM -123
20339 kill -- -123
```

#### 20340 RATIONALE

20341 The *-l* option originated from the C shell, and is also implemented in the KornShell. The C shell  
 20342 output can consist of multiple output lines because the signal names do not always fit on a  
 20343 single line on some terminal screens. The KornShell output also included the implementation-  
 20344 defined signal numbers and was considered by the standard developers to be too difficult for  
 20345 scripts to parse conveniently. The specified output format is intended not only to accommodate  
 20346 the historical C shell output, but also to permit an entirely vertical or entirely horizontal listing  
 20347 on systems for which this is appropriate.

20348 An early proposal invented the name SIGNULL as a *signal\_name* for signal 0 (used by the System  
 20349 Interfaces volume of IEEE Std 1003.1-200x to test for the existence of a process without sending  
 20350 it a signal). Since the *signal\_name* 0 can be used in this case unambiguously, SIGNULL has been  
 20351 removed.

20352 An early proposal also required symbolic *signal\_names* to be recognized with or without the **SIG**  
 20353 prefix. Historical versions of *kill* have not written the **SIG** prefix for the *-l* option and have not  
 20354 recognized the **SIG** prefix on *signal\_names*. Since neither applications portability nor ease-of-use  
 20355 would be improved by requiring this extension, it is no longer required.

20356 To avoid an ambiguity of an initial negative number argument specifying either a signal number |  
 20357 or a process group, IEEE Std 1003.1-200x mandates that it is always considered the former by |  
 20358 implementations that support the XSI option. It also requires that conforming applications |  
 20359 always use the "--" options terminator argument when specifying a process group, unless an |  
 20360 option is also specified. |

20361 The *-s* option was added in response to international interest in providing some form of *kill* that |  
 20362 meets the Utility Syntax Guidelines. |

20363 The job control job ID notation is not required to work as expected when *kill* is operating in its  
20364 own utility execution environment. In either of the following examples:

```
20365 nohup kill %1 &  
20366 system("kill %1");
```

20367 the *kill* operates in a different environment and does not understand how the shell has managed  
20368 its job numbers.

20369 **FUTURE DIRECTIONS**

20370 None.

20371 **SEE ALSO**

20372 *ps*, *wait*, the System Interfaces volume of IEEE Std 1003.1-200x, *kill()*, the Base Definitions |  
20373 volume of IEEE Std 1003.1-200x, <**signal.h**> |

20374 **CHANGE HISTORY**

20375 First released in Issue 2.

20376 **Issue 6**

20377 The obsolescent versions of the SYNOPSIS were turned into non-obsolescent features of the XSI |  
20378 option, corresponding to a similar change in the *trap* special built-in. |

20379 **NAME**20380 lex — generate programs for lexical tasks (**DEVELOPMENT**)20381 **SYNOPSIS**20382 CD lex [-t][-n|-v][file ...]  
2038320384 **DESCRIPTION**

20385 The *lex* utility shall generate C programs to be used in lexical processing of character input, and  
 20386 that can be used as an interface to *yacc*. The C programs shall be generated from *lex* source code  
 20387 and conform to the ISO C standard. Usually, the *lex* utility shall write the program it generates to  
 20388 the file **lex.yy.c**; the state of this file is unspecified if *lex* exits with a non-zero exit status. See the  
 20389 EXTENDED DESCRIPTION section for a complete description of the *lex* input language.

20390 **OPTIONS**

20391 The *lex* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2,  
 20392 Utility Syntax Guidelines.

20393 The following options shall be supported:

20394 **-n** Suppress the summary of statistics usually written with the **-v** option. If no table  
 20395 sizes are specified in the *lex* source code and the **-v** option is not specified, then **-n**  
 20396 is implied.

20397 **-t** Write the resulting program to standard output instead of **lex.yy.c**.

20398 **-v** Write a summary of *lex* statistics to the standard output. (See the discussion of *lex*  
 20399 table sizes in **Definitions in lex** (on page 2736).) If the **-t** option is specified and  
 20400 **-n** is not specified, this report shall be written to standard error. If table sizes are  
 20401 specified in the *lex* source code, and if the **-n** option is not specified, the **-v** option  
 20402 may be enabled.

20403 **OPERANDS**

20404 The following operand shall be supported:

20405 *file* A pathname of an input file. If more than one such *file* is specified, all files shall be  
 20406 concatenated to produce a single *lex* program. If no *file* operands are specified, or if  
 20407 a *file* operand is '-', the standard input shall be used.

20408 **STDIN**

20409 The standard input shall be used if no *file* operands are specified, or if a *file* operand is '-'. See  
 20410 INPUT FILES.

20411 **INPUT FILES**

20412 The input files shall be text files containing *lex* source code, as described in the EXTENDED  
 20413 DESCRIPTION section.

20414 **ENVIRONMENT VARIABLES**

20415 The following environment variables shall affect the execution of *lex*:

20416 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 20417 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
 20418 Internationalization Variables for the precedence of internationalization variables  
 20419 used to determine the values of locale categories.)

20420 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 20421 internationalization variables.

20422 **LC\_COLLATE**

20423 Determine the locale for the behavior of ranges, equivalence classes and multi-

20424 character collating elements within regular expressions. If this variable is not set to  
20425 the POSIX locale, the results are unspecified.

20426 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
20427 characters (for example, single-byte as opposed to multi-byte characters in  
20428 arguments and input files), and the behavior of character classes within regular  
20429 expressions. If this variable is not set to the POSIX locale, the results are  
20430 unspecified.

20431 **LC\_MESSAGES**  
20432 Determine the locale that should be used to affect the format and contents of  
20433 diagnostic messages written to standard error.

20434 XSI **NLSPATH** Determine the location of message catalogs for the processing of **LC\_MESSAGES**.

#### 20435 ASYNCHRONOUS EVENTS

20436 Default.

#### 20437 STDOUT

20438 If the **-t** option is specified, the text file of C source code output of *lex* shall be written to  
20439 standard output.

20440 If the **-t** option is not specified:

20441 • Implementation-defined informational, error, and warning messages concerning the contents  
20442 of *lex* source code input shall be written to either the standard output or standard error.

20443 • If the **-v** option is specified and the **-n** option is not specified, *lex* statistics shall also be  
20444 written to either the standard output or standard error, in an implementation-defined format.  
20445 These statistics may also be generated if table sizes are specified with a '%' operator in the  
20446 *Definitions* section, as long as the **-n** option is not specified.

#### 20447 STDERR

20448 If the **-t** option is specified, implementation-defined informational, error, and warning messages  
20449 concerning the contents of *lex* source code input shall be written to the standard error.

20450 If the **-t** option is not specified:

20451 1. Implementation-defined informational, error, and warning messages concerning the  
20452 contents of *lex* source code input shall be written to either the standard output or standard  
20453 error.

20454 2. If the **-v** option is specified and the **-n** option is not specified, *lex* statistics shall also be  
20455 written to either the standard output or standard error, in an implementation-defined  
20456 format. These statistics may also be generated if table sizes are specified with a '%' operator  
20457 in the *Definitions* section, as long as the **-n** option is not specified.

#### 20458 OUTPUT FILES

20459 A text file containing C source code shall be written to **lex.yy.c**, or to the standard output if the  
20460 **-t** option is present.

#### 20461 EXTENDED DESCRIPTION

20462 Each input file shall contain *lex* source code, which is a table of regular expressions with  
20463 corresponding actions in the form of C program fragments.

20464 When **lex.yy.c** is compiled and linked with the *lex* library (using the **-ll** operand with *c99*), the  
20465 resulting program shall read character input from the standard input and shall partition it into  
20466 strings that match the given expressions.

- 20467 When an expression is matched, these actions shall occur:
- 20468 • The input string that was matched shall be left in *ytext* as a null-terminated string; *ytext* |
  - 20469 shall either be an external character array or a pointer to a character string. As explained in |
  - 20470 **Definitions in lex**, the type can be explicitly selected using the **%array** or **%pointer** |
  - 20471 declarations, but the default is implementation-defined.
  - 20472 • The external **int** *yylen* shall be set to the length of the matching string. |
  - 20473 • The expression's corresponding program fragment, or action, shall be executed. |
- 20474 During pattern matching, *lex* shall search the set of patterns for the single longest possible
- 20475 match. Among rules that match the same number of characters, the rule given first shall be
- 20476 chosen.
- 20477 The general format of *lex* source shall be:
- 20478       *Definitions*
- 20479       %%
- 20480       *Rules*
- 20481       %%
- 20482       *UserSubroutines*
- 20483 The first "%%" is required to mark the beginning of the rules (regular expressions and actions);
- 20484 the second "%%" is required only if user subroutines follow.
- 20485 Any line in the *Definitions* section beginning with a <blank> shall be assumed to be a C program
- 20486 fragment and shall be copied to the external definition area of the **lex.yy.c** file. Similarly,
- 20487 anything in the *Definitions* section included between delimiter lines containing only "%{" and
- 20488 "%}" shall also be copied unchanged to the external definition area of the **lex.yy.c** file.
- 20489 Any such input (beginning with a <blank> or within "%{" and "%}" delimiter lines) appearing
- 20490 at the beginning of the *Rules* section before any rules are specified shall be written to **lex.yy.c**
- 20491 after the declarations of variables for the *yylex()* function and before the first line of code in
- 20492 *yylex()*. Thus, user variables local to *yylex()* can be declared here, as well as application code to
- 20493 execute upon entry to *yylex()*.
- 20494 The action taken by *lex* when encountering any input beginning with a <blank> or within "%{"
- 20495 and "%}" delimiter lines appearing in the *Rules* section but coming after one or more rules is
- 20496 undefined. The presence of such input may result in an erroneous definition of the *yylex()*
- 20497 function.
- 20498 **Definitions in lex**
- 20499 *Definitions* appear before the first "%%" delimiter. Any line in this section not contained between
- 20500 "%{" and "%}" lines and not beginning with a <blank> shall be assumed to define a *lex*
- 20501 substitution string. The format of these lines shall be:
- 20502       *name substitute*
- 20503 If a *name* does not meet the requirements for identifiers in the ISO C standard, the result is
- 20504 undefined. The string *substitute* shall replace the string {*name*} when it is used in a rule. The *name*
- 20505 string shall be recognized in this context only when the braces are provided and when it does
- 20506 not appear within a bracket expression or within double-quotes.
- 20507 In the *Definitions* section, any line beginning with a '%' (percent sign) character and followed by
- 20508 an alphanumeric word beginning with either 's' or 'S' shall define a set of start conditions.
- 20509 Any line beginning with a '%' followed by a word beginning with either 'x' or 'X' shall define
- 20510 a set of exclusive start conditions. When the generated scanner is in a "%s" state, patterns with

20511 no state specified shall be also active; in a "%x" state, such patterns shall not be active. The rest  
 20512 of the line, after the first word, shall be considered to be one or more <blank>-separated names  
 20513 of start conditions. Start condition names shall be constructed in the same way as definition  
 20514 names. Start conditions can be used to restrict the matching of regular expressions to one or  
 20515 more states as described in **Regular Expressions in lex** (on page 2738).

20516 Implementations shall accept either of the following two mutually exclusive declarations in the  
 20517 *Definitions* section:

20518 **%array**      Declare the type of *yytext* to be a null-terminated character array.

20519 **%pointer**    Declare the type of *yytext* to be a pointer to a null-terminated character string.

20520 The default type of *yytext* is implementation-defined. If an application refers to *yytext* outside of  
 20521 the scanner source file (that is, via an **extern**), the application shall include the appropriate  
 20522 **%array** or **%pointer** declaration in the scanner source file.

20523 Implementations shall accept declarations in the *Definitions* section for setting certain internal  
 20524 table sizes. The declarations are shown in the following table.

20525 **Table 4-9** Table Size Declarations in *lex*

| Declaration        | Description                        | Minimum Value |
|--------------------|------------------------------------|---------------|
| <b>%p</b> <i>n</i> | Number of positions                | 2 500         |
| <b>%n</b> <i>n</i> | Number of states                   | 500           |
| <b>%a</b> <i>n</i> | Number of transitions              | 2 000         |
| <b>%e</b> <i>n</i> | Number of parse tree nodes         | 1 000         |
| <b>%k</b> <i>n</i> | Number of packed character classes | 1 000         |
| <b>%o</b> <i>n</i> | Size of the output array           | 3 000         |

20533 In the table, *n* represents a positive decimal integer, preceded by one or more <blank>s. The  
 20534 exact meaning of these table size numbers is implementation-defined. The implementation shall  
 20535 document how these numbers affect the *lex* utility and how they are related to any output that  
 20536 may be generated by the implementation should limitations be encountered during the  
 20537 execution of *lex*. It shall be possible to determine from this output which of the table size values  
 20538 needs to be modified to permit *lex* to successfully generate tables for the input language. The  
 20539 values in the column Minimum Value represent the lowest values conforming implementations  
 20540 shall provide.

### 20541 **Rules in lex**

20542 The rules in *lex* source files are a table in which the left column contains regular expressions and  
 20543 the right column contains actions (C program fragments) to be executed when the expressions  
 20544 are recognized.

20545 *ERE* *action*

20546 *ERE* *action*

20547 ...

20548 The extended regular expression (*ERE*) portion of a row shall be separated from *action* by one or  
 20549 more <blank>s. A regular expression containing <blank>s shall be recognized under one of the  
 20550 following conditions:

- 20551 • The entire expression appears within double-quotes.
- 20552 • The <blank>s appear within double-quotes or square brackets.

- 20553           • Each <blank> is preceded by a backslash character.

20554           **User Subroutines in lex**

20555           Anything in the user subroutines section shall be copied to **lex.yy.c** following **yylex()**.

20556           **Regular Expressions in lex**

20557           The *lex* utility shall support the set of extended regular expressions (see the Base Definitions volume of IEEE Std 1003.1-200x, Section 9.4, Extended Regular Expressions), with the following additions and exceptions to the syntax:

20560           ". . ."       Any string enclosed in double-quotes shall represent the characters within the double-quotes as themselves, except that backslash escapes (which appear in the following table) shall be recognized. Any backslash-escape sequence shall be terminated by the closing quote. For example, "\01" "1" represents a single string: the octal value 1 followed by the character ' 1 '.

20565           <state>r, <state1,state2,..>r

20566           The regular expression *r* shall be matched only when the program is in one of the start conditions indicated by *state*, *state1*, and so on; see **Actions in lex** (on page 2740). (As an exception to the typographical conventions of the rest of this volume of IEEE Std 1003.1-200x, in this case <state> does not represent a metavariable, but the literal angle-bracket characters surrounding a symbol.) The start condition shall be recognized as such only at the beginning of a regular expression.

20572           *r/x*

20573           The regular expression *r* shall be matched only if it is followed by an occurrence of regular expression *x* (*x* is the instance of trailing context, further defined below). The token returned in *yytext* shall only match *r*. If the trailing portion of *r* matches the beginning of *x*, the result is unspecified. The *r* expression cannot include further trailing context or the '\$' (match-end-of-line) operator; *x* cannot include the '^' (match-beginning-of-line) operator, nor trailing context, nor the '\$' operator. That is, only one occurrence of trailing context is allowed in a *lex* regular expression, and the '^' operator only can be used at the beginning of such an expression.

20581           {*name*}

20582           When *name* is one of the substitution symbols from the *Definitions* section, the string, including the enclosing braces, shall be replaced by the *substitute* value. The *substitute* value shall be treated in the extended regular expression as if it were enclosed in parentheses. No substitution shall occur if {*name*} occurs within a bracket expression or within double-quotes.

20586           Within an ERE, a backslash character shall be considered to begin an escape sequence as specified in the table in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 5, File Format Notation ('\\', '\a', '\b', '\f', '\n', '\r', '\t', '\v'). In addition, the escape sequences in the following table shall be recognized.

20590           A literal <newline> cannot occur within an ERE; the escape sequence '\n' can be used to represent a <newline>. A <newline> shall not be matched by a period operator.

20592

Table 4-10 Escape Sequences in *lex*

20593

20594

20595

20596

20597

20598

20599

20600

20601

20602

20603

20604

20605

20606

20607

20608

20609

20610

20611

20612

20613

20614

20615

20616

20617

20618

20619

20620

20621

20622

| Escape Sequence       | Description                                                                                                                                                                                                                                                                                                                                                  | Meaning                                                                                                                                                                                                                                                                                                                                                                             |
|-----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>\digits</code>  | A backslash character followed by the longest sequence of one, two, or three octal-digit characters (01234567). If all of the digits are 0 (that is, representation of the NUL character), the behavior is undefined.                                                                                                                                        | The character whose encoding is represented by the one, two, or three-digit octal integer. If the size of a byte on the system is greater than nine bits, the valid escape sequence used to represent a byte is implementation-defined. Multi-byte characters require multiple, concatenated escape sequences of this type, including the leading ' <code>\</code> ' for each byte. |
| <code>\xdigits</code> | A backslash character followed by the longest sequence of hexadecimal-digit characters (01234567abcdefABCDEF). If all of the digits are 0 (that is, representation of the NUL character), the behavior is undefined.                                                                                                                                         | The character whose encoding is represented by the hexadecimal integer.                                                                                                                                                                                                                                                                                                             |
| <code>\c</code>       | A backslash character followed by any character not described in this table or in the table in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 5, File Format Notation (' <code>\</code> ', ' <code>\a</code> ', ' <code>\b</code> ', ' <code>\f</code> ', ' <code>\n</code> ', ' <code>\r</code> ', ' <code>\t</code> ', ' <code>\v</code> '). | The character ' <code>c</code> ', unchanged.                                                                                                                                                                                                                                                                                                                                        |

20623

20624

20625

**Note:** If a '`\x`' sequence needs to be immediately followed by a hexadecimal digit character, a sequence such as "`\x1`" "1" can be used, which represents a character containing the value 1, followed by the character '1'.

20626

20627

20628

20629

The order of precedence given to extended regular expressions for *lex* differs from that specified in the Base Definitions volume of IEEE Std 1003.1-200x, Section 9.4, Extended Regular Expressions. The order of precedence for *lex* shall be as shown in the following table, from high to low.

20630

20631

20632

20633

20634

**Note:** The escaped characters entry is not meant to imply that these are operators, but they are included in the table to show their relationships to the true operators. The start condition, trailing context, and anchoring notations have been omitted from the table because of the placement restrictions described in this section; they can only appear at the beginning or ending of an ERE.

20635

Table 4-11 ERE Precedence in *lex*

20636

20637

20638

20639

20640

20641

20642

20643

20644

20645

20646

| Extended Regular Expression              | Precedence            |
|------------------------------------------|-----------------------|
| <i>collation-related bracket symbols</i> | [ = ] [ : : ] [ . . ] |
| <i>escaped characters</i>                | \<special character>  |
| <i>bracket expression</i>                | [ ]                   |
| <i>quoting</i>                           | " . . . "             |
| <i>grouping</i>                          | ( )                   |
| <i>definition</i>                        | { name }              |
| <i>single-character RE duplication</i>   | * + ?                 |
| <i>concatenation</i>                     |                       |
| <i>interval expression</i>               | { m, n }              |
| <i>alternation</i>                       |                       |

20647

20648

20649

20650

20651

20652

20653

20654

The ERE anchoring operators '*^*' and '*\$*' do not appear in the table. With *lex* regular expressions, these operators are restricted in their use: the '*^*' operator can only be used at the beginning of an entire regular expression, and the '*\$*' operator only at the end. The operators apply to the entire regular expression. Thus, for example, the pattern "*(^abc)|(def\$)*" is undefined; it can instead be written as two separate rules, one with the regular expression "*^abc*" and one with "*def\$*", which share a common action via the special '*|*' action (see below). If the pattern were written "*^abc|def\$*", it would match either "*abc*" or "*def*" on a line by itself.

20655

20656

20657

20658

Unlike the general ERE rules, embedded anchoring is not allowed by most historical *lex* implementations. An example of embedded anchoring would be for patterns such as "*(^| )foo( |\$)*" to match "*foo*" when it exists as a complete word. This functionality can be obtained using existing *lex* features:

20659

20660

```
^foo/[ \n]      |
" foo"/[ \n]    /* Found foo as a separate word. */
```

20661

20662

20663

Note also that '*\$*' is a form of trailing context (it is equivalent to "*/\n*") and as such cannot be used with regular expressions containing another instance of the operator (see the preceding discussion of trailing context).

20664

20665

20666

20667

20668

The additional regular expressions trailing-context operator '*/'*' can be used as an ordinary character if presented within double-quotes, "*/"*"; preceded by a backslash, "*\"*"; or within a bracket expression, "*[ / ]*". The start-condition '*<*' and '*>*' operators shall be special only in a start condition at the beginning of a regular expression; elsewhere in the regular expression they shall be treated as ordinary characters.

20669

### Actions in *lex*

20670

20671

20672

20673

20674

20675

The action to be taken when an ERE is matched can be a C program fragment or the special actions described below; the program fragment can contain one or more C statements, and can also include special actions. The empty C statement '*;*' shall be a valid action; any string in the *lex.yy.c* input that matches the pattern portion of such a rule is effectively ignored or skipped. However, the absence of an action shall not be valid, and the action *lex* takes in such a condition is undefined.

20676

20677

The specification for an action, including C statements and special actions, can extend across several lines if enclosed in braces:

20678

20679

```
ERE <one or more blanks> { program statement
                          program statement }
```

20680 The default action when a string in the input to a **lex.yy.c** program is not matched by any  
 20681 expression shall be to copy the string to the output. Because the default behavior of a program  
 20682 generated by *lex* is to read the input and copy it to the output, a minimal *lex* source program that  
 20683 has just "%%" shall generate a C program that simply copies the input to the output unchanged.

20684 Four special actions shall be available:

20685 | ECHO; REJECT; BEGIN

20686 | The action ' | ' means that the action for the next rule is the action for this rule.  
 20687 Unlike the other three actions, ' | ' cannot be enclosed in braces or be semicolon-  
 20688 terminated; the application shall ensure that it is specified alone, with no other  
 20689 actions.

20690 **ECHO;** Write the contents of the string *yytext* on the output.

20691 **REJECT;** Usually only a single expression is matched by a given string in the input. **REJECT**  
 20692 means "continue to the next expression that matches the current input", and shall  
 20693 cause whatever rule was the second choice after the current rule to be executed for  
 20694 the same input. Thus, multiple rules can be matched and executed for one input  
 20695 string or overlapping input strings. For example, given the regular expressions  
 20696 "xyz" and "xy" and the input "xyz", usually only the regular expression "xyz"  
 20697 would match. The next attempted match would start after z. If the last action in the  
 20698 "xyz" rule is **REJECT**, both this rule and the "xy" rule would be executed. The  
 20699 **REJECT** action may be implemented in such a fashion that flow of control does not  
 20700 continue after it, as if it were equivalent to a **goto** to another part of *yylex()*. The  
 20701 use of **REJECT** may result in somewhat larger and slower scanners.

20702 **BEGIN** The action:

20703 BEGIN *newstate*;

20704 switches the state (start condition) to *newstate*. If the string *newstate* has not been  
 20705 declared previously as a start condition in the *Definitions* section, the results are  
 20706 unspecified. The initial state is indicated by the digit '0' or the token **INITIAL**.

20707 The functions or macros described below are accessible to user code included in the *lex* input. It  
 20708 is unspecified whether they appear in the C code output of *lex*, or are accessible only through the  
 20709 **-ll** operand to *c99* (the *lex* library).

20710 **int yylex(void)**

20711 Performs lexical analysis on the input; this is the primary function generated by the *lex*  
 20712 utility. The function shall return zero when the end of input is reached; otherwise, it shall  
 20713 return non-zero values (tokens) determined by the actions that are selected.

20714 **int yymore(void)**

20715 When called, indicates that when the next input string is recognized, it is to be appended to  
 20716 the current value of *yytext* rather than replacing it; the value in *yyleng* shall be adjusted  
 20717 accordingly.

20718 **int yylless(int n)**

20719 Retains *n* initial characters in *yytext*, NUL-terminated, and treats the remaining characters  
 20720 as if they had not been read; the value in *yyleng* shall be adjusted accordingly.

20721 **int input(void)**

20722 Returns the next character from the input, or zero on end-of-file. It shall obtain input from  
 20723 the stream pointer *yyin*, although possibly via an intermediate buffer. Thus, once scanning  
 20724 has begun, the effect of altering the value of *yyin* is undefined. The character read shall be  
 20725 removed from the input stream of the scanner without any processing by the scanner.

20726 **int unput(int c)**  
 20727 Returns the character 'c' to the input; *ytext* and *yyleng* are undefined until the next  
 20728 expression is matched. The result of using *unput()* for more characters than have been input  
 20729 is unspecified.

20730 The following functions shall appear only in the *lex* library accessible through the `-ll` operand; |  
 20731 they can therefore be redefined by a conforming application: |

20732 **int yywrap(void)**  
 20733 Called by *yylex()* at end-of-file; the default *yywrap()* shall always return 1. If the application |  
 20734 requires *yylex()* to continue processing with another source of input, then the application |  
 20735 can include a function *yywrap()*, which associates another file with the external variable |  
 20736 **FILE \* yyin** and shall return a value of zero.

20737 **int main(int argc, char \*argv[ ])**  
 20738 Calls *yylex()* to perform lexical analysis, then exits. The user code can contain *main()* to  
 20739 perform application-specific operations, calling *yylex()* as applicable.

20740 Except for *input()*, *unput()*, and *main()*, all external and static names generated by *lex* shall begin  
 20741 with the prefix **yy** or **YY**.

#### 20742 EXIT STATUS

20743 The following exit values shall be returned:

20744 0 Successful completion.

20745 >0 An error occurred.

#### 20746 CONSEQUENCES OF ERRORS

20747 Default.

#### 20748 APPLICATION USAGE

20749 Conforming applications are warned that in the *Rules* section, an *ERE* without an action is not |  
 20750 acceptable, but need not be detected as erroneous by *lex*. This may result in compilation or  
 20751 runtime errors.

20752 The purpose of *input()* is to take characters off the input stream and discard them as far as the  
 20753 lexical analysis is concerned. A common use is to discard the body of a comment once the  
 20754 beginning of a comment is recognized.

20755 The *lex* utility is not fully internationalized in its treatment of regular expressions in the *lex*  
 20756 source code or generated lexical analyzer. It would seem desirable to have the lexical analyzer  
 20757 interpret the regular expressions given in the *lex* source according to the environment specified  
 20758 when the lexical analyzer is executed, but this is not possible with the current *lex* technology.  
 20759 Furthermore, the very nature of the lexical analyzers produced by *lex* must be closely tied to the  
 20760 lexical requirements of the input language being described, which is frequently locale-specific  
 20761 anyway. (For example, writing an analyzer that is used for French text is not automatically  
 20762 useful for processing other languages.)

#### 20763 EXAMPLES

20764 The following is an example of a *lex* program that implements a rudimentary scanner for a  
 20765 Pascal-like syntax:

```
20766 %{
20767 /* Need this for the call to atof() below. */
20768 #include <math.h>
20769 /* Need this for printf(), fopen(), and stdin below. */
20770 #include <stdio.h>
20771 %}
```

```

20772     DIGIT      [0-9]
20773     ID        [a-z][a-z0-9]*
20774     %%
20775     {DIGIT}+ {
20776         printf("An integer: %s (%d)\n", yytext,
20777             atoi(yytext));
20778     }
20779     {DIGIT}+"."{DIGIT}* {
20780         printf("A float: %s (%g)\n", yytext,
20781             atof(yytext));
20782     }
20783     if|then|begin|end|procedure|function {
20784         printf("A keyword: %s\n", yytext);
20785     }
20786     {ID}     printf("An identifier: %s\n", yytext);
20787     "+"|"-"|"*"|"|" /"     printf("An operator: %s\n", yytext);
20788     "{ "[^]\n]*" /* Eat up one-line comments. */
20789     [ \t\n]+ /* Eat up white space. */
20790     . printf("Unrecognized character: %s\n", yytext);
20791     %%
20792     int main(int argc, char *argv[])
20793     {
20794         ++argv, --argc; /* Skip over program name. */
20795         if (argc > 0)
20796             yyin = fopen(argv[0], "r");
20797         else
20798             yyin = stdin;
20799         yylex();
20800     }

```

#### 20801 RATIONALE

20802 Even though the `-c` option and references to the C language are retained in this description, *lex*  
 20803 may be generalized to other languages, as was done at one time for EFL, the Extended  
 20804 FORTRAN Language. Since the *lex* input specification is essentially language-independent,  
 20805 versions of this utility could be written to produce Ada, Modula-2, or Pascal code, and there are  
 20806 known historical implementations that do so.

20807 The current description of *lex* bypasses the issue of dealing with internationalized EREs in the *lex*  
 20808 source code or generated lexical analyzer. If it follows the model used by *awk* (the source code is  
 20809 assumed to be presented in the POSIX locale, but input and output are in the locale specified by  
 20810 the environment variables), then the tables in the lexical analyzer produced by *lex* would  
 20811 interpret EREs specified in the *lex* source in terms of the environment variables specified when  
 20812 *lex* was executed. The desired effect would be to have the lexical analyzer interpret the EREs  
 20813 given in the *lex* source according to the environment specified when the lexical analyzer is  
 20814 executed, but this is not possible with the current *lex* technology.

20815 The description of octal and hexadecimal-digit escape sequences agrees with the ISO C standard  
 20816 use of escape sequences. See the RATIONALE for *ed* (on page 2537) for a discussion of bytes

20817 larger than 9 bits being represented by octal values. Hexadecimal values can represent larger  
20818 bytes and multi-byte characters directly, using as many digits as required.

20819 There is no detailed output format specification. The observed behavior of *lex* under four  
20820 different historical implementations was that none of these implementations consistently  
20821 reported the line numbers for error and warning messages. Furthermore, there was a desire that  
20822 *lex* be allowed to output additional diagnostic messages. Leaving message formats unspecified  
20823 avoids these formatting questions and problems with internationalization.

20824 Although the `%x` specifier for *exclusive* start conditions is not historical practice, it is believed to  
20825 be a minor change to historical implementations and greatly enhances the usability of *lex*  
20826 programs since it permits an application to obtain the expected functionality with fewer  
20827 statements.

20828 The `%array` and `%pointer` declarations were added as a compromise between historical systems.  
20829 The System V-based *lex* copies the matched text to a *yytext* array. The *flex* program, supported in  
20830 BSD and GNU systems, uses a pointer. In the latter case, significant performance improvements  
20831 are available for some scanners. Most historical programs should require no change in porting  
20832 from one system to another because the string being referenced is null-terminated in both cases.  
20833 (The method used by *flex* in its case is to null-terminate the token in place by remembering the  
20834 character that used to come right after the token and replacing it before continuing on to the next  
20835 scan.) Multi-file programs with external references to *yytext* outside the scanner source file  
20836 should continue to operate on their historical systems, but would require one of the new  
20837 declarations to be considered strictly portable.

20838 The description of EREs avoids unnecessary duplication of ERE details because their meanings  
20839 within a *lex* ERE are the same as that for the ERE in this volume of IEEE Std 1003.1-200x.

20840 The reason for the undefined condition associated with text beginning with a <blank> or within  
20841 "% { " and "% } " delimiter lines appearing in the *Rules* section is historical practice. Both the BSD  
20842 and System V *lex* copy the indented (or enclosed) input in the *Rules* section (except at the  
20843 beginning) to unreachable areas of the *yylex()* function (the code is written directly after a *break*  
20844 statement). In some cases, the System V *lex* generates an error message or a syntax error,  
20845 depending on the form of indented input.

20846 The intention in breaking the list of functions into those that may appear in *lex.yy.c* versus those  
20847 that only appear in *libl.a* is that only those functions in *libl.a* can be reliably redefined by a  
20848 conforming application.

20849 The descriptions of standard output and standard error are somewhat complicated because  
20850 historical *lex* implementations chose to issue diagnostic messages to standard output (unless `-t`  
20851 was given). This standard allows this behavior, but leaves an opening for the more expected  
20852 behavior of using standard error for diagnostics. Also, the System V behavior of writing the  
20853 statistics when any table sizes are given is allowed, while BSD-derived systems can avoid it. The  
20854 programmer can always precisely obtain the desired results by using either the `-t` or `-n` options.

20855 The OPERANDS section does not mention the use of `-` as a synonym for standard input; not all  
20856 historical implementations support such usage for any of the *file* operands.

20857 A description of the *translation table* was deleted from early proposals because of its relatively  
20858 low usage in historical applications.

20859 The change to the definition of the *input()* function that allows buffering of input presents the  
20860 opportunity for major performance gains in some applications.

20861 The following examples clarify the differences between *lex* regular expressions and regular  
20862 expressions appearing elsewhere in this volume of IEEE Std 1003.1-200x. For regular expressions  
20863 of the form "`r/x`", the string matching *r* is always returned; confusion may arise when the

- 20864 beginning of *x* matches the trailing portion of *r*. For example, given the regular expression  
20865 "a\*b/cc" and the input "aaabcc", *yytext* would contain the string "aaab" on this match. But  
20866 given the regular expression "x\*/xy" and the input "xxxxy", the token **xxx**, not **xx**, is returned  
20867 by some implementations because **xxx** matches "x\*".
- 20868 In the rule "ab\*/bc", the "b\*" at the end of *r* extends *r*'s match into the beginning of the  
20869 trailing context, so the result is unspecified. If this rule were "ab/bc", however, the rule  
20870 matches the text "ab" when it is followed by the text "bc". In this latter case, the matching of *r*  
20871 cannot extend into the beginning of *x*, so the result is specified.
- 20872 **FUTURE DIRECTIONS**
- 20873 None.
- 20874 **SEE ALSO**
- 20875 *c99*, *yacc*
- 20876 **CHANGE HISTORY**
- 20877 First released in Issue 2.
- 20878 **Issue 6**
- 20879 This utility is now marked as part of the C-Language Development Utilities option.
- 20880 The obsolescent **-c** option is withdrawn in this issue.
- 20881 The normative text is reworded to avoid use of the term "must" for application requirements.

20882 **NAME**20883 link — call *link()* function20884 **SYNOPSIS**20885 XSI link *file1 file2*

20886

20887 **DESCRIPTION**20888 The *link* utility shall perform the function call:20889 link(*file1*, *file2*);20890 A user may need appropriate privilege to invoke the *link* utility.20891 **OPTIONS**

20892 None.

20893 **OPERANDS**

20894 The following operands shall be supported:

20895 *file1* The pathname of an existing file.20896 *file2* The pathname of the new directory entry to be created.20897 **STDIN**

20898 Not used.

20899 **INPUT FILES**

20900 Not used.

20901 **ENVIRONMENT VARIABLES**20902 The following environment variables shall affect the execution of *link*:20903 *LANG* Provide a default value for the internationalization variables that are unset or null.  
20904 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
20905 Internationalization Variables for the precedence of internationalization variables  
20906 used to determine the values of locale categories.)20907 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
20908 internationalization variables.20909 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
20910 characters (for example, single-byte as opposed to multi-byte characters in  
20911 arguments).20912 *LC\_MESSAGES*20913 Determine the locale that should be used to affect the format and contents of  
20914 diagnostic messages written to standard error.20915 *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.20916 **ASYNCHRONOUS EVENTS**

20917 Default.

20918 **STDOUT**

20919 None.

20920 **STDERR**

20921 The standard error shall be used only for diagnostic messages. |

20922 **OUTPUT FILES**

20923 None.

20924 **EXTENDED DESCRIPTION**

20925 None.

20926 **EXIT STATUS**

20927 The following exit values shall be returned:

20928 0 Successful completion.

20929 &gt;0 An error occurred.

20930 **CONSEQUENCES OF ERRORS**

20931 Default.

20932 **APPLICATION USAGE**

20933 None.

20934 **EXAMPLES**

20935 None.

20936 **RATIONALE**

20937 None.

20938 **FUTURE DIRECTIONS**

20939 None.

20940 **SEE ALSO**20941 *In, unlink*, the System Interfaces volume of IEEE Std 1003.1-200x, *link()*20942 **CHANGE HISTORY**

20943 First released in Issue 5.

## 20944 NAME

20945 ln — link files

## 20946 SYNOPSIS

20947 ln [-fs] *source\_file target\_file*20948 ln [-fs] *source\_file ... target\_dir*

## 20949 DESCRIPTION

20950 In the first synopsis form, the *ln* utility shall create a new directory entry (link) at the destination |  
 20951 path specified by the *target\_file* operand. If the *-s* option is specified, a symbolic link shall be |  
 20952 created for the file specified by the *source\_file* operand. This first synopsis form shall be assumed |  
 20953 when the final operand does not name an existing directory; if more than two operands are  
 20954 specified and the final is not an existing directory, an error shall result.

20955 In the second synopsis form, the *ln* utility shall create a new directory entry (link), or if the *-s*  
 20956 option is specified a symbolic link, for each file specified by a *source\_file* operand, at a *destination*  
 20957 path in the existing directory named by *target\_dir*.

20958 If the last operand specifies an existing file of a type not specified by the System Interfaces  
 20959 volume of IEEE Std 1003.1-200x, the behavior is implementation-defined.

20960 The corresponding *destination* path for each *source\_file* shall be the concatenation of the target  
 20961 directory pathname, a slash character, and the last pathname component of the *source\_file*. The  
 20962 second synopsis form shall be assumed when the final operand names an existing directory.

20963 For each *source\_file*:

- 20964 1. If the *destination* path exists:
- 20965 a. If the *-f* option is not specified, *ln* shall write a diagnostic message to standard error,  
 20966 do nothing more with the current *source\_file*, and go on to any remaining *source\_files*.
  - 20967 b. Actions shall be performed equivalent to the *unlink()* function defined in the System  
 20968 Interfaces volume of IEEE Std 1003.1-200x, called using *destination* as the *path*  
 20969 argument. If this fails for any reason, *ln* shall write a diagnostic message to standard  
 20970 error, do nothing more with the current *source\_file*, and go on to any remaining  
 20971 *source\_files*.
- 20972 2. If the *-s* option is specified, *ln* shall create a symbolic link named by the *destination* path  
 20973 and containing as its pathname *source\_file*. The *ln* utility shall do nothing more with  
 20974 *source\_file* and shall go on to any remaining files.
- 20975 3. If *source\_file* is a symbolic link, actions shall be performed equivalent to the *link()* function  
 20976 using the object that *source\_file* references as the *path1* argument and the destination path  
 20977 as the *path2* argument. The *ln* utility shall do nothing more with *source\_file* and shall go on  
 20978 to any remaining files.
- 20979 4. Actions shall be performed equivalent to the *link()* function defined in the System  
 20980 Interfaces volume of IEEE Std 1003.1-200x using *source\_file* as the *path1* argument, and the  
 20981 *destination* path as the *path2* argument.

## 20982 OPTIONS

20983 The *ln* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2,  
 20984 Utility Syntax Guidelines.

20985 The following option shall be supported:

20986 *-f* Force existing *destination* pathnames to be removed to allow the link.

- 20987        -s            Create symbolic links instead of hard links.
- 20988 **OPERANDS**
- 20989        The following operands shall be supported:
- 20990        *source\_file*    A pathname of a file to be linked. If the -s option is specified, no restrictions on the  
20991                            type of file or on its existence shall be made. If the -s option is not specified,  
20992                            whether a directory can be linked is implementation-defined.
- 20993        *target\_file*    The pathname of the new directory entry to be created.
- 20994        *target\_dir*     A pathname of an existing directory in which the new directory entries are created.
- 20995 **STDIN**
- 20996        Not used.
- 20997 **INPUT FILES**
- 20998        None.
- 20999 **ENVIRONMENT VARIABLES**
- 21000        The following environment variables shall affect the execution of *ln*:
- 21001        *LANG*            Provide a default value for the internationalization variables that are unset or null.  
21002                            (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
21003                            Internationalization Variables for the precedence of internationalization variables  
21004                            used to determine the values of locale categories.)
- 21005        *LC\_ALL*         If set to a non-empty string value, override the values of all the other  
21006                            internationalization variables.
- 21007        *LC\_CTYPE*        Determine the locale for the interpretation of sequences of bytes of text data as  
21008                            characters (for example, single-byte as opposed to multi-byte characters in  
21009                            arguments).
- 21010        *LC\_MESSAGES*
- 21011                            Determine the locale that should be used to affect the format and contents of  
21012                            diagnostic messages written to standard error.
- 21013 XSI        *NLSPATH*     Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 21014 **ASYNCHRONOUS EVENTS**
- 21015        Default.
- 21016 **STDOUT**
- 21017        Not used.
- 21018 **STDERR**
- 21019        The standard error shall be used only for diagnostic messages.
- 21020 **OUTPUT FILES**
- 21021        None.
- 21022 **EXTENDED DESCRIPTION**
- 21023        None.
- 21024 **EXIT STATUS**
- 21025        The following exit values shall be returned:
- 21026        0    All the specified files were linked successfully.
- 21027        >0   An error occurred.

## 21028 CONSEQUENCES OF ERRORS

21029 Default.

## 21030 APPLICATION USAGE

21031 None.

## 21032 EXAMPLES

21033 None.

## 21034 RATIONALE

21035 Some historic versions of *ln* (including the one specified by the SVID, unlink the destination file,  
21036 if it exists, by default. If the mode does not permit writing, these versions prompt for  
21037 confirmation before attempting the unlink. In these versions the *-f* option causes *ln* not to  
21038 attempt to prompt for confirmation.

21039 This allows *ln* to succeed in creating links when the target file already exists, even if the file itself  
21040 is not writable (although the directory must be). Early proposals specified this functionality.

21041 This volume of IEEE Std 1003.1-200x does not allow the *ln* utility to unlink existing destination  
21042 paths by default for the following reasons:

21043 • The *ln* utility has historically been used to provide locking for shell applications, a usage that  
21044 is incompatible with *ln* unlinking the destination path by default. There was no  
21045 corresponding technical advantage to adding this functionality.

21046 • This functionality gave *ln* the ability to destroy the link structure of files, which changes the  
21047 historical behavior of *ln*.

21048 • This functionality is easily replicated with a combination of *rm* and *ln*.

21049 • It is not historical practice in many systems; BSD and BSD-derived systems do not support  
21050 this behavior. Unfortunately, whichever behavior is selected can cause scripts written  
21051 expecting the other behavior to fail.

21052 • It is preferable that *ln* perform in the same manner as the *link()* function, which does not  
21053 permit the target to exist already.

21054 This volume of IEEE Std 1003.1-200x retains the *-f* option to provide support for shell scripts  
21055 depending on the SVID semantics. It seems likely that shell scripts would not be written to  
21056 handle prompting by *ln* and would therefore have specified the *-f* option.

21057 The *-f* option is an undocumented feature of many historical versions of the *ln* utility, allowing  
21058 linking to directories. These versions require modification.

21059 Early proposals of this volume of IEEE Std 1003.1-200x also required an *-i* option, which  
21060 behaved like the *-i* options in *cp* and *mv*, prompting for confirmation before unlinking existing  
21061 files. This was not historical practice for the *ln* utility and has been omitted.

## 21062 FUTURE DIRECTIONS

21063 None.

## 21064 SEE ALSO

21065 *chmod*, *find*, *pax*, *rm*, the System Interfaces volume of IEEE Std 1003.1-200x, *link()*

## 21066 CHANGE HISTORY

21067 First released in Issue 2.

21068 **Issue 6**

21069

21070

The *ln* utility is updated to include symbolic link processing as defined in the IEEE P1003.2b draft standard.

21071 **NAME**

21072 locale — get locale-specific information

21073 **SYNOPSIS**

21074 locale [-a | -m]

21075 locale [-ck] *name*...21076 **DESCRIPTION**

21077 The *locale* utility shall write information about the current locale environment, or all public  
 21078 locales, to the standard output. For the purposes of this section, a *public locale* is one provided by  
 21079 the implementation that is accessible to the application.

21080 When *locale* is invoked without any arguments, it shall summarize the current locale  
 21081 environment for each locale category as determined by the settings of the environment variables  
 21082 defined in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 7, Locale.

21083 When invoked with operands, it shall write values that have been assigned to the keywords in  
 21084 the locale categories, as follows:

- 21085 • Specifying a keyword name shall select the named keyword and the category containing that  
 21086 keyword.
- 21087 • Specifying a category name shall select the named category and all keywords in that  
 21088 category.

21089 **OPTIONS**

21090 The *locale* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section  
 21091 12.2, Utility Syntax Guidelines.

21092 The following options shall be supported:

- 21093 **-a** Write information about all available public locales. The available locales shall  
 21094 include **POSIX**, representing the POSIX locale. The manner in which the  
 21095 implementation determines what other locales are available is implementation-  
 21096 defined.
- 21097 **-c** Write the names of selected locale categories; see the **STDOUT** section. The **-c**  
 21098 option increases readability when more than one category is selected (for example,  
 21099 via more than one keyword name or via a category name). It is valid both with  
 21100 and without the **-k** option.
- 21101 **-k** Write the names and values of selected keywords. The implementation may omit  
 21102 values for some keywords; see the **OPERANDS** section.
- 21103 **-m** Write names of available charmaps; see the Base Definitions volume of  
 21104 IEEE Std 1003.1-200x, Section 6.1, Portable Character Set.

21105 **OPERANDS**

21106 The following operand shall be supported:

- 21107 *name* The name of a locale category as defined in the Base Definitions volume of  
 21108 IEEE Std 1003.1-200x, Chapter 7, Locale, the name of a keyword in a locale  
 21109 category, or the reserved name **charmap**. The named category or keyword shall be  
 21110 selected for output. If a single *name* represents both a locale category name and a  
 21111 keyword name in the current locale, the results are unspecified. Otherwise, both  
 21112 category and keyword names can be specified as *name* operands, in any sequence.  
 21113 It is implementation-defined whether any keyword values are written for the  
 21114 categories *LC\_CTYPE* and *LC\_COLLATE*.

21115 **STDIN**

21116 Not used.

21117 **INPUT FILES**

21118 None.

21119 **ENVIRONMENT VARIABLES**21120 The following environment variables shall affect the execution of *locale*:

21121 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 21122 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
 21123 Internationalization Variables for the precedence of internationalization variables  
 21124 used to determine the values of locale categories.)

21125 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 21126 internationalization variables.

21127 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 21128 characters (for example, single-byte as opposed to multi-byte characters in  
 21129 arguments and input files).

21130 *LC\_MESSAGES*

21131 Determine the locale that should be used to affect the format and contents of  
 21132 diagnostic messages written to standard error.

21133 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

21134 XSI The application shall ensure that the *LANG*, *LC\_\**, and *NLSPATH* environment variables specify  
 21135 the current locale environment to be written out; they shall be used if the *-a* option is not  
 21136 specified.

21137 **ASYNCHRONOUS EVENTS**

21138 Default.

21139 **STDOUT**

21140 If *locale* is invoked without any options or operands, the names and values of the *LANG* and  
 21141 *LC\_\** environment variables described in this volume of IEEE Std 1003.1-200x shall be written to  
 21142 the standard output, one variable per line, with *LANG* first, and each line using the following  
 21143 format. Only those variables set in the environment and not overridden by *LC\_ALL* shall be  
 21144 written using this format:

21145 "%s=%s\n", &lt;variable\_name&gt;, &lt;value&gt;

21146 The names of those *LC\_\** variables associated with locale categories defined in this volume of  
 21147 IEEE Std 1003.1-200x that are not set in the environment or are overridden by *LC\_ALL* shall be  
 21148 written in the following format:

21149 "%s=\"%s\""\n", &lt;variable\_name&gt;, &lt;implied value&gt;

21150 The <implied value> shall be the name of the locale that has been selected for that category by the  
 21151 implementation, based on the values in *LANG* and *LC\_ALL*, as described in the Base Definitions  
 21152 volume of IEEE Std 1003.1-200x, Chapter 8, Environment Variables.

21153 The <value> and <implied value> shown above shall be properly quoted for possible later reentry  
 21154 to the shell. The <value> shall not be quoted using double-quotes (so that it can be distinguished  
 21155 by the user from the <implied value> case, which always requires double-quotes).

21156 The *LC\_ALL* variable shall be written last, using the first format shown above. If it is not set, it  
 21157 shall be written as:

21158 "LC\_ALL=\n"

21159 If any arguments are specified:

21160 1. If the **-a** option is specified, the names of all the public locales shall be written, each in the  
21161 following format:

21162 "%s\n", <locale name>

21163 2. If the **-c** option is specified, the names of all selected categories shall be written, each in the  
21164 following format:

21165 "%s\n", <category name>

21166 If keywords are also selected for writing (see following items), the category name output  
21167 shall precede the keyword output for that category.

21168 If the **-c** option is not specified, the names of the categories shall not be written; only the  
21169 keywords, as selected by the <name> operand, shall be written.

21170 3. If the **-k** option is specified, the names and values of selected keywords shall be written. If  
21171 a value is non-numeric, it shall be written in the following format:

21172 "%s=\"%s\"\\n", <keyword name>, <keyword value>

21173 If the keyword was **charmap**, the name of the charmap (if any) that was specified via the  
21174 *localedef* **-f** option when the locale was created shall be written, with the word **charmap** as  
21175 <keyword name>.

21176 If a value is numeric, it shall be written in one of the following formats:

21177 "%s=%d\n", <keyword name>, <keyword value>

21178 "%s=%c%o\n", <keyword name>, <escape character>, <keyword value>

21179 "%s=%cx%x\n", <keyword name>, <escape character>, <keyword value>

21180 where the <escape character> is that identified by the **escape\_char** keyword in the current  
21181 locale; see the Base Definitions volume of IEEE Std 1003.1-200x, Section 7.3, Locale  
21182 Definition.

21183 Compound keyword values (list entries) shall be separated in the output by semicolons.  
21184 When included in keyword values, the semicolon, the double-quote, the backslash, and  
21185 any control character shall be preceded (escaped) with the escape character.

21186 4. If the **-k** option is not specified, selected keyword values shall be written, each in the  
21187 following format:

21188 "%s\n", <keyword value>

21189 If the keyword was **charmap**, the name of the charmap (if any) that was specified via the  
21190 *localedef* **-f** option when the locale was created shall be written.

21191 5. If the **-m** option is specified, then a list of all available charmaps shall be written, each in  
21192 the format:

21193 "%s\n", <charmap>

21194 where <charmap> is in a format suitable for use as the option-argument to the *localedef* **-f**  
21195 option.

21196 **STDERR**

21197       The standard error shall be used only for diagnostic messages.

21198 **OUTPUT FILES**

21199       None.

21200 **EXTENDED DESCRIPTION**

21201       None.

21202 **EXIT STATUS**

21203       The following exit values shall be returned:

21204       0   All the requested information was found and output successfully.

21205       >0  An error occurred.

21206 **CONSEQUENCES OF ERRORS**

21207       Default.

21208 **APPLICATION USAGE**

21209       If the *LANG* environment variable is not set or set to an empty value, or one of the *LC\_\** environment variables is set to an unrecognized value, the actual locales assumed (if any) are implementation-defined as described in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 8, Environment Variables.

21213       Implementations are not required to write out the actual values for keywords in the categories *LC\_CTYPE* and *LC\_COLLATE*; however, they must write out the categories (allowing an application to determine, for example, which character classes are available).

21216 **EXAMPLES**

21217       In the following examples, the assumption is that locale environment variables are set as follows:

21219       LANG=locale\_x

21220       LC\_COLLATE=locale\_y

21221       The command *locale* would result in the following output:

21222       LANG=locale\_x

21223       LC\_CTYPE="locale\_x"

21224       LC\_COLLATE=locale\_y

21225       LC\_TIME="locale\_x"

21226       LC\_NUMERIC="locale\_x"

21227       LC\_MONETARY="locale\_x"

21228       LC\_MESSAGES="locale\_x"

21229       LC\_ALL=

21230       The order of presentation of the categories is not specified by this volume of IEEE Std 1003.1-200x.

21232       The command:

21233       LC\_ALL=POSIX locale -ck decimal\_point

21234       would produce:

21235       LC\_NUMERIC

21236       decimal\_point="."

21237       The following command shows an application of *locale* to determine whether a user-supplied response is affirmative:

21238

```
21239     if printf "%s\n" "$response" | grep -Eq "$(locale yesexpr)"
21240     then
21241         affirmative processing goes here
21242     else
21243         non-affirmative processing goes here
21244     fi
```

**21245 RATIONALE**

21246 The output for categories *LC\_CTYPE* and *LC\_COLLATE* has been made implementation-defined  
21247 because there is a questionable value in having a shell script receive an entire array of characters.  
21248 It is also difficult to return a logical collation description, short of returning a complete *localedef*  
21249 source.

21250 The **-m** option was included to allow applications to query for the existence of charmaps. The  
21251 output is a list of the charmaps (implementation-supplied and user-supplied, if any) on the  
21252 system.

21253 The **-c** option was included for readability when more than one category is selected (for  
21254 example, via more than one keyword name or via a category name). It is valid both with and  
21255 without the **-k** option.

21256 The **charmap** keyword, which returns the name of the charmap (if any) that was used when the  
21257 current locale was created, was included to allow applications needing the information to  
21258 retrieve it.

**21259 FUTURE DIRECTIONS**

21260 None.

**21261 SEE ALSO**

21262 *localedef*, the Base Definitions volume of IEEE Std 1003.1-200x, Section 7.3, Locale Definition

**21263 CHANGE HISTORY**

21264 First released in Issue 4.

**21265 Issue 5**

21266 FUTURE DIRECTIONS section added.

**21267 Issue 6**

21268 The normative text is reworded to avoid use of the term “must” for application requirements.

21269 **NAME**

21270 localedef — define locale environment

21271 **SYNOPSIS**21272 localedef [-c][-f *charmap*][-i *sourcefile*][-u *code\_set\_name*] *name*21273 **DESCRIPTION**

21274 The *localedef* utility shall convert source definitions for locale categories into a format usable by  
 21275 the functions and utilities whose operational behavior is determined by the setting of the locale  
 21276 environment variables defined in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter  
 21277 7, Locale. It is implementation-defined whether users have the capability to create new locales,  
 21278 in addition to those supplied by the implementation. If the symbolic constant  
 21279 XSI POSIX2\_LOCALEDEF is defined, the system supports the creation of new locales. On XSI-  
 21280 conformant systems, the symbolic constant POSIX2\_LOCALEDEF shall be defined.

21281 The utility shall read source definitions for one or more locale categories belonging to the same  
 21282 locale from the file named in the *-i* option (if specified) or from standard input.

21283 The *name* operand identifies the target locale. The utility shall support the creation of *public*, or  
 21284 generally accessible locales, as well as *private*, or restricted-access locales. Implementations may  
 21285 restrict the capability to create or modify public locales to users with the appropriate privileges.

21286 Each category source definition shall be identified by the corresponding environment variable  
 21287 name and terminated by an **END *category-name*** statement. The following categories shall be  
 21288 supported. In addition, the input may contain source for implementation-defined categories.

21289 **LC\_CTYPE** Defines character classification and case conversion.

21290 **LC\_COLLATE**

21291 Defines collation rules.

21292 **LC\_MONETARY**

21293 Defines the format and symbols used in formatting of monetary information.

21294 **LC\_NUMERIC**

21295 Defines the decimal delimiter, grouping, and grouping symbol for non-monetary  
 21296 numeric editing.

21297 **LC\_TIME** Defines the format and content of date and time information.

21298 **LC\_MESSAGES**

21299 Defines the format and values of affirmative and negative responses.

21300 **OPTIONS**

21301 The *localedef* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section  
 21302 12.2, Utility Syntax Guidelines.

21303 The following options shall be supported:

21304 **-c** Create permanent output even if warning messages have been issued.

21305 **-f *charmap*** Specify the pathname of a file containing a mapping of character symbols and  
 21306 collating element symbols to actual character encodings. The format of the  
 21307 *charmap* is described under the Base Definitions volume of IEEE Std 1003.1-200x,  
 21308 Section 6.4, Character Set Description File. The application shall ensure that this  
 21309 option is specified if symbolic names (other than collating symbols defined in a  
 21310 **collating-symbol** keyword) are used. If the *-f* option is not present, an  
 21311 implementation-defined character mapping shall be used.



- 21358 *LC\_MESSAGES*  
 21359 Determine the locale that should be used to affect the format and contents of  
 21360 diagnostic messages written to standard error.
- 21361 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 21362 **ASYNCHRONOUS EVENTS**  
 21363 Default.
- 21364 **STDOUT**  
 21365 The utility shall report all categories successfully processed, in an unspecified format.
- 21366 **STDERR**  
 21367 The standard error shall be used only for diagnostic messages.
- 21368 **OUTPUT FILES**  
 21369 The format of the created output is unspecified. If the *name* operand does not contain a slash, the  
 21370 existence of an output file for the locale is unspecified.
- 21371 **EXTENDED DESCRIPTION**  
 21372 When the *-u* option is used, the *code\_set\_name* option-argument shall be interpreted as an  
 21373 implementation-defined name of a codeset to which the ISO/IEC 10646-1:2000 standard  
 21374 position constant values shall be converted via an implementation-defined method. Both the  
 21375 ISO/IEC 10646-1:2000 standard position constant values and other formats (decimal,  
 21376 hexadecimal, or octal) shall be valid as encoding values within the *charmap* file. The codeset  
 21377 represented by the implementation-defined name can be any codeset that is supported by the  
 21378 implementation.
- 21379 When conflicts occur between the *charmap* specification of *<code\_set\_name>*, *<mb\_cur\_max>*, or  
 21380 *<mb\_cur\_min>* and the implementation-defined interpretation of these respective items for the  
 21381 codeset represented by the *-u* option-argument *code\_set\_name*, the result is unspecified.
- 21382 When conflicts occur between the *charmap* encoding values specified for symbolic names of  
 21383 characters of the portable character set and the implementation-defined assignment of character  
 21384 encoding values, the result is unspecified.
- 21385 If a non-printable character in the *charmap* has a width specified that is not *-1*, *localedef* shall  
 21386 generate a warning.
- 21387 **EXIT STATUS**  
 21388 The following exit values shall be returned:
- 21389 0 No errors occurred and the locales were successfully created.  
 21390 1 Warnings occurred and the locales were successfully created.  
 21391 2 The locale specification exceeded implementation limits or the coded character set or sets  
 21392 used were not supported by the implementation, and no locale was created.  
 21393 3 The capability to create new locales is not supported by the implementation.  
 21394 >3 Warnings or errors occurred and no output was created.
- 21395 **CONSEQUENCES OF ERRORS**  
 21396 If an error is detected, no permanent output shall be created.
- 21397 If warnings occur, permanent output shall be created if the *-c* option was specified. The  
 21398 following conditions shall cause warning messages to be issued:
- 21399 • If a symbolic name not found in the *charmap* file is used for the descriptions of the *LC\_CTYPE*  
 21400 or *LC\_COLLATE* categories (for other categories, this shall be an error condition).

- 21401           • If the number of operands to the **order** keyword exceeds the {COLL\_WEIGHTS\_MAX} limit.
  - 21402           • If optional keywords not supported by the implementation are present in the source.
  - 21403           • If a non-printable character has a width specified other than -1.
  - 21404           Other implementation-defined conditions may also cause warnings.
- 21405 **APPLICATION USAGE**
- 21406           The *charmap* definition is optional, and is contained outside the locale definition. This allows  
21407           both completely self-defined source files, and generic sources (applicable to more than one  
21408           codeset). To aid portability, all *charmap* definitions must use the same symbolic names for the  
21409           portable character set. As explained in the Base Definitions volume of IEEE Std 1003.1-200x,  
21410           Section 6.4, Character Set Description File, it is implementation-defined whether or not users or  
21411           applications can provide additional character set description files. Therefore, the -f option might  
21412           be operable only when an implementation-defined *charmap* is named.
- 21413 **EXAMPLES**
- 21414           None.
- 21415 **RATIONALE**
- 21416           The output produced by the *localedef* utility is implementation-defined. The *name* operand is  
21417           used to identify the specific locale. (As a consequence, although several categories can be  
21418           processed in one execution, only categories belonging to the same locale can be processed.)
- 21419 **FUTURE DIRECTIONS**
- 21420           None.
- 21421 **SEE ALSO**
- 21422           *locale*, the Base Definitions volume of IEEE Std 1003.1-200x, Section 7.3, Locale Definition
- 21423 **CHANGE HISTORY**
- 21424           First released in Issue 4.
- 21425 **Issue 6**
- 21426           The -u option is added, as specified in the IEEE P1003.2b draft standard.
- 21427           The normative text is reworded to avoid use of the term “must” for application requirements.

21428 **NAME**

21429           logger — log messages

21430 **SYNOPSIS**21431           logger *string* ...21432 **DESCRIPTION**

21433           The *logger* utility saves a message, in an unspecified manner and format, containing the *string* operands provided by the user. The messages are expected to be evaluated later by personnel performing system administration tasks.

21436           It is implementation-defined whether messages written in locales other than the POSIX locale are effective.

21438 **OPTIONS**

21439           None.

21440 **OPERANDS**

21441           The following operand shall be supported:

21442           *string*       One of the string arguments whose contents are concatenated together, in the order specified, separated by single <space>s.

21444 **STDIN**

21445           Not used.

21446 **INPUT FILES**

21447           None.

21448 **ENVIRONMENT VARIABLES**21449           The following environment variables shall affect the execution of *logger*:

21450           *LANG*        Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)

21454           *LC\_ALL*     If set to a non-empty string value, override the values of all the other internationalization variables.

21456           *LC\_CTYPE*   Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

21459           *LC\_MESSAGES*

21460           Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error. (This means diagnostics from *logger* to the user or application, not diagnostic messages that the user is sending to the system administrator.)

21464 XSI        *NLSPATH*   Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

21465 **ASYNCHRONOUS EVENTS**

21466           Default.

21467 **STDOUT**

21468           Not used.

21469 **STDERR**

21470 The standard error shall be used only for diagnostic messages. |

21471 **OUTPUT FILES**

21472 Unspecified.

21473 **EXTENDED DESCRIPTION**

21474 None.

21475 **EXIT STATUS**

21476 The following exit values shall be returned:

21477 0 Successful completion.

21478 >0 An error occurred.

21479 **CONSEQUENCES OF ERRORS**

21480 Default.

21481 **APPLICATION USAGE**

21482 This utility allows logging of information for later use by a system administrator or programmer  
21483 in determining why non-interactive utilities have failed. The locations of the saved messages,  
21484 their format, and retention period are all unspecified. There is no method for a conforming  
21485 application to read messages, once written. |

21486 **EXAMPLES**

21487 A batch application, running non-interactively, tries to read a configuration file and fails; it may  
21488 attempt to notify the system administrator with:

21489 logger myname: unable to read file foo. [timestamp]

21490 **RATIONALE**

21491 The standard developers believed strongly that some method of alerting administrators to errors  
21492 was necessary. The obvious example is a batch utility, running non-interactively, that is unable  
21493 to read its configuration files or that is unable to create or write its results file. However, the  
21494 standard developers did not wish to define the format or delivery mechanisms as they have  
21495 historically been (and will probably continue to be) very system-specific, as well as involving  
21496 functionality clearly outside of the scope of this volume of IEEE Std 1003.1-200x.

21497 The text with *LC\_MESSAGES* about diagnostic messages means diagnostics from *logger* to the  
21498 user or application, not diagnostic messages that the user is sending to the system administrator.

21499 Multiple *string* arguments are allowed, similar to *echo*, for ease-of-use.

21500 Like the utilities *mailx* and *lp*, *logger* is admittedly difficult to test. This was not deemed sufficient  
21501 justification to exclude these utilities from this volume of IEEE Std 1003.1-200x. It is also  
21502 arguable that they are, in fact, testable, but that the tests themselves are not portable.

21503 **FUTURE DIRECTIONS**

21504 None.

21505 **SEE ALSO**

21506 *mailx*, *write*

21507 **CHANGE HISTORY**

21508 First released in Issue 4.

21509 **NAME**

21510 logname — return the user's login name

21511 **SYNOPSIS**

21512 logname

21513 **DESCRIPTION**

21514 The *logname* utility shall write the user's login name to standard output. The login name shall be  
 21515 the string that would be returned by the *getlogin()* function defined in the System Interfaces  
 21516 volume of IEEE Std 1003.1-200x. Under the conditions where the *getlogin()* function would fail,  
 21517 the *logname* utility shall write a diagnostic message to standard error and exit with a non-zero  
 21518 exit status.

21519 **OPTIONS**

21520 None.

21521 **OPERANDS**

21522 None.

21523 **STDIN**

21524 Not used.

21525 **INPUT FILES**

21526 None.

21527 **ENVIRONMENT VARIABLES**21528 The following environment variables shall affect the execution of *logname*:

21529 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 21530 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
 21531 Internationalization Variables for the precedence of internationalization variables  
 21532 used to determine the values of locale categories.)

21533 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 21534 internationalization variables.

21535 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 21536 characters (for example, single-byte as opposed to multi-byte characters in  
 21537 arguments).

21538 *LC\_MESSAGES*

21539 Determine the locale that should be used to affect the format and contents of  
 21540 diagnostic messages written to standard error.

21541 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

21542 **ASYNCHRONOUS EVENTS**

21543 Default.

21544 **STDOUT**21545 The *logname* utility output shall be a single line consisting of the user's login name:

21546 "%s\n", &lt;login name&gt;

21547 **STDERR**

21548 The standard error shall be used only for diagnostic messages. |

21549 **OUTPUT FILES**

21550 None.

21551 **EXTENDED DESCRIPTION**

21552 None.

21553 **EXIT STATUS**

21554 The following exit values shall be returned:

21555 0 Successful completion.

21556 &gt;0 An error occurred.

21557 **CONSEQUENCES OF ERRORS**

21558 Default.

21559 **APPLICATION USAGE**21560 The *logname* utility explicitly ignores the *LOGNAME* environment variable because environment  
21561 changes could produce erroneous results.21562 **EXAMPLES**

21563 None.

21564 **RATIONALE**21565 The **passwd** file is not listed as required because the implementation may have other means of  
21566 mapping login names.21567 **FUTURE DIRECTIONS**

21568 None.

21569 **SEE ALSO**21570 *id, who*21571 **CHANGE HISTORY**

21572 First released in Issue 2.

21573 **NAME**21574 `lp` — send files to a printer21575 **SYNOPSIS**21576 `lp [-c][-d dest][-n copies][-msw][-o option]... [-t title][file...]`21577 **DESCRIPTION**

21578 The `lp` utility shall copy the input files to an output destination in an unspecified manner. The  
 21579 default output destination should be to a hardcopy device, such as a printer or microfilm  
 21580 recorder, that produces non-volatile, human-readable documents. If such a device is not  
 21581 available to the application, or if the system provides no such device, the `lp` utility shall exit with  
 21582 a non-zero exit status.

21583 The actual writing to the output device may occur some time after the `lp` utility successfully  
 21584 exits. During the portion of the writing that corresponds to each input file, the implementation  
 21585 shall guarantee exclusive access to the device.

21586 The `lp` utility shall associate a unique *request ID* with each request.

21587 Normally, a banner page is produced to separate and identify each print job. This page may be  
 21588 suppressed by implementation-defined conditions, such as an operator command or one of the  
 21589 `-o option` values.

21590 **OPTIONS**

21591 The `lp` utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2,  
 21592 Utility Syntax Guidelines.

21593 The following options shall be supported:

21594 `-c` Exit only after further access to any of the input files is no longer required. The  
 21595 application can then safely delete or modify the files without affecting the output  
 21596 operation. Normally, files are not copied, but are linked whenever possible. If the  
 21597 `-c` option is not given, then the user should be careful not to remove any of the  
 21598 files before the request has been printed in its entirety. It should also be noted that  
 21599 in the absence of the `-c` option, any changes made to the named files after the  
 21600 request is made but before it is printed may be reflected in the printed output. On  
 21601 some implementations, `-c` may be on by default.

21602 `-d dest` Specify a string that names the destination (*dest*). If *dest* is a printer, the request  
 21603 shall be printed only on that specific printer. If *dest* is a class of printers, the request  
 21604 shall be printed on the first available printer that is a member of the class. Under  
 21605 certain conditions (printer unavailability, file space limitation, and so on), requests  
 21606 for specific destinations need not be accepted. Destination names vary between  
 21607 systems.

21608 If `-d` is not specified, and neither the `LPDEST` nor `PRINTER` environment variable  
 21609 is set, an unspecified destination is used. The `-d dest` option shall take precedence  
 21610 over `LPDEST`, which in turn shall take precedence over `PRINTER`. Results are  
 21611 undefined when *dest* contains a value that is not a valid destination name.

21612 `-m` Send mail (see *mailx* (on page 2785)) after the files have been printed. By default,  
 21613 no mail is sent upon normal completion of the print request.

21614 `-n copies` Write *copies* number of copies of the files, where *copies* is a positive decimal integer.  
 21615 The methods for producing multiple copies and for arranging the multiple copies  
 21616 when multiple *file* operands are used are unspecified, except that each file shall be  
 21617 output as an integral whole, not interleaved with portions of other files.

|       |                              |                                                                                                                                                                                                                                                                                                                                                                                       |
|-------|------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 21618 | <b>-o option</b>             | Specify printer-dependent or class-dependent <i>options</i> . Several such <i>options</i> may be collected by specifying the <b>-o</b> option more than once.                                                                                                                                                                                                                         |
| 21619 |                              |                                                                                                                                                                                                                                                                                                                                                                                       |
| 21620 | <b>-s</b>                    | Suppress messages from <i>lp</i> .                                                                                                                                                                                                                                                                                                                                                    |
| 21621 | <b>-t title</b>              | Write <i>title</i> on the banner page of the output.                                                                                                                                                                                                                                                                                                                                  |
| 21622 | <b>-w</b>                    | Write a message on the user's terminal after the files have been printed. If the user is not logged in, then mail shall be sent instead.                                                                                                                                                                                                                                              |
| 21623 |                              |                                                                                                                                                                                                                                                                                                                                                                                       |
| 21624 | <b>OPERANDS</b>              |                                                                                                                                                                                                                                                                                                                                                                                       |
| 21625 |                              | The following operand shall be supported:                                                                                                                                                                                                                                                                                                                                             |
| 21626 | <b>file</b>                  | A pathname of a file to be output. If no <i>file</i> operands are specified, or if a <i>file</i> operand is '-', the standard input shall be used. If a <i>file</i> operand is used, but the <b>-c</b> option is not specified, the process performing the writing to the output device may have user and group permissions that differ from that of the process invoking <i>lp</i> . |
| 21627 |                              |                                                                                                                                                                                                                                                                                                                                                                                       |
| 21628 |                              |                                                                                                                                                                                                                                                                                                                                                                                       |
| 21629 |                              |                                                                                                                                                                                                                                                                                                                                                                                       |
| 21630 |                              |                                                                                                                                                                                                                                                                                                                                                                                       |
| 21631 | <b>STDIN</b>                 |                                                                                                                                                                                                                                                                                                                                                                                       |
| 21632 |                              | The standard input shall be used only if no <i>file</i> operands are specified, or if a <i>file</i> operand is '-'.                                                                                                                                                                                                                                                                   |
| 21633 |                              | See the INPUT FILES section.                                                                                                                                                                                                                                                                                                                                                          |
| 21634 | <b>INPUT FILES</b>           |                                                                                                                                                                                                                                                                                                                                                                                       |
| 21635 |                              | The input files shall be text files.                                                                                                                                                                                                                                                                                                                                                  |
| 21636 | <b>ENVIRONMENT VARIABLES</b> |                                                                                                                                                                                                                                                                                                                                                                                       |
| 21637 |                              | The following environment variables shall affect the execution of <i>lp</i> :                                                                                                                                                                                                                                                                                                         |
| 21638 | <b>LANG</b>                  | Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)                                                                                 |
| 21639 |                              |                                                                                                                                                                                                                                                                                                                                                                                       |
| 21640 |                              |                                                                                                                                                                                                                                                                                                                                                                                       |
| 21641 |                              |                                                                                                                                                                                                                                                                                                                                                                                       |
| 21642 | <b>LC_ALL</b>                | If set to a non-empty string value, override the values of all the other internationalization variables.                                                                                                                                                                                                                                                                              |
| 21643 |                              |                                                                                                                                                                                                                                                                                                                                                                                       |
| 21644 | <b>LC_CTYPE</b>              | Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).                                                                                                                                                                                             |
| 21645 |                              |                                                                                                                                                                                                                                                                                                                                                                                       |
| 21646 |                              |                                                                                                                                                                                                                                                                                                                                                                                       |
| 21647 | <b>LC_MESSAGES</b>           |                                                                                                                                                                                                                                                                                                                                                                                       |
| 21648 |                              | Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error and informative messages written to standard output.                                                                                                                                                                                                      |
| 21649 |                              |                                                                                                                                                                                                                                                                                                                                                                                       |
| 21650 |                              |                                                                                                                                                                                                                                                                                                                                                                                       |
| 21651 | <b>LC_TIME</b>               | Determine the format and contents of date and time strings displayed in the <i>lp</i> banner page, if any.                                                                                                                                                                                                                                                                            |
| 21652 |                              |                                                                                                                                                                                                                                                                                                                                                                                       |
| 21653 | <b>LPDEST</b>                | Determine the destination. If the <i>LPDEST</i> environment variable is not set, the <i>PRINTER</i> environment variable shall be used. The <b>-d dest</b> option takes precedence over <i>LPDEST</i> . Results are undefined when <b>-d</b> is not specified and <i>LPDEST</i> contains a value that is not a valid destination name.                                                |
| 21654 |                              |                                                                                                                                                                                                                                                                                                                                                                                       |
| 21655 |                              |                                                                                                                                                                                                                                                                                                                                                                                       |
| 21656 |                              |                                                                                                                                                                                                                                                                                                                                                                                       |
| 21657 | <b>XSI</b> <b>NLS_PATH</b>   | Determine the location of message catalogs for the processing of <i>LC_MESSAGES</i> .                                                                                                                                                                                                                                                                                                 |
| 21658 | <b>PRINTER</b>               | Determine the output device or destination. If the <i>LPDEST</i> and <i>PRINTER</i> environment variables are not set, an unspecified output device is used. The <b>-d dest</b> option and the <i>LPDEST</i> environment variable shall take precedence over <i>PRINTER</i> . Results are undefined when <b>-d</b> is not specified, <i>LPDEST</i> is unset, and                      |
| 21659 |                              |                                                                                                                                                                                                                                                                                                                                                                                       |
| 21660 |                              |                                                                                                                                                                                                                                                                                                                                                                                       |
| 21661 |                              |                                                                                                                                                                                                                                                                                                                                                                                       |

21662                    **PRINTER** contains a value that is not a valid device or destination name.

21663            **TZ**            Determine the timezone used to calculate date and time strings displayed in the *lp*  
 21664                    banner page, if any. If *TZ* is unset or null, an unspecified default timezone shall be  
 21665                    used.

21666 **ASYNCHRONOUS EVENTS**

21667            Default.

21668 **STDOUT**

21669            The *lp* utility shall write a *request ID* to the standard output, unless **-s** is specified. The format of  
 21670            the message is unspecified. The request ID can be used on systems supporting the historical  
 21671            *cancel* and *lpstat* utilities.

21672 **STDERR**

21673            The standard error shall be used only for diagnostic messages.

21674 **OUTPUT FILES**

21675            None.

21676 **EXTENDED DESCRIPTION**

21677            None.

21678 **EXIT STATUS**

21679            The following exit values shall be returned:

21680            0   All input files were processed successfully.

21681            >0 No output device was available, or an error occurred.

21682 **CONSEQUENCES OF ERRORS**

21683            Default.

21684 **APPLICATION USAGE**

21685            The *pr* and *fold* utilities can be used to achieve reasonable formatting for the implementation's  
 21686            default page size.

21687            A conforming application can use one of the *file* operands only with the **-c** option or if the file is  
 21688            publicly readable and guaranteed to be available at the time of printing. This is because the  
 21689            standard gives the implementation the freedom to queue up the request for printing at some  
 21690            later time by a different process that might not be able to access the file.

21691 **EXAMPLES**

21692            1. To print file *file*:

21693                    `lp -c file`

21694            2. To print multiple files with headers:

21695                    `pr file1 file2 | lp`

21696 **RATIONALE**

21697            The *lp* utility was designed to be a basic version of a utility that is already available in many  
 21698            historical implementations. The standard developers considered that it should be implementable  
 21699            simply as:

21700            `cat "$@" > /dev/lp`

21701            after appropriate processing of options, if that is how the implementation chose to do it and if  
 21702            exclusive access could be granted (so that two users did not write to the device simultaneously).  
 21703            Although in the future the standard developers may add other options to this utility, it should

21704 always be able to execute with no options or operands and send the standard input to an  
21705 unspecified output device.

21706 This volume of IEEE Std 1003.1-200x makes no representations concerning the format of the  
21707 printed output, except that it must be “human-readable” and “non-volatile”. Thus, writing by  
21708 default to a disk or tape drive or a display terminal would not qualify. (Such destinations are not  
21709 prohibited when `-d dest`, `LPDEST`, or `PRINTER` are used, however.)

21710 This volume of IEEE Std 1003.1-200x is worded such that a “print job” consisting of multiple  
21711 input files, possibly in multiple copies, is guaranteed to print so that any one file is not  
21712 intermixed with another, but there is no statement that all the files or copies have to print out  
21713 together.

21714 The `-c` option may imply a spooling operation, but this is not required. The utility can be  
21715 implemented to wait until the printer is ready and then wait until it is finished. Because of that,  
21716 there is no attempt to define a queuing mechanism (priorities, classes of output, and so on).

21717 On some historical systems, the request ID reported on the `STDOUT` can be used to later cancel  
21718 or find the status of a request using utilities not defined in this volume of IEEE Std 1003.1-200x.

21719 Although the historical System V `lp` and BSD `lpr` utilities have provided similar functionality,  
21720 they used different names for the environment variable specifying the destination printer. Since  
21721 the name of the utility here is `lp`, `LPDEST` (used by the System V `lp` utility) was given precedence  
21722 over `PRINTER` (used by the BSD `lpr` utility). Since environments of users frequently contain one  
21723 or the other environment variable, the `lp` utility is required to recognize both. If this was not  
21724 done, many applications would send output to unexpected output devices when users moved  
21725 from system to system.

21726 Some have commented that `lp` has far too little functionality to make it worthwhile. Requests  
21727 have proposed additional options or operands or both that added functionality. The requests  
21728 included:

- 21729 • Wording *requiring* the output to be “hardcopy”
- 21730 • A requirement for multiple printers
- 21731 • Options for supporting various page-description languages

21732 Given that a compliant system is not required to even have a printer, placing further restrictions  
21733 upon the behavior of the printer is not useful. Since hardcopy format is so application-  
21734 dependent, it is difficult, if not impossible, to select a reasonable subset of functionality that  
21735 should be required on all compliant systems.

21736 The term “unspecified” is used in this section in lieu of “implementation-defined” as most  
21737 known implementations would not be able to make definitive statements in their conformance  
21738 documents: the existence and usage of printers is very dependent on how the system  
21739 administrator configures each individual system.

21740 Since the default destination, device type, queuing mechanisms, and acceptable forms of input  
21741 are all unspecified, usage guidelines for what a conforming application can do are as follows:

- 21742 • Use the command in a pipeline, or with `-c`, so that there are no permission problems and the  
21743 files can be safely deleted or modified.
- 21744 • Limit output to text files of reasonable line lengths and printable characters and include no  
21745 device-specific formatting information, such as a page description language. The meaning of  
21746 “reasonable” in this context can only be answered as a quality-of-implementation issue, but  
21747 it should be apparent from historical usage patterns in the industry and the locale. The `pr` and  
21748 `fold` utilities can be used to achieve reasonable formatting for the default page size of the

- 21749 implementation.
- 21750 Alternatively, the application can arrange its installation in such a way that it requires the  
21751 system administrator or operator to provide the appropriate information on *lp* options and  
21752 environment variable values.
- 21753 At a minimum, having this utility in this volume of IEEE Std 1003.1-200x tells the industry that  
21754 conforming applications require a means to print output and provides at least a command name  
21755 and *LPDEST* routing mechanism that can be used for discussions between vendors, application  
21756 writers, and users. The use of “should” in the DESCRIPTION of *lp* clearly shows the intent of  
21757 the standard developers, even if they cannot mandate that all systems (such as laptops) have  
21758 printers.
- 21759 This volume of IEEE Std 1003.1-200x does not specify what the ownership of the process  
21760 performing the writing to the output device may be. If *-c* is not used, it is unspecified whether  
21761 the process performing the writing to the output device has permission to read *file* if there are  
21762 any restrictions in place on who may read *file* until after it is printed. Also, if *-c* is not used, the  
21763 results of deleting *file* before it is printed are unspecified.
- 21764 **FUTURE DIRECTIONS**
- 21765 None.
- 21766 **SEE ALSO**
- 21767 *mailx*
- 21768 **CHANGE HISTORY**
- 21769 First released in Issue 2.
- 21770 **Issue 6**
- 21771 The following new requirements on POSIX implementations derive from alignment with the  
21772 Single UNIX Specification:
- 21773 • In the DESCRIPTION, the requirement to associate a unique request ID, and the normal  
21774 generation of a banner page is added.
  - 21775 • In the OPTIONS section:
    - 21776 — The *-d dest* description is expanded, but references to *lpstat* are removed.
    - 21777 — The *-m*, *-o*, *-s*, *-t*, and *-w* options are added.
  - 21778 • In the ENVIRONMENT VARIABLES section, *LC\_TIME* may now affect the execution.
  - 21779 • The STDOUT section is added.
- 21780 The normative text is reworded to avoid use of the term “must” for application requirements.
- 21781 The *TZ* entry is added to the ENVIRONMENT VARIABLES section.

21782 **NAME**

21783        ls — list directory contents

21784 **SYNOPSIS**

21785 xsl        ls [-CFRacdilqrutl][-H | -L ][-fgmnoptsx][file...]

21786 **DESCRIPTION**

21787        For each operand that names a file of a type other than directory or symbolic link to a directory,  
 21788        *ls* shall write the name of the file as well as any requested, associated information. For each  
 21789        operand that names a file of type directory, *ls* shall write the names of files contained within the  
 21790        directory as well as any requested, associated information. If one of the **-d**, **-F**, or **-l** options are  
 21791        specified, and one of the **-H** or **-L** options are not specified, for each operand that names a file of  
 21792        type symbolic link to a directory, *ls* shall write the name of the file as well as any requested,  
 21793        associated information. If none of the **-d**, **-F**, or **-l** options are specified, or the **-H** or **-L** options  
 21794        are specified, for each operand that names a file of type symbolic link to a directory, *ls* shall write  
 21795        the names of files contained within the directory as well as any requested, associated  
 21796        information.

21797        If no operands are specified, *ls* shall write the contents of the current directory. If more than one  
 21798        operand is specified, *ls* shall write non-directory operands first; it shall sort directory and non-  
 21799        directory operands separately according to the collating sequence in the current locale.

21800        The *ls* utility shall detect infinite loops; that is, entering a previously visited directory that is an  
 21801        ancestor of the last file encountered. When it detects an infinite loop, *ls* shall write a diagnostic  
 21802        message to standard error and shall either recover its position in the hierarchy or terminate.

21803 **OPTIONS**

21804        The *ls* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2,  
 21805        Utility Syntax Guidelines.

21806        The following options shall be supported:

21807        **-C**        Write multi-text-column output with entries sorted down the columns, according  
 21808        to the collating sequence. The number of text columns and the column separator  
 21809        characters are unspecified, but should be adapted to the nature of the output  
 21810        device.

21811        **-F**        Do not follow symbolic links named as operands unless the **-H** or **-L** options are  
 21812        specified. Write a slash ( '/' ) immediately after each pathname that is a directory,  
 21813        an asterisk ( '\*' ) after each that is executable, a vertical bar ( '|' ) after each that is  
 21814        a FIFO, and an at sign ( '@' ) after each that is a symbolic link. For other file types,  
 21815        other symbols may be written.

21816        **-H**        If a symbolic link referencing a file of type directory is specified on the command  
 21817        line, *ls* shall evaluate the file information and file type to be those of the file  
 21818        referenced by the link, and not the link itself; however, *ls* shall write the name of  
 21819        the link itself and not the file referenced by the link.

21820        **-L**        Evaluate the file information and file type for all symbolic links (whether named  
 21821        on the command line or encountered in a file hierarchy) to be those of the file  
 21822        referenced by the link, and not the link itself; however, *ls* shall write the name of  
 21823        the link itself and not the file referenced by the link. When **-L** is used with **-l**, write  
 21824        the contents of symbolic links in the long format (see the STDOUT section).

21825        **-R**        Recursively list subdirectories encountered.

21826        **-a**        Write out all directory entries, including those whose names begin with a period  
 21827        ( '.' ). Entries beginning with a period shall not be written out unless explicitly

|           |           |                                                                                                                                                                                                                                                                                                                                                                               |
|-----------|-----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 21828     |           | referenced, the <b>-a</b> option is supplied, or an implementation-defined condition shall cause them to be written.                                                                                                                                                                                                                                                          |
| 21829     |           |                                                                                                                                                                                                                                                                                                                                                                               |
| 21830     | <b>-c</b> | Use time of last modification of the file status information (see <code>&lt;sys/stat.h&gt;</code> in the System Interfaces volume of IEEE Std 1003.1-200x) instead of last modification of the file itself for sorting ( <b>-t</b> ) or writing ( <b>-l</b> ).                                                                                                                |
| 21831     |           |                                                                                                                                                                                                                                                                                                                                                                               |
| 21832     |           |                                                                                                                                                                                                                                                                                                                                                                               |
| 21833     | <b>-d</b> | Do not follow symbolic links named as operands unless the <b>-H</b> or <b>-L</b> options are specified. Do not treat directories differently than other types of files. The use of <b>-d</b> with <b>-R</b> produces unspecified results.                                                                                                                                     |
| 21834     |           |                                                                                                                                                                                                                                                                                                                                                                               |
| 21835     |           |                                                                                                                                                                                                                                                                                                                                                                               |
| 21836 XSI | <b>-f</b> | Force each argument to be interpreted as a directory and list the name found in each slot. This option shall turn off <b>-l</b> , <b>-t</b> , <b>-s</b> , and <b>-r</b> , and shall turn on <b>-a</b> ; the order is the order in which entries appear in the directory.                                                                                                      |
| 21837     |           |                                                                                                                                                                                                                                                                                                                                                                               |
| 21838     |           |                                                                                                                                                                                                                                                                                                                                                                               |
| 21839 XSI | <b>-g</b> | The same as <b>-l</b> , except that the owner shall not be written.                                                                                                                                                                                                                                                                                                           |
| 21840     | <b>-i</b> | For each file, write the file's file serial number (see <code>stat()</code> in the System Interfaces volume of IEEE Std 1003.1-200x).                                                                                                                                                                                                                                         |
| 21841     |           |                                                                                                                                                                                                                                                                                                                                                                               |
| 21842     | <b>-l</b> | (The letter ell.) Do not follow symbolic links named as operands unless the <b>-H</b> or <b>-L</b> options are specified. Write out in long format (see the STDOUT section). When <b>-l</b> (ell) is specified, <b>-1</b> (one) shall be assumed.                                                                                                                             |
| 21843     |           |                                                                                                                                                                                                                                                                                                                                                                               |
| 21844     |           |                                                                                                                                                                                                                                                                                                                                                                               |
| 21845 XSI | <b>-m</b> | Stream output format; list files across the page, separated by commas.                                                                                                                                                                                                                                                                                                        |
| 21846 XSI | <b>-n</b> | The same as <b>-l</b> , except that the owner's UID and GID numbers shall be written, rather than the associated character strings.                                                                                                                                                                                                                                           |
| 21847     |           |                                                                                                                                                                                                                                                                                                                                                                               |
| 21848 XSI | <b>-o</b> | The same as <b>-l</b> , except that the group shall not be written.                                                                                                                                                                                                                                                                                                           |
| 21849 XSI | <b>-p</b> | Write a slash ( <code>' / '</code> ) after each filename if that file is a directory.                                                                                                                                                                                                                                                                                         |
| 21850     | <b>-q</b> | Force each instance of non-printable filename characters and <code>&lt;tab&gt;</code> s to be written as the question-mark ( <code>' ? '</code> ) character. Implementations may provide this option by default if the output is to a terminal device.                                                                                                                        |
| 21851     |           |                                                                                                                                                                                                                                                                                                                                                                               |
| 21852     |           |                                                                                                                                                                                                                                                                                                                                                                               |
| 21853     | <b>-r</b> | Reverse the order of the sort to get reverse collating sequence or oldest first.                                                                                                                                                                                                                                                                                              |
| 21854 XSI | <b>-s</b> | Indicate the total number of file system blocks consumed by each file displayed. The block size is implementation-defined.                                                                                                                                                                                                                                                    |
| 21855     |           |                                                                                                                                                                                                                                                                                                                                                                               |
| 21856     | <b>-t</b> | Sort with the primary key being time modified (most recently modified first) and the secondary key being filename in the collating sequence.                                                                                                                                                                                                                                  |
| 21857     |           |                                                                                                                                                                                                                                                                                                                                                                               |
| 21858     | <b>-u</b> | Use time of last access (see <code>&lt;sys/stat.h&gt;</code> in the System Interfaces volume of IEEE Std 1003.1-200x) instead of last modification of the file for sorting ( <b>-t</b> ) or writing ( <b>-l</b> ).                                                                                                                                                            |
| 21859     |           |                                                                                                                                                                                                                                                                                                                                                                               |
| 21860     |           |                                                                                                                                                                                                                                                                                                                                                                               |
| 21861 XSI | <b>-x</b> | The same as <b>-C</b> , except that the multi-text-column output is produced with entries sorted across, rather than down, the columns.                                                                                                                                                                                                                                       |
| 21862     |           |                                                                                                                                                                                                                                                                                                                                                                               |
| 21863     | <b>-1</b> | (The numeric digit one.) Force output to be one entry per line.                                                                                                                                                                                                                                                                                                               |
| 21864     |           | Specifying more than one of the options in the following mutually exclusive pairs shall not be considered an error: <b>-C</b> and <b>-l</b> (ell), <b>-m</b> and <b>-l</b> (ell), <b>-x</b> and <b>-l</b> (ell), <b>-C</b> and <b>-1</b> (one), <b>-H</b> and <b>-L</b> , <b>-c</b> and <b>-u</b> . The last option specified in each pair shall determine the output format. |
| 21865 XSI |           |                                                                                                                                                                                                                                                                                                                                                                               |
| 21866     |           |                                                                                                                                                                                                                                                                                                                                                                               |

21867 **OPERANDS**

21868 The following operand shall be supported:

21869 *file* A pathname of a file to be written. If the file specified is not found, a diagnostic  
21870 message shall be output on standard error.

21871 **STDIN**

21872 Not used.

21873 **INPUT FILES**

21874 None.

21875 **ENVIRONMENT VARIABLES**21876 The following environment variables shall affect the execution of *ls*:

21877 *COLUMNS* Determine the user's preferred column position width for writing multiple text-  
21878 column output. If this variable contains a string representing a decimal integer, the  
21879 *ls* utility shall calculate how many pathname text columns to write (see *-C*) based  
21880 on the width provided. If *COLUMNS* is not set or invalid, an implementation-  
21881 defined number of column positions shall be assumed, based on the  
21882 implementation's knowledge of the output device. The column width chosen to  
21883 write the names of files in any given directory shall be constant. Filenames shall  
21884 not be truncated to fit into the multiple text-column output.

21885 *LANG* Provide a default value for the internationalization variables that are unset or null.  
21886 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
21887 Internationalization Variables for the precedence of internationalization variables  
21888 used to determine the values of locale categories.)

21889 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
21890 internationalization variables.

21891 *LC\_COLLATE*21892 Determine the locale for character collation information in determining the  
21893 pathname collation sequence.

21894 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
21895 characters (for example, single-byte as opposed to multi-byte characters in  
21896 arguments) and which characters are defined as printable (character class **print**).

21897 *LC\_MESSAGES*21898 Determine the locale that should be used to affect the format and contents of  
21899 diagnostic messages written to standard error.21900 *LC\_TIME* Determine the format and contents for date and time strings written by *ls*.21901 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

21902 *TZ* Determine the timezone for date and time strings written by *ls*. If *TZ* is unset or  
21903 null, an unspecified default timezone shall be used.

21904 **ASYNCHRONOUS EVENTS**

21905 Default.

21906 **STDOUT**

21907 The default format shall be to list one entry per line to standard output; the exceptions are to  
21908 XSI terminals or when one of the *-C*, *-m*, or *-x* options is specified. If the output is to a terminal, the  
21909 format is implementation-defined.

21910 XSI When **-m** is specified, the format used shall be:

21911 "%s, %s, ...\n", <filename1>, <filename2>

21912 where the largest number of filenames shall be written without exceeding the length of the line.

21913 If the **-i** option is specified, the file's file serial number (see <sys/stat.h> in the System Interfaces  
21914 volume of IEEE Std 1003.1-200x) shall be written in the following format before any other output  
21915 for the corresponding entry:

21916 %u ", <file serial number>

21917 If the **-l** option is specified without **-L**, the following information shall be written:

21918 "%s %u %s %s %u %s %s\n", <file mode>, <number of links>,  
21919 <owner name>, <group name>, <number of bytes in the file>,  
21920 <date and time>, <pathname>

21921 If the file is a symbolic link, this information shall be about the link itself and the <pathname>  
21922 field shall be of the form:

21923 "%s -> %s", <pathname of link>, <contents of link>

21924 If both **-l** and **-L** are specified, the following information shall be written:

21925 "%s %u %s %s %u %s %s\n", <file mode>, <number of links>,  
21926 <owner name>, <group name>, <number of bytes in the file>,  
21927 <date and time>, <pathname of link>

21928 where all fields except <pathname of link> shall be for the file resolved from the symbolic link.

21929 XSI The **-g**, **-n**, and **-o** options use the same format as **-l**, but with omitted items and their  
21930 associated <blank>s. See the OPTIONS section.

21931 XSI In both the preceding **-l** forms, if <owner name> or <group name> cannot be determined, or if **-n** |  
21932 is given, they shall be replaced with their associated numeric values using the format %u.

21933 The <date and time> field shall contain the appropriate date and timestamp of when the file was  
21934 last modified. In the POSIX locale, the field shall be the equivalent of the output of the following  
21935 *date* command:

21936 date "+%b %e %H:%M"

21937 if the file has been modified in the last six months, or:

21938 date "+%b %e %Y"

21939 (where two <space>s are used between %e and %Y) if the file has not been modified in the last six  
21940 months or if the modification date is in the future, except that, in both cases, the final <newline>  
21941 produced by *date* shall not be included and the output shall be as if the *date* command were  
21942 executed at the time of the last modification date of the file rather than the current time. When  
21943 the *LC\_TIME* locale category is not set to the POSIX locale, a different format and order of  
21944 presentation of this field may be used.

21945 If the file is a character special or block special file, the size of the file may be replaced with  
21946 implementation-defined information associated with the device in question.

21947 If the pathname was specified as a *file* operand, it shall be written as specified.

21948 XSI The file mode written under the **-l**, **-g**, **-n**, and **-o** options shall consist of the following format:

21949 "%c%s%s%c", <entry type>, <owner permissions>,  
21950 <group permissions>, <other permissions>,

- 21951            `<optional alternate access method flag>`
- 21952            The `<optional alternate access method flag>` shall be a single `<space>` if there is no alternate or  
21953 additional access control method associated with the file; otherwise, a printable character shall  
21954 be used.
- 21955            The `<entry type>` character shall describe the type of file, as follows:
- 21956            d        Directory.
- 21957            b        Block special file.
- 21958            c        Character special file.
- 21959            l (ell) Symbolic link.
- 21960            p        FIFO.
- 21961            -        Regular file.
- 21962            Implementations may add other characters to this list to represent other implementation-defined  
21963 file types.
- 21964            The next three fields shall be three characters each:
- 21965            `<owner permissions>`  
21966            Permissions for the file owner class (see the Base Definitions volume of  
21967            IEEE Std 1003.1-200x, Section 4.4, File Access Permissions).
- 21968            `<group permissions>`  
21969            Permissions for the file group class.
- 21970            `<other permissions>`  
21971            Permissions for the file other class.
- 21972            Each field shall have three character positions:
- 21973            1. If 'r', the file is readable; if '-', the file is not readable.
- 21974            2. If 'w', the file is writable; if '-', the file is not writable.
- 21975            3. The first of the following that applies:
- 21976            S        If in `<owner permissions>`, the file is not executable and set-user-ID mode is set. If in  
21977            `<group permissions>`, the file is not executable and set-group-ID mode is set.
- 21978            s        If in `<owner permissions>`, the file is executable and set-user-ID mode is set. If in  
21979            `<group permissions>`, the file is executable and set-group-ID mode is set.
- 21980            x        The file is executable or the directory is searchable.
- 21981            -        None of the attributes of 'S', 's', or 'x' applies.
- 21982            Implementations may add other characters to this list for the third character position. Such  
21983 additions shall, however, be written in lowercase if the file is executable or searchable, and  
21984 in uppercase if it is not.
- 21985 XSI        If any of the `-l`, `-g`, `-n`, `-o`, or `-s` options is specified, each list of files within the directory shall be  
21986 preceded by a status line indicating the number of file system blocks occupied by files in the  
21987 directory in 512-byte units, rounded up to the next integral number of units, if necessary. In the  
21988 POSIX locale, the format shall be:
- 21989            "total %u\n", `<number of units in the directory>`

21990 If more than one directory, or a combination of non-directory files and directories are written,  
 21991 either as a result of specifying multiple operands, or the **-R** option, each list of files within a  
 21992 directory shall be preceded by:

21993 "\n%s:\n", <directory name>

21994 If this string is the first thing to be written, the first <newline> shall not be written. This output  
 21995 shall precede the number of units in the directory.

21996 XSI If the **-s** option is given, each file shall be written with the number of blocks used by the file.  
 21997 Along with **-C**, **-l**, **-m**, or **-x**, the number and a <space> shall precede the filename; with **-g**, **-l**,  
 21998 **-n**, or **-o**, they shall precede each line describing a file.

#### 21999 **STDERR**

22000 The standard error shall be used only for diagnostic messages.

#### 22001 **OUTPUT FILES**

22002 None.

#### 22003 **EXTENDED DESCRIPTION**

22004 None.

#### 22005 **EXIT STATUS**

22006 The following exit values shall be returned:

22007 0 Successful completion.

22008 >0 An error occurred.

#### 22009 **CONSEQUENCES OF ERRORS**

22010 Default.

#### 22011 **APPLICATION USAGE**

22012 Many implementations use the equal sign ('=') to denote sockets bound to the file system for  
 22013 the **-F** option. Similarly, many historical implementations use the 's' character to denote  
 22014 sockets as the entry type characters for the **-l** option.

22015 It is difficult for an application to use every part of the file modes field of *ls -l* in a portable  
 22016 manner. Certain file types and executable bits are not guaranteed to be exactly as shown, as  
 22017 implementations may have extensions. Applications can use this field to pass directly to a user  
 22018 printout or prompt, but actions based on its contents should generally be deferred, instead, to  
 22019 the *test* utility.

22020 The output of *ls* (with the **-l** and related options) contains information that logically could be  
 22021 used by utilities such as *chmod* and *touch* to restore files to a known state. However, this  
 22022 information is presented in a format that cannot be used directly by those utilities or be easily  
 22023 translated into a format that can be used. A character has been added to the end of the  
 22024 permissions string so that applications at least have an indication that they may be working in  
 22025 an area they do not understand instead of assuming that they can translate the permissions  
 22026 string into something that can be used. Future issues or related documents may define one or  
 22027 more specific characters to be used based on different standard additional or alternative access  
 22028 control mechanisms.

22029 As with many of the utilities that deal with filenames, the output of *ls* for multiple files or in one  
 22030 of the long listing formats must be used carefully on systems where filenames can contain  
 22031 embedded white space. Systems and system administrators should institute policies and user  
 22032 training to limit the use of such filenames.

22033 The number of disk blocks occupied by the file that it reports varies depending on underlying  
 22034 file system type, block size units reported, and the method of calculating the number of blocks.

22035 On some file system types, the number is the actual number of blocks occupied by the file  
 22036 (counting indirect blocks and ignoring holes in the file); on others it is calculated based on the  
 22037 file size (usually making an allowance for indirect blocks, but ignoring holes).

#### 22038 EXAMPLES

22039 An example of a small directory tree being fully listed with *ls -laRF a* in the POSIX locale:

```
22040 total 11
22041 drwxr-xr-x  3 hlj   prog           64 Jul  4 12:07 ./
22042 drwxrwxrwx  4 hlj   prog        3264 Jul  4 12:09 ../
22043 drwxr-xr-x  2 hlj   prog           48 Jul  4 12:07 b/
22044 -rwxr--r--  1 hlj   prog          572 Jul  4 12:07 foo*

22045 a/b:
22046 total 4
22047 drwxr-xr-x  2 hlj   prog           48 Jul  4 12:07 ./
22048 drwxr-xr-x  3 hlj   prog           64 Jul  4 12:07 ../
22049 -rw-r--r--  1 hlj   prog          700 Jul  4 12:07 bar
```

#### 22050 RATIONALE

22051 Some historical implementations of the *ls* utility show all entries in a directory except dot and  
 22052 dot-dot when a superuser invokes *ls* without specifying the *-a* option. When “normal” users  
 22053 invoke *ls* without specifying *-a*, they should not see information about any files with names  
 22054 beginning with period unless they were named as *file* operands.

22055 Implementations are expected to traverse arbitrary depths when processing the *-R* option. The  
 22056 only limitation on depth should be based on running out of physical storage for keeping track of  
 22057 untraversed directories.

22058 The *-1* (one) option is currently found in BSD and BSD-derived implementations only. It is |  
 22059 required in this volume of IEEE Std 1003.1-200x so that conforming applications might ensure |  
 22060 that output is one entry per line, even if the output is to a terminal.

22061 Generally, this volume of IEEE Std 1003.1-200x is silent about what happens when options are  
 22062 given multiple times. In the cases of *-C*, *-l*, and *-1*, however, it does specify the results of these  
 22063 overlapping options. Since *ls* is one of the most aliased commands, it is important that the  
 22064 implementation perform intuitively. For example, if the alias were:

```
22065 alias ls="ls -C"
```

22066 and the user typed *ls -1*, single-text-column output should result, not an error.

22067 The BSD *ls* provides a *-A* option (like *-a*, but dot and dot-dot are not written out). The small  
 22068 difference from *-a* did not seem important enough to require both.

22069 Implementations may make *-q* the default for terminals to prevent trojan horse attacks on |  
 22070 terminals with special escape sequences. This is not required because:

- 22071 • Some control characters may be useful on some terminals; for example, a system might write  
 22072 them as "\001" or "^A".

- 22073 • Special behavior for terminals is not relevant to application portability.

22074 An early proposal specified that the optional alternate access method flag had to be '+' if there  
 22075 was an alternate access method used on the file or <space> if there was not. This was changed to  
 22076 be <space> if there is not and a single printable character if there is. This was done for three  
 22077 reasons:

- 22078 1. There are historical implementations using characters other than '+'.

- 22079           2. There are implementations that vary this character used in that position to distinguish  
22080           between various alternate access methods in use.
- 22081           3. The standard developers did not want to preclude futures specifications that might need a  
22082           way to specify more than one alternate access method.
- 22083           Nonetheless, implementations providing a single alternate access method are encouraged to use  
22084           '+'.
- 22085           In an early proposal, the units used to specify the number of blocks occupied by files in a  
22086           directory in an *ls -l* listing was implementation-defined. This was because BSD systems have  
22087           historically used 1024-byte units and System V systems have historically used 512-byte units. It  
22088           was pointed out by BSD developers that their system has used 512-byte units in some places and  
22089           1024-byte units in other places. (System V has consistently used 512.) Therefore, this volume of  
22090           IEEE Std 1003.1-200x usually specifies 512. Future releases of BSD are expected to consistently  
22091           provide 512 bytes as a default with a way of specifying 1024-byte units where appropriate.
- 22092           The *<date and time>* field in the *-l* format is specified only for the POSIX locale. As noted, the  
22093           format can be different in other locales. No mechanism for defining this is present in this volume  
22094           of IEEE Std 1003.1-200x, as the appropriate vehicle is a messaging system; that is, the format  
22095           should be specified as a "message".
- 22096 **FUTURE DIRECTIONS**
- 22097           The *-s* uses implementation-defined units and cannot be used portably; it may be withdrawn in  
22098           a future issue.
- 22099 **SEE ALSO**
- 22100           *chmod*, *find*, the System Interfaces volume of IEEE Std 1003.1-200x, *<sys/stat.h>*
- 22101 **CHANGE HISTORY**
- 22102           First released in Issue 2.
- 22103 **Issue 5**
- 22104           Second FUTURE DIRECTION added.
- 22105 **Issue 6**
- 22106           The following new requirements on POSIX implementations derive from alignment with the  
22107           Single UNIX Specification:
- 22108
  - In the *-F* option, other symbols are allowed for other file types.
- 22109           Treatment of symbolic links is added, as defined in the IEEE P1003.2b draft standard.

## 22110 NAME

22111 m4 — macro processor (**DEVELOPMENT**)

## 22112 SYNOPSIS

22113 xsl m4 [-s][-D name[=val]]...[-U name]... file...  
22114

## 22115 DESCRIPTION

22116 The *m4* utility is a macro processor that shall read one or more text files, process them according  
22117 to their included macro statements, and write the results to standard output.

## 22118 OPTIONS

22119 The *m4* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section  
22120 12.2, Utility Syntax Guidelines, except that the order of the **-D** and **-U** options shall be  
22121 significant.

22122 The following options shall be supported:

22123 **-s** Enable line synchronization output for the *c99* preprocessor phase (that is, **#line**  
22124 directives).22125 **-D name[=val]**  
22126 Define *name* to *val* or to null if *=val* is omitted.22127 **-U name** Undefine *name*.

## 22128 OPERANDS

22129 The following operand shall be supported:

22130 *file* A pathname of a text file to be processed. If no *file* is given, or if it is '-', the  
22131 standard input shall be read.

## 22132 STDIN

22133 The standard input shall be a text file that is used if no *file* operand is given, or if it is '-'.

## 22134 INPUT FILES

22135 The input file named by the *file* operand shall be a text file.

## 22136 ENVIRONMENT VARIABLES

22137 The following environment variables shall affect the execution of *m4*:22138 *LANG* Provide a default value for the internationalization variables that are unset or null.  
22139 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
22140 Internationalization Variables for the precedence of internationalization variables  
22141 used to determine the values of locale categories.)22142 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
22143 internationalization variables.22144 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
22145 characters (for example, single-byte as opposed to multi-byte characters in  
22146 arguments and input files).22147 *LC\_MESSAGES*22148 Determine the locale that should be used to affect the format and contents of  
22149 diagnostic messages written to standard error.22150 *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

22151 **ASYNCHRONOUS EVENTS**

22152 Default.

22153 **STDOUT**22154 The standard output shall be the same as the input files, after being processed for macro  
22155 expansion.22156 **STDERR**22157 The standard error shall be used to display strings with the **errprint** macro, macro tracing |  
22158 enabled by the **traceon** macro, the defined text for macros written by the **dumpdef** macro, or for  
22159 diagnostic messages.22160 **OUTPUT FILES**

22161 None.

22162 **EXTENDED DESCRIPTION**22163 The *m4* utility shall compare each token from the input against the set of built-in and user-  
22164 defined macros. If the token matches the name of a macro, then the token shall be replaced by |  
22165 the macro's defining text, if any, and rescanned for matching macro names. Once no portion of |  
22166 the token matches the name of a macro, it shall be written to standard output. Macros may have  
22167 arguments, in which case the arguments shall be substituted into the defining text before it is  
22168 rescanned.

22169 Macro calls have the form:

22170 *name(arg1, arg2, ..., argn)*22171 Macro names shall consist of letters, digits, and underscores, where the first character is not a  
22172 digit. Tokens not of this form shall not be treated as macros.22173 The application shall ensure that the left parenthesis immediately follows the name of the  
22174 macro. If a token matching the name of a macro is not followed by a left parenthesis, it is  
22175 handled as a use of that macro without arguments.22176 If a macro name is followed by a left parenthesis, its arguments are the comma-separated tokens  
22177 between the left parenthesis and the matching right parenthesis. Unquoted <blank>s and  
22178 <newline>s preceding each argument shall be ignored. All other characters, including trailing  
22179 <blank>s and <newline>s, are retained. Commas enclosed between left and right parenthesis  
22180 characters do not delimit arguments.22181 Arguments are positionally defined and referenced. The string "\$1" in the defining text shall be  
22182 replaced by the first argument. Systems shall support at least nine arguments; only the first nine  
22183 can be referenced, using the strings "\$1" to "\$9", inclusive. The string "\$0" is replaced with  
22184 the name of the macro. The string "\$#" is replaced by the number of arguments as a string. The  
22185 string "\$\*" is replaced by a list of all of the arguments, separated by commas. The string "\$@"  
22186 is replaced by a list of all of the arguments separated by commas, and each argument is quoted  
22187 using the current left and right quoting strings.22188 If fewer arguments are supplied than are in the macro definition, the omitted arguments are  
22189 taken to be null. It is not an error if more arguments are supplied than are in the macro  
22190 definition.22191 No special meaning is given to any characters enclosed between matching left and right quoting  
22192 strings, but the quoting strings are themselves discarded. By default, the left quoting string  
22193 consists of a grave accent (``) and the right quoting string consists of an acute accent ('); |  
22194 see also the **changequote** macro.22195 Comments are written but not scanned for matching macro names; by default, the begin-  
22196 comment string consists of the number sign character and the end-comment string consists of a

|       |                                                                                                                                              |
|-------|----------------------------------------------------------------------------------------------------------------------------------------------|
| 22197 | <newline>. See also the <b>changecom</b> and <b>dnl</b> macros.                                                                              |
| 22198 | The <i>m4</i> utility shall make available the following built-in macros. They can be redefined, but                                         |
| 22199 | once this is done the original meaning is lost. Their values shall be null unless otherwise stated.                                          |
| 22200 | In the descriptions below, the term <i>defining text</i> refers to the value of the macro: the second                                        |
| 22201 | argument to the <b>define</b> macro, among other things. Except for the first argument to the <b>eval</b>                                    |
| 22202 | macro, all numeric arguments to built-in macros shall be interpreted as decimal values. The                                                  |
| 22203 | string values produced as the defining text of the <b>decr</b> , <b>divnum</b> , <b>incr</b> , <b>index</b> , <b>len</b> , and <b>sysval</b> |
| 22204 | built-in macros shall be in the form of a decimal-constant as defined in the C language.                                                     |
| 22205 | <b>changecom</b> The <b>changecom</b> macro shall set the begin-comment and end-comment strings.                                             |
| 22206 | With no arguments, the comment mechanism shall be disabled. With a single                                                                    |
| 22207 | argument, that argument shall become the begin-comment string and the                                                                        |
| 22208 | <newline> shall become the end-comment string. With two arguments, the first                                                                 |
| 22209 | argument shall become the begin-comment string and the second argument shall                                                                 |
| 22210 | become the end-comment string. Systems shall support comment strings of at least                                                             |
| 22211 | five characters.                                                                                                                             |
| 22212 | <b>changequote</b> The <b>changequote</b> macro shall set the begin-quote and end-quote strings. With no                                     |
| 22213 | arguments, the quote strings shall be set to the default values (that is, ' '). With a                                                       |
| 22214 | single argument, that argument shall become the begin-quote string and the                                                                   |
| 22215 | <newline> shall become the end-quote string. With two arguments, the first                                                                   |
| 22216 | argument shall become the begin-quote string and the second argument shall                                                                   |
| 22217 | become the end-quote string. Systems shall support quote strings of at least five                                                            |
| 22218 | characters.                                                                                                                                  |
| 22219 | <b>decr</b> The defining text of the <b>decr</b> macro shall be its first argument decremented by 1. It                                      |
| 22220 | shall be an error to specify an argument containing any non-numeric characters.                                                              |
| 22221 | <b>define</b> The second argument shall become the defining text of the macro whose name is                                                  |
| 22222 | the first argument.                                                                                                                          |
| 22223 | <b>defn</b> The defining text of the <b>defn</b> macro shall be the quoted definition (using the                                             |
| 22224 | current quoting strings) of its arguments.                                                                                                   |
| 22225 | <b>divert</b> The <i>m4</i> utility maintains nine temporary buffers, numbered 1 to 9, inclusive. When                                       |
| 22226 | the last of the input has been processed, any output that has been placed in these                                                           |
| 22227 | buffers shall be written to standard output in buffer-numerical order. The <b>divert</b>                                                     |
| 22228 | macro shall divert future output to the buffer specified by its argument. Specifying                                                         |
| 22229 | no argument or an argument of 0 shall resume the normal output process. Output                                                               |
| 22230 | diverted to a stream other than 0 to 9 shall be discarded. It shall be an error to                                                           |
| 22231 | specify an argument containing any non-numeric characters.                                                                                   |
| 22232 | <b>divnum</b> The defining text of the <b>divnum</b> macro shall be the number of the current output                                         |
| 22233 | stream as a string.                                                                                                                          |
| 22234 | <b>dnl</b> The <b>dnl</b> macro shall cause <i>m4</i> to discard all input characters up to and including                                    |
| 22235 | the next <newline>.                                                                                                                          |
| 22236 | <b>dumpdef</b> The <b>dumpdef</b> macro shall write the defined text to standard error for each of the                                       |
| 22237 | macros specified as arguments, or, if no arguments are specified, for all macros.                                                            |
| 22238 | <b>errprint</b> The <b>errprint</b> macro shall write its arguments to standard error.                                                       |
| 22239 | <b>eval</b> The <b>eval</b> macro shall evaluate its first argument as an arithmetic expression, using                                       |
| 22240 | 32-bit signed integer arithmetic. All of the C-language operators shall be                                                                   |
| 22241 | supported, except for:                                                                                                                       |

|       |                 |                                                                                               |
|-------|-----------------|-----------------------------------------------------------------------------------------------|
| 22242 | [ ]             |                                                                                               |
| 22243 | ->              |                                                                                               |
| 22244 | ++              |                                                                                               |
| 22245 | --              |                                                                                               |
| 22246 | ( <i>type</i> ) |                                                                                               |
| 22247 | unary *         |                                                                                               |
| 22248 | sizeof          |                                                                                               |
| 22249 | ,               |                                                                                               |
| 22250 | .               |                                                                                               |
| 22251 | ?:              |                                                                                               |
| 22252 | unary &         |                                                                                               |
| 22253 |                 | and all assignment operators. It shall be an error to specify any of these operators.         |
| 22254 |                 | Precedence and associativity shall be as in the ISO C standard. Systems shall                 |
| 22255 |                 | support octal and hexadecimal numbers as in the ISO C standard. The second                    |
| 22256 |                 | argument, if specified, shall set the radix for the result; the default is 10. The third      |
| 22257 |                 | argument, if specified, sets the minimum number of digits in the result. It shall be          |
| 22258 |                 | an error to specify the second or third argument containing any non-numeric                   |
| 22259 |                 | characters.                                                                                   |
| 22260 | <b>ifdef</b>    | If the first argument to the <b>ifdef</b> macro is defined, the defining text shall be the    |
| 22261 |                 | second argument. Otherwise, the defining text shall be the third argument, if                 |
| 22262 |                 | specified, or the null string, if not.                                                        |
| 22263 | <b>ifelse</b>   | The <b>ifelse</b> macro takes three or more arguments. If the first two arguments             |
| 22264 |                 | compare as equal strings (after macro expansion of both arguments), the defining              |
| 22265 |                 | text shall be the third argument. If the first two arguments do not compare as                |
| 22266 |                 | equal strings and there are three arguments, the defining text shall be null. If the          |
| 22267 |                 | first two arguments do not compare as equal strings and there are four or five                |
| 22268 |                 | arguments, the defining text shall be the fourth argument. If the first two                   |
| 22269 |                 | arguments do not compare as equal strings and there are six or more arguments,                |
| 22270 |                 | the first three arguments shall be discarded and processing shall restart with the            |
| 22271 |                 | remaining arguments.                                                                          |
| 22272 | <b>include</b>  | The defining text for the <b>include</b> macro shall be the contents of the file named by     |
| 22273 |                 | the first argument. It shall be an error if the file cannot be read.                          |
| 22274 | <b>incr</b>     | The defining text of the <b>incr</b> macro shall be its first argument incremented by 1. It   |
| 22275 |                 | shall be an error to specify an argument containing any non-numeric characters.               |
| 22276 | <b>index</b>    | The defining text of the <b>index</b> macro shall be the first character position (as a       |
| 22277 |                 | string) in the first argument where a string matching the second argument begins              |
| 22278 |                 | (zero origin), or -1 if the second argument does not occur.                                   |
| 22279 | <b>len</b>      | The defining text of the <b>len</b> macro shall be the length (as a string) of the first      |
| 22280 |                 | argument.                                                                                     |
| 22281 | <b>m4exit</b>   | Exit from the <i>m4</i> utility. If the first argument is specified, it is the exit code. The |
| 22282 |                 | default is zero. It shall be an error to specify an argument containing any non-              |
| 22283 |                 | numeric characters.                                                                           |
| 22284 | <b>m4wrap</b>   | The first argument shall be processed when EOF is reached. If the <b>m4wrap</b> macro         |
| 22285 |                 | is used multiple times, the arguments specified shall be processed in the order in            |
| 22286 |                 | which the <b>m4wrap</b> macros were processed.                                                |
| 22287 | <b>maketemp</b> | The defining text shall be the first argument, with any trailing 'X' characters               |
| 22288 |                 | replaced with the current process ID as a string.                                             |

|       |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|-------|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 22289 | <b>popdef</b>      | The <b>popdef</b> macro shall delete the current definition of its arguments, replacing that definition with the previous one. If there is no previous definition, the macro is undefined.                                                                                                                                                                                                                                                                                                                                                                                                              |
| 22290 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 22291 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 22292 | <b>pushdef</b>     | The <b>pushdef</b> macro shall be equivalent to the <b>define</b> macro with the exception that it shall preserve any current definition for future retrieval using the <b>popdef</b> macro.                                                                                                                                                                                                                                                                                                                                                                                                            |
| 22293 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 22294 | <b>shift</b>       | The defining text for the <b>shift</b> macro shall be all of its arguments except for the first one.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 22295 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 22296 | <b>sinclude</b>    | The <b>sinclude</b> macro shall be equivalent to the <b>include</b> macro, except that it shall not be an error if the file is inaccessible.                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 22297 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 22298 | <b>substr</b>      | The defining text for the <b>substr</b> macro shall be the substring of the first argument beginning at the zero-offset character position specified by the second argument. The third argument, if specified, shall be the number of characters to select; if not specified, the characters from the starting point to the end of the first argument shall become the defining text. It shall not be an error to specify a starting point beyond the end of the first argument and the defining text shall be null. It shall be an error to specify an argument containing any non-numeric characters. |
| 22299 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 22300 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 22301 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 22302 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 22303 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 22304 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 22305 | <b>syscmd</b>      | The <b>syscmd</b> macro shall interpret its first argument as a shell command line. The defining text shall be the string result of that command. No output redirection shall be performed by the <i>m4</i> utility. The exit status value from the command can be retrieved using the <b>sysval</b> macro.                                                                                                                                                                                                                                                                                             |
| 22306 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 22307 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 22308 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 22309 | <b>sysval</b>      | The defining text of the <b>sysval</b> macro shall be the exit value of the utility last invoked by the <b>syscmd</b> macro (as a string).                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 22310 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 22311 | <b>traceon</b>     | The <b>traceon</b> macro shall enable tracing for the macros specified as arguments, or, if no arguments are specified, for all macros. The trace output shall be written to standard error in an unspecified format.                                                                                                                                                                                                                                                                                                                                                                                   |
| 22312 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 22313 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 22314 | <b>traceoff</b>    | The <b>traceoff</b> macro shall disable tracing for the macros specified as arguments, or, if no arguments are specified, for all macros.                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 22315 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 22316 | <b>translit</b>    | The defining text of the <b>translit</b> macro shall be the first argument with every character that occurs in the second argument replaced with the corresponding character from the third argument.                                                                                                                                                                                                                                                                                                                                                                                                   |
| 22317 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 22318 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 22319 | <b>undefine</b>    | The <b>undefine</b> macro shall delete all definitions (including those preserved using the <b>pushdef</b> macro) of the macros named by its arguments.                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 22320 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 22321 | <b>undivert</b>    | The <b>undivert</b> macro shall cause immediate output of any text in temporary buffers named as arguments, or all temporary buffers if no arguments are specified. Buffers can be undiverted into other temporary buffers. Undiverting shall discard the contents of the temporary buffer. It shall be an error to specify an argument containing any non-numeric characters.                                                                                                                                                                                                                          |
| 22322 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 22323 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 22324 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 22325 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 22326 | <b>EXIT STATUS</b> |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 22327 |                    | The following exit values shall be returned:                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 22328 | 0                  | Successful completion.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 22329 | >0                 | An error occurred                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 22330 |                    | If the <b>m4exit</b> macro is used, the exit value can be specified by the input file.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |

22331 **CONSEQUENCES OF ERRORS**

22332 Default.

22333 **APPLICATION USAGE**22334 The **defn** macro is useful for renaming macros, especially built-ins.22335 **EXAMPLES**22336 An example of a single *m4* input file capable of generating two output files follows. The file  
22337 **file1.m4** could contain lines such as:22338 `if(VER, 1, do_something)`22339 `if(VER, 2, do_something)`

22340 The makefile for the program might include:

22341 `file1.1.c : file1.m4`22342 `m4 -D VER=1 file1.m4 > file1.1.c`22343 `...`22344 `file1.2.c : file1.m4`22345 `m4 -D VER=2 file1.m4 > file1.2.c`22346 `...`22347 The **-U** option can be used to undefine **VER**. If **file1.m4** contains:22348 `if(VER, 1, do_something)`22349 `if(VER, 2, do_something)`22350 `ifndef(VER, do_something)`

22351 then the makefile would contain:

22352 `file1.0.c : file1.m4`22353 `m4 -U VER file1.m4 > file1.0.c`22354 `...`22355 `file1.1.c : file1.m4`22356 `m4 -D VER=1 file1.m4 > file1.1.c`22357 `...`22358 `file1.2.c : file1.m4`22359 `m4 -D VER=2 file1.m4 > file1.2.c`22360 `...`22361 **RATIONALE**

22362 None.

22363 **FUTURE DIRECTIONS**

22364 None.

22365 **SEE ALSO**22366 *c99*22367 **CHANGE HISTORY**

22368 First released in Issue 2.

22369 **Issue 5**22370 The phrase “the defined text for macros written by the **dumpdef** macro” is added to the  
22371 description of **STDERR**, and the description of **dumpdef** is updated to indicate that output is  
22372 written to standard error. The description of **eval** is updated to indicate that the list of excluded  
22373 C operators excludes unary **&** and **.**. In the description of **ifdef**, the phrase “and it is not  
22374 defined to be zero” is deleted.

22375 **Issue 6**

22376 In the EXTENDED DESCRIPTION, the **eval** text is updated to include a ‘&’ character in the  
22377 excepted list.

22378 The EXTENDED DESCRIPTION of **divert** is updated to clarify that there are only nine diversion  
22379 buffers.

22380 The normative text is reworded to avoid use of the term “must” for application requirements.

22381 The Open Group Base Resolution bwg2000-006 is applied.

22382 **NAME**

22383 mailx — process messages

22384 **SYNOPSIS**22385 **Send Mode**22386 mailx [-s *subject*] *address...*22387 **Receive Mode**

22388 mailx -e

22389 mailx [-HiNn][-F][-u *user*]22390 mailx -f[-HiNn][-F][*file*]22391 **DESCRIPTION**

22392 The *mailx* utility provides a message sending and receiving facility. It has two major modes,  
 22393 selected by the options used: Send Mode and Receive Mode.

22394 On systems that do not support the User Portability Utilities option, an application using *mailx*  
 22395 shall have the ability to send messages in an unspecified manner (Send Mode). Unless the first  
 22396 character of one or more lines is tilde ('~'), all characters in the input message shall appear in  
 22397 the delivered message, but additional characters may be inserted in the message before it is  
 22398 retrieved.

22399 On systems supporting the User Portability Utilities option, mail-receiving capabilities and other  
 22400 interactive features, Receive Mode, described below, also shall be enabled.

22401 **Send Mode**

22402 Send Mode can be used by applications or users to send messages from the text in standard  
 22403 input.

22404 **Receive Mode**

22405 Receive Mode is more oriented to interactive users. Mail can be read and sent in this interactive  
 22406 mode.

22407 When reading mail, *mailx* provides commands to facilitate saving, deleting, and responding to  
 22408 messages. When sending mail, *mailx* allows editing, reviewing, and other modification of the  
 22409 message as it is entered.

22410 Incoming mail shall be stored in one or more unspecified locations for each user, collectively  
 22411 called the system *mailbox* for that user. When *mailx* is invoked in Receive Mode, the system  
 22412 mailbox shall be the default place to find new mail. As messages are read, they shall be marked  
 22413 to be moved to a secondary file for storage, unless specific action is taken. This secondary file is  
 22414 called the **mbox** and is normally located in the directory referred to by the *HOME* environment  
 22415 variable (see *MBOX* in the ENVIRONMENT VARIABLES section for a description of this file).  
 22416 Messages shall remain in this file until explicitly removed. When the **-f** option is used to read  
 22417 mail messages from secondary files, messages shall be retained in those files unless specifically  
 22418 removed. All three of these locations—system mailbox, **mbox**, and secondary file—are referred  
 22419 to in this section as simply “mailboxes”, unless more specific identification is required.

22420 **OPTIONS**

22421 The *mailx* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section  
22422 12.2, Utility Syntax Guidelines.

22423 The following options shall be supported. (Only the **-s subject** option shall be required on all  
22424 systems. The other options are required only on systems supporting the User Portability Utilities  
22425 option.)

22426 **-e** Test for the presence of mail in the system mailbox. The *mailx* utility shall write  
22427 nothing and exit with a successful return code if there is mail to read.

22428 **-f** Read messages from the file named by the *file* operand instead of the system  
22429 mailbox. (See also **folder**.) If no *file* operand is specified, read messages from the  
22430 **mbox** instead of the system mailbox.

22431 **-F** Record the message in a file named after the first recipient. The name is the login-  
22432 name portion of the address found first on the **To:** line in the mail header.  
22433 Overrides the **record** variable, if set (see **Internal Variables in mailx** (on page  
22434 2792).)

22435 **-H** Write a header summary only.

22436 **-i** Ignore interrupts. (See also **ignore**).

22437 **-n** Do not initialize from the system default start-up file. See the EXTENDED  
22438 DESCRIPTION section.

22439 **-N** Do not write an initial header summary.

22440 **-s subject** Set the **Subject** header field to *subject*. All characters in the *subject* string shall  
22441 appear in the delivered message. The results are unspecified if *subject* is longer  
22442 than {LINE\_MAX} – 10 bytes or contains a <newline>.

22443 **-u user** Read the system mailbox of the login name *user*. This shall only be successful if  
22444 the invoking user has the appropriate privileges to read the system mailbox of that  
22445 user.

22446 **OPERANDS**

22447 The following operands shall be supported:

22448 *address* Addressee of message. When **-n** is specified and no user start-up files are accessed  
22449 (see the EXTENDED DESCRIPTION section), the user or application shall ensure  
22450 this is an address to pass to the mail delivery system. Any system or user start-up  
22451 files may enable aliases (see **alias** under **Commands in mailx** (on page 2795)) that  
22452 may modify the form of *address* before it is passed to the mail delivery system.

22453 *file* A pathname of a file to be read instead of the system mailbox when **-f** is specified.  
22454 The meaning of the *file* option-argument shall be affected by the contents of the  
22455 **folder** internal variable; see **Internal Variables in mailx** (on page 2792).

22456 **STDIN**

22457 When *mailx* is invoked in Send Mode (the first synopsis line), standard input shall be the  
22458 message to be delivered to the specified addresses. When in Receive Mode, user commands shall  
22459 be accepted from *stdin*. If the User Portability Utilities option is not supported, standard input  
22460 lines beginning with a tilde ('~') character produce unspecified results.

22461 If the User Portability Utilities option is supported, then in both Send and Receive Modes,  
22462 standard input lines beginning with the escape character (usually tilde ('~')) shall affect  
22463 processing as described in **Command Escapes in mailx** (on page 2803).

22464 **INPUT FILES**

22465 When *mailx* is used as described by this volume of IEEE Std 1003.1-200x, the *file* option-  
 22466 argument (see the `-f` option) and the **mbox** shall be text files containing mail messages,  
 22467 formatted as described in the OUTPUT FILES section. The nature of the system mailbox is  
 22468 unspecified; it need not be a file.

22469 **ENVIRONMENT VARIABLES**

22470 The following environment variables shall affect the execution of *mailx*:

22471 **DEAD** Determine the pathname of the file in which to save partial messages in case of  
 22472 interrupts or delivery errors. The default shall be **dead.letter** in the directory  
 22473 named by the *HOME* variable. The behavior of *mailx* in saving partial messages is  
 22474 unspecified if the User Portability Utilities option is not supported and *DEAD* is  
 22475 not defined with the value **/dev/null**.

22476 **EDITOR** Determine the name of a utility to invoke when the **edit** (see **Commands in mailx**  
 22477 (on page 2795)) or **e** (see **Command Escapes in mailx** (on page 2803)) command is  
 22478 XSI used. The default editor is unspecified. On XSI-conformant systems it is *ed*. The  
 22479 effects of this variable are unspecified if the User Portability Utilities option is not  
 22480 supported.

22481 **HOME** Determine the pathname of the user's home directory.

22482 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 22483 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
 22484 Internationalization Variables for the precedence of internationalization variables  
 22485 used to determine the values of locale categories.)

22486 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 22487 internationalization variables.

22488 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 22489 characters (for example, single-byte as opposed to multi-byte characters in  
 22490 arguments and input files) and the handling of case-insensitive address and  
 22491 header-field comparisons.

22492 **LC\_TIME** Determine the format and contents of the date and time strings written by *mailx*.

22493 **LC\_MESSAGES**

22494 Determine the locale that should be used to affect the format and contents of  
 22495 diagnostic messages written to standard error and informative messages written to  
 22496 standard output.

22497 **LISTER** Determine a string representing the command for writing the contents of the  
 22498 **folder** directory to standard output when the **folders** command is given (see  
 22499 **folders** in **Commands in mailx** (on page 2795)). Any string acceptable as a  
 22500 *command\_string* operand to the *sh -c* command shall be valid. If this variable is null  
 22501 or not set, the output command shall be *ls*. The effects of this variable are  
 22502 unspecified if the User Portability Utilities option is not supported.

22503 **MAILRC** Determine the pathname of the start-up file. The default shall be **.mailrc** in the  
 22504 directory referred to by the *HOME* environment variable. The behavior of *mailx* is  
 22505 unspecified if the User Portability Utilities option is not supported and *MAILRC* is  
 22506 not defined with the value **/dev/null**.

22507 **MBOX** Determine a pathname of the file to save messages from the system mailbox that  
 22508 have been read. The **exit** command shall override this function, as shall saving the  
 22509 message explicitly in another file. The default shall be **mbox** in the directory

- 22510                    named by the *HOME* variable. The effects of this variable are unspecified if the  
22511                    User Portability Utilities option is not supported.
- 22512 XSI                **NLSPATH**    Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 22513                **PAGER**        Determine a string representing an output filtering or pagination command for  
22514                writing the output to the terminal. Any string acceptable as a *command\_string*  
22515                operand to the *sh -c* command shall be valid. When standard output is a terminal  
22516                device, the message output shall be piped through the command if the *mailx*  
22517                internal variable *crt* is set to a value less the number of lines in the message; see  
22518                **Internal Variables in mailx** (on page 2792). If the *PAGER* variable is null or not  
22519                set, the paginator shall be either *more* or another paginator utility documented in  
22520                the system documentation. The effects of this variable are unspecified if the User  
22521                Portability Utilities option is not supported.
- 22522                **SHELL**        Determine the name of a preferred command interpreter. The default shall be *sh*.  
22523                The effects of this variable are unspecified if the User Portability Utilities option is  
22524                not supported.
- 22525                **TERM**         Determine the name of the terminal type, to indicate in an unspecified manner, if  
22526                the internal variable *screen* is not specified, the number of lines in a screenful of  
22527                headers. If *TERM* is not set or is set to null, an unspecified default terminal type  
22528                shall be used and the value of a screenful is unspecified. The effects of this variable  
22529                are unspecified if the User Portability Utilities option is not supported.
- 22530                **TZ**            This variable may determine the timezone used to calculate date and time strings  
22531                written by *mailx*. If *TZ* is unset or null, an unspecified default timezone shall be  
22532                used.
- 22533                **VISUAL**       Determine a pathname of a utility to invoke when the **visual** command (see  
22534                **Commands in mailx** (on page 2795)) or *~v* command-escape (see **Command**  
22535                **Escapes in mailx** (on page 2803)) is used. If this variable is null or not set, the full-  
22536                screen editor shall be *vi*. The effects of this variable are unspecified if the User  
22537                Portability Utilities option is not supported.
- 22538 **ASYNCHRONOUS EVENTS**
- 22539                When *mailx* is in Send Mode and standard input is not a terminal, it shall take the standard  
22540                action for all signals.
- 22541                In Receive Mode, or in Send Mode when standard input is a terminal, if a SIGINT signal is  
22542                received:
- 22543                1. If in command mode, the current command, if there is one, shall be aborted, and a  
22544                command-mode prompt shall be written.
  - 22545                2. If in input mode:
    - 22546                a. If **ignore** is set, *mailx* shall write "@\n", discard the current input line, and continue  
22547                processing, bypassing the message-abort mechanism described in item 2b.
    - 22548                b. If the interrupt was received while sending mail, either when in Receive Mode or in  
22549                Send Mode, a message shall be written, and another subsequent interrupt, with no  
22550                other intervening characters typed, shall be required to abort the mail message. If in  
22551                Receive Mode and another interrupt is received, a command-mode prompt shall be  
22552                written. If in Send Mode and another interrupt is received, *mailx* shall terminate with  
22553                a non-zero status.
- 22554                In both cases listed in item b, if the message is not empty:

22555 i. If **save** is enabled and the file named by *DEAD* can be created, the message  
 22556 shall be written to the file named by *DEAD*. If the file exists, the message shall  
 22557 be written to replace the contents of the file.

22558 ii. If **save** is not enabled, or the file named by *DEAD* cannot be created, the  
 22559 message shall not be saved.

22560 The *mailx* utility shall take the standard action for all other signals.

#### 22561 **STDOUT**

22562 In command and input modes, all output, including prompts and messages, shall be written to  
 22563 standard output.

#### 22564 **STDERR**

22565 The standard error shall be used only for diagnostic messages.

#### 22566 **OUTPUT FILES**

22567 Various *mailx* commands and command escapes can create or add to files, including the **mbox**,  
 22568 the dead-letter file, and secondary mailboxes. When *mailx* is used as described in this volume of  
 22569 IEEE Std 1003.1-200x, these files shall be text files, formatted as follows:

22570 line beginning with **From**<space>  
 22571 [one or more *header-lines*; see **Commands in mailx** (on page 2795)]  
 22572 *empty line*  
 22573 [zero or more *body lines*  
 22574 *empty line*]  
 22575 [line beginning with **From**<space>...]

22576 where each message begins with the **From** <space> line shown, preceded by the beginning of  
 22577 the file or an empty line. (The **From** <space> line is considered to be part of the message header,  
 22578 but not one of the header-lines referred to in **Commands in mailx** (on page 2795); thus, it shall  
 22579 not be affected by the **discard**, **ignore**, or **retain** commands.) The formats of the remainder of the  
 22580 **From** <space> line and any additional header lines are unspecified, except that none shall be  
 22581 empty. The format of a message body line is also unspecified, except that no line following an  
 22582 empty line shall start with **From** <space>; *mailx* shall modify any such user-entered message  
 22583 body lines (following an empty line and beginning with **From** <space>) by adding one or more  
 22584 characters to precede the 'F'; it may add these characters to **From** <space> lines that are not  
 22585 preceded by an empty line.

22586 When a message from the system mailbox or entered by the user is not a text file, it is  
 22587 implementation-defined how such a message is stored in files written by *mailx*.

#### 22588 **EXTENDED DESCRIPTION**

22589 The entire EXTENDED DESCRIPTION section shall apply only to implementations supporting  
 22590 the User Portability Utilities option.

22591 The *mailx* utility cannot guarantee support for all character encodings in all circumstances. For  
 22592 example, inter-system mail may be restricted to 7-bit data by the underlying network, 8-bit data  
 22593 need not be portable to non-internationalized systems, and so on. Under these circumstances, it  
 22594 is recommended that only characters defined in the ISO/IEC 646:1991 standard International  
 22595 Reference Version (equivalent to ASCII) 7-bit range of characters be used.

22596 When *mailx* is invoked using one of the Receive Mode synopsis forms, it shall write a page of  
 22597 header-summary lines (if **-N** was not specified and there are messages, see below), followed by  
 22598 a prompt indicating that *mailx* can accept regular commands (see **Commands in mailx** (on page  
 22599 2795)); this is termed *command mode*. The page of header-summary lines shall contain the first  
 22600 new message if there are new messages, or the first unread message if there are unread  
 22601 messages, or the first message. When *mailx* is invoked using the Send Mode synopsis and

22602 standard input is a terminal, if no subject is specified on the command line and the **asksub**  
 22603 variable is set, a prompt for the subject shall be written. At this point, *mailx* shall be in input  
 22604 mode. This input mode shall also be entered when using one of the Receive Mode synopsis  
 22605 forms and a reply or new message is composed using the **reply**, **Reply**, **followup**, **Followup**, or  
 22606 **mail** commands and standard input is a terminal. When the message is typed and the end of  
 22607 message is encountered, the message shall be passed to the mail delivery software. Commands  
 22608 can be entered by beginning a line with the escape character (by default, tilde ('~')) followed by  
 22609 a single command letter and optional arguments. See **Commands in mailx** (on page 2795) for a  
 22610 summary of these commands. It is unspecified what effect these commands will have if  
 22611 standard input is not a terminal when a message is entered using either the Send Mode synopsis,  
 22612 or the Read Mode commands **reply**, **Reply**, **followup**, **Followup**, or **mail**.

22613 **Note:** For notational convenience, this section uses the default escape character, tilde, in all references  
 22614 and examples.

22615 At any time, the behavior of *mailx* shall be governed by a set of environmental and internal  
 22616 variables. These are flags and valued parameters that can be set and cleared via the *mailx set*  
 22617 and *unset* commands.

22618 Regular commands are of the form:

22619 `[command] [msglist] [argument ...]`

22620 If no *command* is specified in command mode, **next** shall be assumed. In input mode, commands  
 22621 shall be recognized by the escape character, and lines not treated as commands shall be taken as  
 22622 input for the message.

22623 In command mode, each message shall be assigned a sequential number, starting with 1.

22624 All messages have a state that shall affect how they are displayed in the header summary and  
 22625 how they are retained or deleted upon termination of *mailx*. There is at any time the notion of a  
 22626 *current* message, which shall be marked by a '>' at the beginning of a line in the header  
 22627 summary. When *mailx* is invoked using one of the Receive Mode synopsis forms, the current  
 22628 message shall be the first new message, if there is a new message, or the first unread message if  
 22629 there is an unread message, or the first message if there are any messages, or unspecified if there  
 22630 are no messages in the mailbox. Each command that takes an optional list of messages (*msglist*)  
 22631 or an optional single message (*message*) on which to operate shall leave the current message set  
 22632 to the highest-numbered message of the messages specified, unless the command deletes  
 22633 messages, in which case the current message shall be set to the first undeleted message (that is, a  
 22634 message not in the deleted state) after the highest-numbered message deleted by the command,  
 22635 if one exists, or the first undeleted message before the highest-numbered message deleted by the  
 22636 command, if one exists, or to an unspecified value if there are no remaining undeleted messages.  
 22637 All messages shall be in one of the following states:

22638 *new* The message is present in the system mailbox and has not been viewed by the user  
 22639 or moved to any other state. Messages in state *new* when *mailx* quits shall be  
 22640 retained in the system mailbox.

22641 *unread* The message has been present in the system mailbox for more than one invocation  
 22642 of *mailx* and has not been viewed by the user or moved to any other state.  
 22643 Messages in state *unread* when *mailx* quits shall be retained in the system mailbox.

22644 *read* The message has been processed by one of the following commands: **~f**, **~m**, **~F**, **~M**,  
 22645 **copy**, **mbox**, **next**, **pipe**, **print**, **Print**, **top**, **type**, **Type**, **undelete**. The **delete**, **dp**, and  
 22646 **dt** commands may also cause the next message to be marked as *read*, depending on  
 22647 the value of the **autoprint** variable. Messages that are in the system mailbox and in  
 22648 state *read* when *mailx* quits shall be saved in the **mbox**, unless the internal variable  
 22649 **hold** was set. Messages that are in the **mbox** or in a secondary mailbox and in state

- 22650 *read* when *mailx* quits shall be retained in their current location.
- 22651 *deleted* The message has been processed by one of the following commands: **delete**, **dp**,  
22652 **dt**. Messages in state *deleted* when *mailx* quits shall be deleted. Deleted messages  
22653 shall be ignored until *mailx* quits or changes mailboxes or they are specified to the  
22654 undelete command; for example, the message specification */string* shall only  
22655 search the subject lines of messages that have not yet been deleted, unless the  
22656 command operating on the list of messages is **undelete**. No deleted message or  
22657 deleted message header shall be displayed by any *mailx* command other than  
22658 **undelete**.
- 22659 *preserved* The message has been processed by a **preserve** command. When *mailx* quits, the  
22660 message shall be retained in its current location.
- 22661 *saved* The message has been processed by one of the following commands: **save** or  
22662 **write**. If the current mailbox is the system mailbox, and the internal variable  
22663 **keepsave** is set, messages in the state *saved* shall be saved to the file designated by  
22664 the *MBOX* variable (see the ENVIRONMENT VARIABLES section). If the current  
22665 mailbox is the system mailbox, messages in the state *saved* shall be deleted from  
22666 the current mailbox, when the **quit** or **file** command is used to exit the current  
22667 mailbox.
- 22668 The header-summary line for each message shall indicate the state of the message.
- 22669 Many commands take an optional list of messages (*msglist*) on which to operate, which defaults  
22670 to the current message. A *msglist* is a list of message specifications separated by <blank>s, which  
22671 can include:
- 22672 *n* Message number *n*.
- 22673 **+** The next undeleted message, or the next deleted message for the **undelete** command.
- 22674 **-** The next previous undeleted message, or the next previous deleted message for the  
22675 **undelete** command.
- 22676 **.** The current message.
- 22677 **^** The first undeleted message, or the first deleted message for the **undelete** command.
- 22678 **\$** The last message.
- 22679 **\*** All messages.
- 22680 *n-m* An inclusive range of message numbers.
- 22681 *address* All messages from *address*; any address as shown in a header summary shall be  
22682 matchable in this form.
- 22683 */string* All messages with *string* in the subject line (case ignored).
- 22684 **:c** All messages of type *c*, where *c* shall be one of:
- 22685 **d** Deleted messages.
- 22686 **n** New messages.
- 22687 **o** Old messages (any not in state *read* or *new*).
- 22688 **r** Read messages.
- 22689 **u** Unread messages.

22690 Other commands take an optional message (*message*) on which to operate, which defaults to the  
 22691 current message. All of the forms allowed for *msglist* are also allowed for *message*, but if more  
 22692 than one message is specified, only the first shall be operated on.

22693 Other arguments are usually arbitrary strings whose usage depends on the command involved.

### 22694 **Start-Up in mailx**

22695 At start-up time, *mailx* shall take the following steps in sequence:

- 22696 1. Establish all variables at their stated default values.
- 22697 2. Process command line options, overriding corresponding default values.
- 22698 3. Import any of the *DEAD*, *EDITOR*, *MBOX*, *LISTER*, *PAGER*, *SHELL*, or *VISUAL* variables  
 22699 that are present in the environment, overriding the corresponding default values.
- 22700 4. Read *mailx* commands from an unspecified system start-up file, unless the **-n** option is  
 22701 given, to initialize any internal *mailx* variables and aliases.
- 22702 5. Process the start-up file of *mailx* commands named in the user *MAILRC* variable.

22703 Most regular *mailx* commands are valid inside start-up files, the most common use being to set  
 22704 up initial display options and alias lists. The following commands shall be invalid in the start-up  
 22705 file: **!**, **edit**, **hold**, **mail**, **preserve**, **reply**, **Reply**, **shell**, **visual**, **Copy**, **followup**, and **Followup**.  
 22706 Any errors in the start-up file shall either cause *mailx* to terminate with a diagnostic message and  
 22707 a non-zero status or to continue after writing a diagnostic message, ignoring the remainder of  
 22708 the lines in the start-up file.

22709 A blank line in a start-up file shall be ignored.

### 22710 **Internal Variables in mailx**

22711 The following variables are internal *mailx* variables. Each internal variable can be set via the  
 22712 *mailx set* command at any time. The **unset** and **set no name** commands can be used to erase  
 22713 variables.

22714 In the following list, variables shown as:

22715 variable

22716 represent Boolean values. Variables shown as:

22717 variable=*value*

22718 shall be assigned string or numeric values. For string values, the rules in **Commands in mailx**  
 22719 (on page 2795) concerning filenames and quoting shall also apply. |

22720 The defaults specified here may be changed by the implementation-defined system start-up file  
 22721 unless the user specifies the **-n** option.

22722 **allnet** All network names whose login name components match shall be treated as |  
 22723 identical. This shall cause the *msglist* message specifications to behave similarly.  
 22724 The default shall be **noallnet**. See also the **alternates** command and the **metoo**  
 22725 variable.

22726 **append** Append messages to the end of the **mbox** file upon termination instead of placing  
 22727 them at the beginning. The default shall be **noappend**. This variable shall not  
 22728 affect the **save** command when saving to the **mbox**.

22729 **ask**, **asksub**

22730 Prompt for a subject line on outgoing mail if one is not specified on the command

|           |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|-----------|-------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 22731     |                         | line with the <code>-s</code> option. The <b>ask</b> and <b>asksub</b> forms are synonyms; the system shall refer to <b>asksub</b> and <b>noasksub</b> in its messages, but shall accept <b>ask</b> and <b>noask</b> as user input to mean <b>asksub</b> and <b>noasksub</b> . It shall not be possible to set both <b>ask</b> and <b>noasksub</b> , or <b>noask</b> and <b>asksub</b> . The default shall be <b>asksub</b> , but no prompting shall be done if standard input is not a terminal.                                                                                                                                                                                                                                                      |
| 22732     |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 22733     |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 22734     |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 22735     |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 22736     | <b>askbcc</b>           | Prompt for the blind copy list. The default shall be <b>noaskbcc</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 22737     | <b>askcc</b>            | Prompt for the copy list. The default shall be <b>noaskcc</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 22738     | <b>autoprint</b>        | Enable automatic writing of messages after <b>delete</b> and <b>undelete</b> commands. The default shall be <b>noautoprint</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 22739     |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 22740     | <b>bang</b>             | Enable the special-case treatment of exclamation marks ( <code>' ! '</code> ) in escape command lines; see the <b>escape</b> command and <b>Command Escapes in mailx</b> (on page 2803). The default shall be <b>nobang</b> , disabling the expansion of <code>' ! '</code> in the <i>command</i> argument to the <code>!</code> command and the <code>~&lt;!command</code> escape.                                                                                                                                                                                                                                                                                                                                                                    |
| 22741     |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 22742     |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 22743     |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 22744     | <b>cmd=command</b>      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 22745     |                         | Set the default command to be invoked by the <b>pipe</b> command. The default shall be <b>nocmd</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 22746     |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 22747     | <b>crt=number</b>       | Pipe messages having more than <i>number</i> lines through the command specified by the value of the <i>PAGER</i> variable. The default shall be <b>nocrt</b> . If it is set to null, the value used is implementation-defined.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 22748     |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 22749     |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 22750 XSI | <b>debug</b>            | Enable verbose diagnostics for debugging. Messages are not delivered. The default shall be <b>nodebug</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 22751     |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 22752     | <b>dot</b>              | When <b>dot</b> is set, a period on a line by itself during message input from a terminal shall also signify end-of-file (in addition to normal end-of-file). The default shall be <b>nodot</b> . If <b>ignoreeof</b> is set (see below), a setting of <b>nodot</b> shall be ignored and the period is the only method to terminate input mode.                                                                                                                                                                                                                                                                                                                                                                                                        |
| 22753     |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 22754     |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 22755     |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 22756     | <b>escape=c</b>         | Set the command escape character to be the character <code>' c '</code> . By default, the command escape character shall be tilde. If <b>escape</b> is unset, tilde shall be used; if it is set to null, command escaping shall be disabled.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 22757     |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 22758     |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 22759     | <b>flipr</b>            | Reverse the meanings of the <b>R</b> and <b>r</b> commands. The default shall be <b>noflipr</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 22760     | <b>folder=directory</b> |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 22761     |                         | The default directory for saving mail files. User-specified filenames beginning with a plus sign ( <code>' + '</code> ) shall be expanded by preceding the filename with this directory name to obtain the real pathname. If <i>directory</i> does not start with a slash ( <code>' / '</code> ), the contents of <i>HOME</i> shall be prefixed to it. The default shall be <b>nofolder</b> . If <b>folder</b> is unset or set to null, user-specified filenames beginning with <code>' + '</code> shall refer to files in the current directory that begin with the literal <code>' + '</code> character. See also <b>outfolder</b> below. The <b>folder</b> value need not affect the processing of the files named in <i>MBOX</i> and <i>DEAD</i> . |
| 22762     |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 22763     |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 22764     |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 22765     |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 22766     |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 22767     |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 22768     |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 22769     | <b>header</b>           | Enable writing of the header summary when entering <i>mailx</i> in Receive Mode. The default shall be <b>header</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 22770     |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 22771     | <b>hold</b>             | Preserve all messages that are read in the system mailbox instead of putting them in the <b>mbox</b> save file. The default shall be <b>nohold</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 22772     |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 22773     | <b>ignore</b>           | Ignore interrupts while entering messages. The default shall be <b>noignore</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 22774     | <b>ignoreeof</b>        | Ignore normal end-of-file during message input. Input can be terminated only by entering a period ( <code>' . '</code> ) on a line by itself or by the <code>~</code> command escape. The default                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 22775     |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |

|           |                            |                                                                                                           |
|-----------|----------------------------|-----------------------------------------------------------------------------------------------------------|
| 22776     |                            | shall be <b>noignoreeof</b> . See also <b>dot</b> above.                                                  |
| 22777     | <b>indentprefix=string</b> |                                                                                                           |
| 22778     |                            | A string that shall be added as a prefix to each line that is inserted into the message                   |
| 22779     |                            | by the <b>~m</b> command escape. This variable shall default to one <tab>.                                |
| 22780     | <b>keep</b>                | When a system mailbox, secondary mailbox, or <b>mbox</b> is empty, truncate it to zero                    |
| 22781     |                            | length instead of removing it. The default shall be <b>nokeep</b> .                                       |
| 22782     | <b>keepsave</b>            | Keep the messages that have been saved from the system mailbox into other files                           |
| 22783     |                            | in the file designated by the variable <i>MBOX</i> , instead of deleting them. The default                |
| 22784     |                            | shall be <b>nokeepsave</b> .                                                                              |
| 22785     | <b>metoo</b>               | Suppress the deletion of the login name of the user from the recipient list when                          |
| 22786     |                            | replying to a message or sending to a group. The default shall be <b>nometoo</b> .                        |
| 22787 XSI | <b>onehop</b>              | When responding to a message that was originally sent to several recipients, the                          |
| 22788     |                            | other recipient addresses are normally forced to be relative to the originating                           |
| 22789     |                            | author's machine for the response. This flag disables alteration of the recipients'                       |
| 22790     |                            | addresses, improving efficiency in a network where all machines can send directly                         |
| 22791     |                            | to all other machines (that is, one hop away). The default shall be <b>noonehop</b> .                     |
| 22792     | <b>outfolder</b>           | Cause the files used to record outgoing messages to be located in the directory                           |
| 22793     |                            | specified by the <b>folder</b> variable unless the pathname is absolute. The default shall                |
| 22794     |                            | be <b>nooutfolder</b> . See the <b>record</b> variable.                                                   |
| 22795     | <b>page</b>                | Insert a <form-feed> after each message sent through the pipe created by the <b>pipe</b>                  |
| 22796     |                            | command. The default shall be <b>nopage</b> .                                                             |
| 22797     | <b>prompt=string</b>       |                                                                                                           |
| 22798     |                            | Set the command-mode prompt to <i>string</i> . If <i>string</i> is null or if <b>noprompt</b> is set, no  |
| 22799     |                            | prompting shall occur. The default shall be to prompt with the string " ? ".                              |
| 22800     | <b>quiet</b>               | Refrain from writing the opening message and version when entering <i>mailx</i> . The                     |
| 22801     |                            | default shall be <b>noquiet</b> .                                                                         |
| 22802     | <b>record=file</b>         | Record all outgoing mail in the file with the pathname <i>file</i> . The default shall be                 |
| 22803     |                            | <b>norecord</b> . See also <b>outfolder</b> above.                                                        |
| 22804     | <b>save</b>                | Enable saving of messages in the dead-letter file on interrupt or delivery error. See                     |
| 22805     |                            | the variable <i>DEAD</i> for the location of the dead-letter file. The default shall be <b>save</b> .     |
| 22806     | <b>screen=number</b>       |                                                                                                           |
| 22807     |                            | Set the number of lines in a screenful of headers for the <b>headers</b> and <b>z</b> commands.           |
| 22808     |                            | If <b>screen</b> is not specified, a value based on the terminal type identified by the                   |
| 22809     |                            | <i>TERM</i> environment variable, the window size, the baud rate, or some combination                     |
| 22810     |                            | of these shall be used.                                                                                   |
| 22811     | <b>sendwait</b>            | Wait for the background mailer to finish before returning. The default shall be                           |
| 22812     |                            | <b>nosendwait</b> .                                                                                       |
| 22813     | <b>showto</b>              | When the sender of the message was the user who is invoking <i>mailx</i> , write the                      |
| 22814     |                            | information from the <b>To:</b> line instead of the <b>From:</b> line in the header summary.              |
| 22815     |                            | The default shall be <b>noshowto</b> .                                                                    |
| 22816     | <b>sign=string</b>         | Set the variable inserted into the text of a message when the <b>~a</b> command escape is                 |
| 22817     |                            | given. The default shall be <b>nosign</b> . The character sequences ' <b>\t</b> ' and ' <b>\n</b> ' shall |
| 22818     |                            | be recognized in the variable as <tab>s and <newline>s, respectively. (See also <b>~i</b> in              |
| 22819     |                            | <b>Command Escapes in mailx</b> (on page 2803).)                                                          |

22820       **Sign=string** Set the variable inserted into the text of a message when the `~A` command escape is  
 22821            given. The default shall be **noSign**. The character sequences `'\t'` and `'\n'` shall  
 22822            be recognized in the variable as `<tab>s` and `<newline>s`, respectively.

22823       **toplines=number**  
 22824            Set the number of lines of the message to write with the **top** command. The default  
 22825            shall be 5.

## 22826       **Commands in mailx**

22827       The following *mailx* commands shall be provided. In the following list, header refers to lines  
 22828       from the message header, as shown in the OUTPUT FILES section. Header-line refers to lines  
 22829       within the header that begin with one or more non-white-space characters, immediately  
 22830       followed by a colon and white space and continuing until the next line beginning with a non-  
 22831       white-space character or an empty line. Header-field refers to the portion of a header line prior  
 22832       to the first colon in that line.

22833       For each of the commands listed below, the command can be entered as the abbreviation (those  
 22834       characters in the Synopsis command word preceding the `'[ '`), the full command (all characters  
 22835       shown for the command word, omitting the `'[ '` and `']'`), or any truncation of the full  
 22836       command down to the abbreviation. For example, the **exit** command (shown as **ex[it]** in the  
 22837       Synopsis) can be entered as **ex**, **exi**, or **exit**.

22838       The arguments to commands can be quoted, using the following methods:

- 22839       • An argument can be enclosed between paired double-quotes (`" "`) or single-quotes (`' '`); any  
 22840       white space, shell word expansion, or backslash characters within the quotes shall be treated  
 22841       literally as part of the argument. A double-quote shall be treated literally within single-  
 22842       quotes and *vice versa*. These special properties of the quote marks shall occur only when they  
 22843       are paired at the beginning and end of the argument.
- 22844       • A backslash outside of the enclosing quotes shall be discarded and the following character  
 22845       treated literally as part of the argument.
- 22846       • An unquoted backslash at the end of a command line shall be discarded and the next line  
 22847       shall continue the command.

22848       Filenames, where expected, shall be subjected to the process of shell word expansions (see  
 22849       Section 2.6 (on page 2238)); if more than a single pathname results and the command is  
 22850       expecting one file, the effects are unspecified. If the filename begins with an unquoted plus sign,  
 22851       it shall not be expanded, but treated as the named file (less the leading plus) in the **folder**  
 22852       directory. (See the **folder** variable.)

## 22853       **Declare Aliases**

22854       **Synopsis:**    a[alias] [alias [address...]]  
 22855                    g[roup] [alias [address...]]

22856       Add the given addresses to the alias specified by *alias*. The names shall be substituted when  
 22857       *alias* is used as a recipient address specified by the user in an outgoing message (that is, other  
 22858       recipients addressed indirectly through the **reply** command shall not be substituted in this  
 22859       manner). Mail address alias substitution shall apply only when the alias string is used as a full  
 22860       address; for example, when **hlj** is an alias, *hlj@posix.com* does not trigger the alias substitution. If  
 22861       no arguments are given, write a listing of the current aliases to standard output. If only an *alias*  
 22862       argument is given, write a listing of the specified alias to standard output. These listings need  
 22863       not reflect the same order of addresses that were entered.

22864       **Declare Alternatives**

22865       *Synopsis:*     alt[ernates] *name...*

22866       (See also the **metoo** command.) Declare a list of alternative names for the user's login. When  
22867       responding to a message, these names shall be removed from the list of recipients for the  
22868       response. The comparison of names shall be in a case-insensitive manner. With no arguments,  
22869       **alternates** shall write the current list of alternative names.

22870       **Change Current Directory**

22871       *Synopsis:*     cd [*directory*]

22872                ch[dir] [*directory*]

22873       Change directory. If *directory* is not specified, the contents of *HOME* shall be used.

22874       **Copy Messages**

22875       *Synopsis:*     c[opy] [*file*]

22876                c[opy] [*msglist*] *file*

22877                C[opy] [*msglist*]

22878       Copy messages to the file named by the pathname *file* without marking the messages as saved.  
22879       Otherwise, it shall be equivalent to the **save** command.

22880       In the capitalized form, save the specified messages in a file whose name is derived from the  
22881       author of the message to be saved, without marking the messages as saved. Otherwise, it shall  
22882       be equivalent to the **Save** command.

22883       **Delete Messages**

22884       *Synopsis:*     d[elete] [*msglist*]

22885       Mark messages for deletion from the mailbox. The deletions shall not occur until *mailx* quits (see  
22886       the **quit** command) or changes mailboxes (see the **folder** command). If **autoprint** is set and there  
22887       are messages remaining after the **delete** command, the current message shall be written as  
22888       described for the **print** command (see the **print** command); otherwise, the *mailx* prompt shall be  
22889       written.

22890       **Discard Header Fields**

22891       *Synopsis:*     di[scard] [*header-field...*]

22892                ig[nore] [*header-field...*]

22893       Suppress the specified header fields when writing messages. Specified *header-fields* shall be  
22894       added to the list of suppressed header fields. Examples of header fields to ignore are **status** and  
22895       **cc**. The fields shall be included when the message is saved. The **Print** and **Type** commands shall  
22896       override this command. The comparison of header fields shall be in a case-insensitive manner. If  
22897       no arguments are specified, write a list of the currently suppressed header fields to standard  
22898       output; the listing need not reflect the same order of header fields that were entered.

22899       If both **retain** and **discard** commands are given, **discard** commands shall be ignored.

22900 **Delete Messages and Display**

22901 *Synopsis:* dp [*msglist*]  
 22902 dt [*msglist*]

22903 Delete the specified messages as described for the **delete** command, except that the **autoprint**  
 22904 variable shall have no effect, and the current message shall be written only if it was set to a  
 22905 message after the last message deleted by the command. Otherwise, an informational message  
 22906 to the effect that there are no further messages in the mailbox shall be written, followed by the  
 22907 *mailx* prompt.

22908 **Echo a String**

22909 *Synopsis:* ec[*ho*] *string* ...

22910 Echo the given strings, equivalent to the shell *echo* utility.

22911 **Edit Messages**

22912 *Synopsis:* e[*dit*] [*msglist*]

22913 Edit the given messages. The messages shall be placed in a temporary file and the utility named  
 22914 by the *EDITOR* variable is invoked to edit each file in sequence. The default *EDITOR* is  
 22915 unspecified.

22916 The **edit** command does not modify the contents of those messages in the mailbox.

22917 **Exit**

22918 *Synopsis:* ex[*it*]  
 22919 x[*it*]

22920 Exit from *mailx* without changing the mailbox. No messages shall be saved in the **mbox** (see also  
 22921 **quit**).

22922 **Change Folder**

22923 *Synopsis:* fi[*le*] [*file*]  
 22924 fold[*er*] [*file*]

22925 Quit (see the **quit** command) from the current file of messages and read in the file named by the  
 22926 pathname *file*. If no argument is given, the name and status of the current mailbox shall be  
 22927 written.

22928 Several unquoted special characters shall be recognized when used as *file* names, with the  
 22929 following substitutions:

22930 % The system mailbox for the invoking user.

22931 %*user* The system mailbox for *user*.

22932 # The previous file.

22933 & The current **mbox**.

22934 +*file* The named file in the **folder** directory. (See the **folder** variable.)

22935 The default file shall be the current mailbox.

22936       **Display List of Folders**22937       *Synopsis:*     folders22938       Write the names of the files in the directory set by the **folder** variable. The command specified by  
22939       the *LISTER* environment variable shall be used (see the ENVIRONMENT VARIABLES section).22940       **Follow Up Specified Messages**22941       *Synopsis:*     fo[llowup] [*message*]22942               F[ollowup] [*msglist*]22943       In the lowercase form, respond to a message, recording the response in a file whose name is  
22944       derived from the author of the message. See also the **save** and **copy** commands and **outfolder**.22945       In the capitalized form, respond to the first message in the *msglist*, sending the message to the  
22946       author of each message in the *msglist*. The subject line shall be taken from the first message and  
22947       the response shall be recorded in a file whose name is derived from the author of the first  
22948       message. See also the **Save** and **Copy** commands and **outfolder**.22949       Both forms shall override the **record** variable, if set. |22950       **Display Header Summary for Specified Messages**22951       *Synopsis:*     f[rom] [*msglist*]

22952       Write the header summary for the specified messages.

22953       **Display Header Summary**22954       *Synopsis:*     h[eaders] [*message*]22955       Write the page of headers that includes the message specified. If the *message* argument is not  
22956       specified, the current message shall not change. However, if the *message* argument is specified,  
22957       the current message shall become the message that appears at the top of the page of headers that  
22958       includes the message specified. The **screen** variable sets the number of headers per page. See  
22959       also the **z** command.22960       **Help**22961       *Synopsis:*     hel[p]

22962               ?

22963       Write a summary of commands.

22964       **Hold Messages**22965       *Synopsis:*     ho[ld] [*msglist*]22966               pre[serve] [*msglist*]22967       Mark the messages in *msglist* to be retained in the mailbox when *mailx* terminates. This shall  
22968       override any commands that might previously have marked the messages to be deleted. During  
22969       the current invocation of *mailx*, only the **delete**, **dp**, or **dt** commands shall remove the *preserve*  
22970       marking of a message.

22971        **Execute Commands Conditionally**

22972        *Synopsis:*    i[f] s|r  
 22973                    mail-commands  
 22974                    el[se]  
 22975                    mail-commands  
 22976                    en[dif]

22977        Execute commands conditionally, where **if s** executes the following *mail-commands*, up to an  
 22978        **else** or **endif**, if the program is in Send Mode, and **if r** shall cause the *mail-commands* to be  
 22979        executed only in Receive Mode.

22980        **List Available Commands**

22981        *Synopsis:*    l[ist]

22982        Write a list of all commands available. No explanation shall be given.

22983        **Mail a Message**

22984        *Synopsis:*    m[ail] address...

22985        Mail a message to the specified addresses or aliases.

22986        **Direct Messages to mbox**

22987        *Synopsis:*    mb[ox] [msglist]

22988        Arrange for the given messages to end up in the **mbox** save file when *mailx* terminates normally.  
 22989        See *MBOX*. See also the **exit** and **quit** commands.

22990        **Process Next Specified Message**

22991        *Synopsis:*    n[ext] [message]

22992        If the current message has not been written (for example, by the **print** command) since *mailx*  
 22993        started or since any other message was the current message, behave as if the **print** command  
 22994        was entered. Otherwise, if there is an undeleted message after the current message, make it the  
 22995        current message and behave as if the **print** command was entered. Otherwise, an informational  
 22996        message to the effect that there are no further messages in the mailbox shall be written, followed  
 22997        by the *mailx* prompt.

22998        **Pipe Message**

22999        *Synopsis:*    pi[pe] [[msglist] command]  
 23000                    | [[msglist] command]

23001        Pipe the messages through the given *command* by invoking the command interpreter specified  
 23002        by *SHELL* with two arguments: **-c** and *command*. (See also *sh -c*.) The application shall ensure  
 23003        that the command is given as a single argument. Quoting, described previously, can be used to  
 23004        accomplish this. If no arguments are given, the current message shall be piped through the  
 23005        command specified by the value of the **cmd** variable. If the **page** variable is set, a <form-feed>  
 23006        shall be inserted after each message.

23007       **Display Message with Headers**

23008       *Synopsis:*    P[rint] [msglist]  
23009                    T[ype] [msglist]

23010       Write the specified messages, including all header lines, to standard output. Override  
23011       suppression of lines by the **discard**, **ignore**, and **retain** commands. If **crt** is set, the messages  
23012       longer than the number of lines specified by the **crt** variable shall be paged through the  
23013       command specified by the *PAGER* environment variable.

23014       **Display Message**

23015       *Synopsis:*    p[rint] [msglist]  
23016                    t[ype] [msglist]

23017       Write the specified messages to standard output. If **crt** is set, the messages longer than the  
23018       number of lines specified by the **crt** variable shall be paged through the command specified by  
23019       the *PAGER* environment variable.

23020       **Quit**

23021       *Synopsis:*    q[uit]  
23022                    end-of-file

23023       Terminate *mailx*, storing messages that were read in **mbox** (if the current mailbox is the system  
23024       mailbox and unless **hold** is set), deleting messages that have been explicitly saved (unless  
23025       **keepsave** is set), discarding messages that have been deleted, and saving all remaining messages  
23026       in the mailbox.

23027       **Reply to a Message List**

23028       *Synopsis:*    R[e]ply [msglist]  
23029                    R[espond] [msglist]

23030       Mail a reply message to the sender of each message in the *msglist*. The subject line shall be  
23031       formed by concatenating **Re:**<space> (unless it already begins with that string) and the subject  
23032       from the first message. If **record** is set to a filename, the response shall be saved at the end of that  
23033       file.

23034       See also the **flipr** variable.

23035       **Reply to a Message**

23036       *Synopsis:*    r[e]ply [message]  
23037                    r[espond] [message]

23038       Mail a reply message to all recipients included in the header of the message. The subject line  
23039       shall be formed by concatenating **Re:**<space> (unless it already begins with that string) and the  
23040       subject from the message. If **record** is set to a filename, the response shall be saved at the end of  
23041       that file.

23042       See also the **flipr** variable.

23043      **Retain Header Fields**23044      *Synopsis:*    ret[ain] [*header-field...*]

23045      Retain the specified header fields when writing messages. This command shall override all  
 23046      **discard** and **ignore** commands. The comparison of header fields shall be in a case-insensitive  
 23047      manner. If no arguments are specified, write a list of the currently retained header fields to  
 23048      standard output; the listing need not reflect the same order of header fields that were entered.

23049      **Save Messages**23050      *Synopsis:*    s[ave] [*file*]23051            s[ave] [*msglist*] *file*23052            S[ave] [*msglist*]

23053      Save the specified messages in the file named by the pathname *file*, or the **mbox** if the *file*  
 23054      argument is omitted. The file shall be created if it does not exist; otherwise, the messages shall be  
 23055      appended to the file. The message shall be put in the state *saved*, and shall behave as specified in  
 23056      the description of the *saved* state when the current mailbox is exited by the **quit** or **file**  
 23057      command.

23058      In the capitalized form, save the specified messages in a file whose name is derived from the  
 23059      author of the first message. The name of the file shall be taken to be the author's name with all  
 23060      network addressing stripped off. See also the **Copy**, **followup**, and **Followup** commands and  
 23061      **outfolder** variable.

23062      **Set Variables**23063      *Synopsis:*    se[t] [*name*[=*string*]] ...] [*name=number* ...] [*noname* ...]

23064      Define one or more variables called *name*. The variable can be given a null, string, or numeric  
 23065      value. Quoting and backslash escapes can occur anywhere in *string*, as described previously, as  
 23066      if the *string* portion of the argument were the entire argument. The forms *name* and *name=* shall  
 23067      be equivalent to *name=""* for variables that take string values. The **set** command without  
 23068      arguments shall write a list of all defined variables and their values. The **no name** form shall be  
 23069      equivalent to **unset name**.

23070      **Invoke a Shell**23071      *Synopsis:*    sh[ell]23072      Invoke an interactive command interpreter (see also *SHELL*).23073      **Display Message Size**23074      *Synopsis:*    si[ze] [*msglist*]

23075      Write the size in bytes of each of the specified messages.

23076      **Read mailx Commands From a File**23077      *Synopsis:*    so[urce] *file*

23078      Read and execute commands from the file named by the pathname *file* and return to command  
 23079      mode.

**23080 Display Beginning of Messages**

23081 *Synopsis:* to[p] [msglist]

23082 Write the top few lines of each of the specified messages. If the **toplines** variable is set, it is taken  
23083 as the number of lines to write. The default shall be 5.

**23084 Touch Messages**

23085 *Synopsis:* tou[ch] [msglist]

23086 Touch the specified messages. If any message in *msglist* is not specifically deleted nor saved in a  
23087 file, it shall be placed in the **mbox** upon normal termination. See **exit** and **quit**.

**23088 Delete Aliases**

23089 *Synopsis:* una[lias] [alias]...

23090 Delete the specified alias names. If a specified alias does not exist, the results are unspecified.

**23091 Undelete Messages**

23092 *Synopsis:* u[ndelete] [msglist]

23093 Change the state of the specified messages from deleted to read. If **autoprint** is set, the last  
23094 message of those restored shall be written. If *msglist* is not specified, the message shall be  
23095 selected as follows:

- 23096 • If there are any deleted messages that follow the current message, the first of these shall be  
23097 chosen.
- 23098 • Otherwise, the last deleted message that also precedes the current message shall be chosen.

**23099 Unset Variables**

23100 *Synopsis:* uns[et] name...

23101 Cause the specified variables to be erased.

**23102 Edit Message with Full-Screen Editor**

23103 *Synopsis:* v[isual] [msglist]

23104 Edit the given messages with a screen editor. Each message shall be placed in a temporary file,  
23105 and the utility named by the *VISUAL* variable shall be invoked to edit each file in sequence. The  
23106 default editor shall be *vi*.

23107 The **visual** command does not modify the contents of those messages in the mailbox.

**23108 Write Messages to a File**

23109 *Synopsis:* w[rite] [msglist] file

23110 Write the given messages to the file specified by the pathname *file*, minus the message header.  
23111 Otherwise, it shall be equivalent to the **save** command.

23112 **Scroll Header Display**23113 *Synopsis:*     z[+|-]

23114 Scroll the header display forward (if '+' is specified or if no option is specified) or backward (if  
 23115 '-' is specified) one screenful. The number of headers written shall be set by the **screen**  
 23116 variable.

23117 **Invoke Shell Command**23118 *Synopsis:*     !*command*

23119 Invoke the command interpreter specified by *SHELL* with two arguments: **-c** and *command*.  
 23120 (See also *sh -c*.) If the **bang** variable is set, each unescaped occurrence of '!' in *command* shall  
 23121 be replaced with the command executed by the previous ! *command* or **!** *command* escape.

23122 **Null Command**23123 *Synopsis:*     # *comment*23124 This null command (comment) shall be ignored by *mailx*.23125 **Display Current Message Number**23126 *Synopsis:*     =

23127 Write the current message number.

23128 **Command Escapes in mailx**

23129 The following commands can be entered only from input mode, by beginning a line with the  
 23130 escape character (by default, tilde ('~')). See the **escape** variable description for changing this  
 23131 special character. The format for the commands shall be:

23132 *<escape-character><command-char><separator>[<arguments>]* |23133 where the *<separator>* can be zero or more *<blank>*s. |

23134 In the following descriptions, the application shall ensure that the argument *command* (but not  
 23135 *mailx-command*) is a shell command string. Any string acceptable to the command interpreter  
 23136 specified by the *SHELL* variable when it is invoked as *SHELL -c command\_string* shall be valid.  
 23137 The command can be presented as multiple arguments (that is, quoting is not required).

23138 Command escapes that are listed with *msglist* or *mailx-command* arguments are invalid in Send  
 23139 Mode and produce unspecified results.

23140 **~!** *command* Invoke the command interpreter specified by *SHELL* with two arguments: **-c** and  
 23141 *command*; and then return to input mode. If the **bang** variable is set, each  
 23142 unescaped occurrence of '!' in *command* shall be replaced with the command  
 23143 executed by the previous ! *command* or **!** *command* escape.

23144 **~.** Simulate end-of-file (terminate message input).23145 **~:** *mailx-command*, **~\_** *mailx-command*

23146 Perform the command-level request.

23147 **~?** Write a summary of command escapes.23148 **~A** This shall be equivalent to **! Sign**.23149 **~a** This shall be equivalent to **! sign**.

- 23150       ~**b** *name...*    Add the *names* to the blind carbon copy (**Bcc**) list.
- 23151       ~**c** *name...*    Add the *names* to the carbon copy (**Cc**) list.
- 23152       ~**d**            Read in the dead-letter file. See *DEAD* for a description of this file.
- 23153       ~**e**            Invoke the editor, as specified by the *EDITOR* environment variable, on the partial message.
- 23154
- 23155       ~**f** [*msglist*]   Forward the specified messages. The specified messages shall be inserted into the current message without alteration. This command escape also shall insert message headers into the message with field selection affected by the **discard**, **ignore**, and **retain** commands.
- 23156
- 23157
- 23158
- 23159       ~**F** [*msglist*]   This shall be the equivalent of the ~**f** command escape, except that all headers shall be included in the message, regardless of previous **discard**, **ignore**, and **retain** commands.
- 23160
- 23161
- 23162       ~**h**            If standard input is a terminal, prompt for a **Subject** line and the **To**, **Cc**, and **Bcc** lists. Other implementation-defined headers may also be presented for editing. If the field is written with an initial value, it can be edited as if it had just been typed.
- 23163
- 23164
- 23165       ~**i** *string*    Insert the value of the named variable, followed by a <newline>, into the text of the message. If the string is unset or null, the message shall not be changed.
- 23166
- 23167       ~**m** [*msglist*]   Insert the specified messages into the message, prefixing non-empty lines with the string in the **indentprefix** variable. This command escape also shall insert message headers into the message, with field selection affected by the **discard**, **ignore**, and **retain** commands.
- 23168
- 23169
- 23170
- 23171       ~**M** [*msglist*]   This shall be the equivalent of the ~**m** command escape, except that all headers shall be included in the message, regardless of previous **discard**, **ignore**, and **retain** commands.
- 23172
- 23173
- 23174       ~**p**            Write the message being entered. If the message is longer than **crt** lines (see **Internal Variables in mailx** (on page 2792)), the output shall be paginated as described for the *PAGER* variable.
- 23175
- 23176
- 23177       ~**q**            Quit (see the **quit** command) from input mode by simulating an interrupt. If the body of the message is not empty, the partial message shall be saved in the dead-letter file. See *DEAD* for a description of this file.
- 23178
- 23179
- 23180       "~**r** *file*, ~< *file*, ~**r** *!command*, ~< *!command*"
- 23181            Read in the file specified by the pathname *file*. If the argument begins with an exclamation mark ('!'), the rest of the string shall be taken as an arbitrary system command; the command interpreter specified by *SHELL* shall be invoked with two arguments: **-c** and *command*. The standard output of *command* shall be inserted into the message.
- 23182
- 23183
- 23184
- 23185
- 23186       ~**s** *string*    Set the subject line to *string*.
- 23187
- 23187       ~**t** *name...*    Add the given *names* to the **To** list.
- 23188
- 23188       ~**v**            Invoke the full-screen editor, as specified by the *VISUAL* environment variable, on the partial message.
- 23189
- 23190       ~**w** *file*        Write the partial message, without the header, onto the file named by the pathname *file*. The file shall be created or the message shall be appended to it if the file exists.
- 23191
- 23192

23193       ~**x**           Exit as with ~**q**, except the message shall not be saved in the dead-letter file.

23194       ~| *command* Pipe the body of the message through the given *command* by invoking the  
23195           command interpreter specified by *SHELL* with two arguments: **-c** and *command*.  
23196           If the *command* returns a successful exit status, the standard output of the  
23197           command shall replace the message. Otherwise, the message shall remain  
23198           unchanged. If the *command* fails, an error message giving the exit status shall be  
23199           written.

23200 **EXIT STATUS**

23201       When the **-e** option is specified, the following exit values are returned:

23202       0 Mail was found.

23203       >0 Mail was not found or an error occurred.

23204       Otherwise, the following exit values are returned:

23205       0 Successful completion; note that this status implies that all messages were *sent*, but it gives  
23206       no assurances that any of them were actually *delivered*.

23207       >0 An error occurred.

23208 **CONSEQUENCES OF ERRORS**

23209       When in input mode (Receive Mode) or Send Mode:

23210       • If an error is encountered processing a command escape (see **Command Escapes in mailx**  
23211       (on page 2803)), a diagnostic message shall be written to standard error, and the message  
23212       being composed may be modified, but this condition shall not prevent the message from  
23213       being sent.

23214       • Other errors shall prevent the sending of the message.

23215       When in command mode:

23216       • Default.

23217 **APPLICATION USAGE**

23218       Delivery of messages to remote systems requires the existence of communication paths to such  
23219       systems. These need not exist.

23220       Input lines are limited to {*LINE\_MAX*} bytes, but mailers between systems may impose more  
23221       severe line-length restrictions. This volume of IEEE Std 1003.1-200x does not place any  
23222       restrictions on the length of messages handled by *mailx*, and for delivery of local messages the  
23223       only limitations should be the normal problems of available disk space for the target mail file.  
23224       When sending messages to external machines, applications are advised to limit messages to less  
23225       than 100,000 bytes because some mail gateways impose message-length restrictions.

23226       The format of the system mailbox is intentionally unspecified. Not all systems implement  
23227       system mailboxes as flat files, particularly with the advent of multimedia mail messages. Some  
23228       system mailboxes may be multiple files, others records in a database. The internal format of the  
23229       messages themselves are specified with the historical format from Version 7, but only after they  
23230       have been saved in some file other than the system mailbox. This was done so that many  
23231       historical applications expecting text-file mailboxes are not broken.

23232       Some new formats for messages can be expected in the future, probably including binary data,  
23233       bit maps, and various multimedia objects. As described here, *mailx* is not prohibited from  
23234       handling such messages, but it must store them as text files in secondary mailboxes (unless  
23235       some extension, such as a variable or command line option, is used to change the stored format).  
23236       Its method of doing so is implementation-defined and might include translating the data into

23237 text file-compatible or readable form or omitting certain portions of the message from the stored  
23238 output.

23239 The **discard** and **ignore** commands are not inverses of the **retain** command. The **retain**  
23240 command discards all header-fields except those explicitly retained. The **discard** command  
23241 keeps all header-fields except those explicitly discarded. If headers exist on the retained header  
23242 list, **discard** and **ignore** commands are ignored.

#### 23243 EXAMPLES

23244 None.

#### 23245 RATIONALE

23246 The standard developers felt strongly that a method for applications to send messages to  
23247 specific users was necessary. The obvious example is a batch utility, running non-interactively,  
23248 that wishes to communicate errors or results to a user. However, the actual format, delivery  
23249 mechanism, and method of reading the message are clearly beyond the scope of this volume of  
23250 IEEE Std 1003.1-200x.

23251 The intent of this command is to provide a simple, portable interface for sending messages non-  
23252 interactively. It merely defines a “front-end” to the historical mail system. It is suggested that  
23253 implementations explicitly denote the sender and recipient in the body of the delivered message.  
23254 Further specification of formats for either the message envelope or the message itself were  
23255 deliberately not made, as the industry is in the midst of changing from the current standards to  
23256 a more internationalized standard and it is probably incorrect, at this time, to require either one.

23257 Implementations are encouraged to conform to the various delivery mechanisms described in  
23258 the CCITT X.400 standards or to the equivalent Internet standards, described in Internet Request  
23259 for Comment (RFC) documents RFC 819, RFC 822, RFC 920, RFC 921, and RFC 1123.

23260 Many historical systems modified each body line that started with **From** by prefixing the ‘F’  
23261 with ‘>’. It is unnecessary, but allowed, to do that when the string does not follow a blank line  
23262 because it cannot be confused with the next header.

23263 The *edit* and *visual* commands merely edit the specified messages in a temporary file. They do  
23264 not modify the contents of those messages in the mailbox; such a capability could be added as an  
23265 extension, such as by using different command names.

23266 The restriction on a subject line being {LINE\_MAX}-10 bytes is based on the historical format  
23267 that consumes 10 bytes for **Subject:** and the trailing <newline>. Many historical mailers that a  
23268 message may encounter on other systems are not able to handle lines that long, however.

23269 Like the utilities *logger* and *lp*, *mailx* admittedly is difficult to test. This was not deemed sufficient  
23270 justification to exclude this utility from this volume of IEEE Std 1003.1-200x. It is also arguable  
23271 that it is, in fact, testable, but that the tests themselves are not portable.

23272 When *mailx* is being used by an application that wishes to receive the results as if none of the  
23273 User Portability Utilities option features were supported, the *DEAD* environment variable must  
23274 be set to **/dev/null**. Otherwise, it may be subject to the file creations described in *mailx*  
23275 ASYNCHRONOUS EVENTS. Similarly, if the *MAILRC* environment variable is not set to  
23276 **/dev/null**, historical versions of *mailx* and *Mail* read initialization commands from a file before  
23277 processing begins. Since the initialization that a user specifies could alter the contents of  
23278 messages an application is trying to send, such applications must set *MAILRC* to **/dev/null**.

23279 The description of *LC\_TIME* uses “may affect” because many historical implementations do not  
23280 or cannot manipulate the date and time strings in the incoming mail headers. Some headers  
23281 found in incoming mail do not have enough information to determine the timezone in which the  
23282 mail originated, and, therefore, *mailx* cannot convert the date and time strings into the internal  
23283 form that then is parsed by routines like *strftime()* that can take *LC\_TIME* settings into account.

23284 Changing all these times to a user-specified format is allowed, but not required.

23285 The paginator selected when *PAGER* is null or unset is partially unspecified to allow the System  
23286 V historical practice of using *pg* as the default. Bypassing the pagination function, such as by  
23287 declaring that *cat* is the paginator, would not meet with the intended meaning of this  
23288 description. However, any “portable user” would have to set *PAGER* explicitly to get his or her  
23289 preferred paginator on all systems. The paginator choice was made partially unspecified, unlike  
23290 the *VISUAL* editor choice (mandated to be *vi*) because most historical pagers follow a common  
23291 theme of user input, whereas editors differ dramatically.

23292 Options to specify addresses as **cc** (carbon copy) or **bcc** (blind carbon copy) were considered to  
23293 be format details and were omitted.

23294 A zero exit status implies that all messages were *sent*, but it gives no assurances that any of them  
23295 were actually *delivered*. The reliability of the delivery mechanism is unspecified and is an  
23296 appropriate marketing distinction between systems.

23297 In order to conform to the Utility Syntax Guidelines, a solution was required to the optional *file*  
23298 option-argument to *-f*. By making *file* an operand, the guidelines are satisfied and users remain  
23299 portable. However, it does force implementations to support usage such as:

23300 `mailx -fin mymail.box`

23301 The **no name** method of unsetting variables is not present in all historical systems, but it is in  
23302 System V and provides a logical set of commands corresponding to the format of the display of  
23303 options from the *mailx set* command without arguments.

23304 The **ask** and **asksub** variables are the names selected by BSD and System V, respectively, for the  
23305 same feature. They are synonyms in this volume of IEEE Std 1003.1-200x.

23306 The *mailx echo* command was not documented in the BSD version and has been omitted here  
23307 because it is not obviously useful for interactive users.

23308 The default prompt on the System V *mailx* is a question mark, on BSD *Mail* an ampersand. Since  
23309 this volume of IEEE Std 1003.1-200x chose the *mailx* name, it kept the System V default,  
23310 assuming that BSD users would not have difficulty with this minor incompatibility (that they  
23311 can override).

23312 The meanings of **r** and **R** are reversed between System V *mailx* and SunOS *Mail*. Once again,  
23313 since this volume of IEEE Std 1003.1-200x chose the *mailx* name, it kept the System V default, but  
23314 allows the SunOS user to achieve the desired results using **flipr**, an internal variable in System V  
23315 *mailx*, although it has not been documented in the SVID

23316 The **indentprefix** variable, the **retain** and **unalias** commands, and the **␣** and **␣** command  
23317 escapes were adopted from 4.3 BSD *Mail*.

23318 The **version** command was not included because no sufficiently general specification of the  
23319 version information could be devised that would still be useful to a portable user. This  
23320 command name should be used by suppliers who wish to provide version information about the  
23321 *mailx* command.

23322 The “implementation-specific (unspecified) system start-up file” historically has been named  
23323 **/etc/mailx.rc**, but this specific name and location are not required.

23324 The intent of the wording for the **next** command is that if any command has already displayed  
23325 the current message it should display a following message, but, otherwise, it should display the  
23326 current message. Consider the command sequence:

23327 `next 3`  
23328 `delete 3`

- 23329 next
- 23330 where the **autoprint** option was not set. The normative text specifies that the second **next**  
23331 command should display a message following the third message, because even though the  
23332 current message has not been displayed since it was set by the **delete** command, it has been  
23333 displayed since the current message was anything other than message number 3. This does not  
23334 always match historical practice in some implementations, where the command file address  
23335 followed by **next** (or the default command) would skip the message for which the user had  
23336 searched.
- 23337 **FUTURE DIRECTIONS**
- 23338 None.
- 23339 **SEE ALSO**
- 23340 *ed, ls, more, vi*
- 23341 **CHANGE HISTORY**
- 23342 First released in Issue 2.
- 23343 **Issue 5**
- 23344 The description of the EDITOR environment variable is changed to indicate that *ed* is the default  
23345 editor if this variable is not set. In previous issues, this default was not stated explicitly at this  
23346 point but was implied further down in the text.
- 23347 FUTURE DIRECTIONS section added.
- 23348 **Issue 6**
- 23349 The following new requirements on POSIX implementations derive from alignment with the  
23350 Single UNIX Specification:
- 23351
  - The **-F** option is added.
- 23352
  - The **allnet**, **debug**, and **sendwait** internal variables are added.
- 23353
  - The **C**, **ec**, **fo**, **F**, and **S** *mailx* commands are added.
- 23354 In the DESCRIPTION and ENVIRONMENT VARIABLES sections, text stating “*HOME*  
23355 directory” is replaced by “directory referred to by the *HOME* environment variable”.
- 23356 The *mailx* utility is aligned with the IEEE P1003.2b draft standard, which included various  
23357 clarifications to resolve IEEE PASC Interpretations submitted for the ISO POSIX-2:1993  
23358 standard. In particular, the changes here address IEEE PASC Interpretations 1003.2 #10, #11,  
23359 #103, #106, #108, #114, #115, #122, and #129.
- 23360 The normative text is reworded to avoid use of the term “must” for application requirements.
- 23361 The *TZ* entry is added to the ENVIRONMENT VARIABLES section.

23362 **NAME**23363 make — maintain, update, and regenerate groups of programs (**DEVELOPMENT**)23364 **SYNOPSIS**

```
23365 SD make [-einpqrst][-f makefile...] [-k | -S][macro=value]...
23366           [target_name...]
```

23367

23368 **DESCRIPTION**

23369 The *make* utility shall update files that are derived from other files. A typical case is one where |  
 23370 object files are derived from the corresponding source files. The *make* utility examines time |  
 23371 relationships and shall update those derived files (called targets) that have modified times |  
 23372 earlier than the modified times of the files (called prerequisites) from which they are derived. A |  
 23373 description file (makefile) contains a description of the relationships between files, and the |  
 23374 commands that need to be executed to update the targets to reflect changes in their |  
 23375 prerequisites. Each specification, or rule, shall consist of a target, optional prerequisites, and |  
 23376 optional commands to be executed when a prerequisite is newer than the target. There are two |  
 23377 types of rule:

23378 1. *Inference rules*, which have one target name with at least one period ( '.' ) and no slash |  
 23379 ( '/' )

23380 2. *Target rules*, which can have more than one target name

23381 In addition, *make* shall have a collection of built-in macros and inference rules that infer |  
 23382 prerequisite relationships to simplify maintenance of programs.

23383 To receive exactly the behavior described in this section, the user shall ensure that a portable |  
 23384 makefile shall:

- 23385 • Include the special target **.POSIX** |
- 23386 • Omit any special target reserved for implementations (a leading period followed by |  
 23387 uppercase letters) that has not been specified by this section

23388 The behavior of *make* is unspecified if either or both of these conditions are not met.

23389 **OPTIONS**

23390 The *make* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section |  
 23391 12.2, Utility Syntax Guidelines.

23392 The following options shall be supported:

23393 **-e** Cause environment variables, including those with null values, to override macro |  
 23394 assignments within makefiles.

23395 **-f *makefile*** Specify a different makefile. The argument *makefile* is a pathname of a description |  
 23396 file, which is also referred to as the *makefile*. A pathname of '-' shall denote the |  
 23397 standard input. There can be multiple instances of this option, and they shall be |  
 23398 processed in the order specified. The effect of specifying the same option- |  
 23399 argument more than once is unspecified.

23400 **-i** Ignore error codes returned by invoked commands. This mode is the same as if the |  
 23401 special target **.IGNORE** were specified without prerequisites.

23402 **-k** Continue to update other targets that do not depend on the current target if a non- |  
 23403 ignored error occurs while executing the commands to bring a target up-to-date.

23404 **-n** Write commands that would be executed on standard output, but do not execute |  
 23405 them. However, lines with a plus sign ( '+' ) prefix shall be executed. In this mode,

- 23406 lines with an at sign ('@') character prefix shall be written to standard output.
- 23407 **-p** Write to standard output the complete set of macro definitions and target  
23408 descriptions. The output format is unspecified.
- 23409 **-q** Return a zero exit value if the target file is up-to-date; otherwise, return an exit  
23410 value of 1. Targets shall not be updated if this option is specified. However, a  
23411 makefile command line (associated with the targets) with a plus sign ('+') prefix  
23412 shall be executed.
- 23413 **-r** Clear the suffix list and does not use the built-in rules.
- 23414 **-S** Terminate *make* if an error occurs while executing the commands to bring a target  
23415 up-to-date. This shall be the default and the opposite of **-k**.
- 23416 **-s** Do not write makefile command lines or touch messages (see **-t**) to standard  
23417 output before executing. This mode shall be the same as if the special target  
23418 **.SILENT** were specified without prerequisites.
- 23419 **-t** Update the modification time of each target as though a *touch target* had been  
23420 executed. Targets that have prerequisites but no commands (see **Target Rules** (on  
23421 page 2813)), or that are already up-to-date, shall not be touched in this manner.  
23422 Write messages to standard output for each target file indicating the name of the  
23423 file and that it was touched. Normally, the makefile command lines associated  
23424 with each target are not executed. However, a command line with a plus sign  
23425 ('+') prefix shall be executed.
- 23426 Any options specified in the *MAKEFLAGS* environment variable shall be evaluated before any  
23427 options specified on the *make* utility command line. If the **-k** and **-S** options are both specified  
23428 on the *make* utility command line or by the *MAKEFLAGS* environment variable, the last option  
23429 specified shall take precedence. If the **-f** or **-p** options appear in the *MAKEFLAGS* environment  
23430 variable, the result is undefined.
- 23431 **OPERANDS**
- 23432 The following operands shall be supported:
- 23433 *target\_name* Target names, as defined in the EXTENDED DESCRIPTION section. If no target is  
23434 specified, while *make* is processing the makefiles, the first target that *make*  
23435 encounters that is not a special target or an inference rule shall be used.
- 23436 *macro=value* Macro definitions, as defined in **Macros** (on page 2815).
- 23437 If the *target\_name* and *macro=value* operands are intermixed on the *make* utility command line,  
23438 the results are unspecified.
- 23439 **STDIN**
- 23440 The standard input shall be used only if the *makefile* option-argument is '-'. See the INPUT  
23441 FILES section.
- 23442 **INPUT FILES**
- 23443 The input file, otherwise known as the makefile, is a text file containing rules, macro definitions, |  
23444 and comments. See the EXTENDED DESCRIPTION section. |
- 23445 **ENVIRONMENT VARIABLES**
- 23446 The following environment variables shall affect the execution of *make*:
- 23447 *LANG* Provide a default value for the internationalization variables that are unset or null.  
23448 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
23449 Internationalization Variables for the precedence of internationalization variables  
23450 used to determine the values of locale categories.)

- 23451 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
23452 internationalization variables.
- 23453 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
23454 characters (for example, single-byte as opposed to multi-byte characters in  
23455 arguments and input files).
- 23456 *LC\_MESSAGES*  
23457 Determine the locale that should be used to affect the format and contents of  
23458 diagnostic messages written to standard error.
- 23459 *MAKEFLAGS*  
23460 This variable shall be interpreted as a character string representing a series of  
23461 option characters to be used as the default options. The implementation shall  
23462 accept both of the following formats (but need not accept them when intermixed):
- 23463 • The characters are option letters without the leading hyphens or <blank>  
23464 separation used on a *make* utility command line.
  - 23465 • The characters are formatted in a manner similar to a portion of the *make* utility  
23466 command line: options are preceded by hyphens and <blank>-separated as  
23467 described in the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2,  
23468 Utility Syntax Guidelines. The *macro=value* macro definition operands can also  
23469 be included. The difference between the contents of *MAKEFLAGS* and the *make*  
23470 utility command line is that the contents of the variable shall not be subjected  
23471 to the word expansions (see Section 2.6 (on page 2238)) associated with parsing  
23472 the command line values.
- 23473 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 23474 XSI *PROJECTDIR*  
23475 Provide a directory to be used to search for SCCS files not found in the current  
23476 directory. In all of the following cases, the search for SCCS files is made in the  
23477 directory *SCCS* in the identified directory. If the value of *PROJECTDIR* begins  
23478 with a slash, it shall be considered an absolute pathname; otherwise, the value of  
23479 *PROJECTDIR* is treated as a user name and that user's initial working directory  
23480 shall be examined for a subdirectory *src* or *source*. If such a directory is found, it  
23481 shall be used. Otherwise, the value is used as a relative pathname.
- 23482 If *PROJECTDIR* is not set or has a null value, the search for SCCS files shall be  
23483 made in the directory *SCCS* in the current directory.
- 23484 The setting of *PROJECTDIR* affects all files listed in the remainder of this utility  
23485 description for files with a component named *SCCS*.
- 23486 The value of the *SHELL* environment variable shall not be used as a macro and shall not be  
23487 modified by defining the *SHELL* macro in a makefile or on the command line. All other  
23488 environment variables, including those with null values, shall be used as macros, as defined in  
23489 *Macros* (on page 2815).
- 23490 **ASYNCHRONOUS EVENTS**  
23491 If not already ignored, *make* shall trap SIGHUP, SIGTERM, SIGINT, and SIGQUIT and remove  
23492 the current target unless the target is a directory or the target is a prerequisite of the special  
23493 target *.PRECIOUS* or unless one of the *-n*, *-p*, or *-q* options was specified. Any targets removed  
23494 in this manner shall be reported in diagnostic messages of unspecified format, written to  
23495 standard error. After this cleanup process, if any, *make* shall take the standard action for all other  
23496 signals.

23497 **STDOUT**

23498 The *make* utility shall write all commands to be executed to standard output unless the `-s` option  
 23499 was specified, the command is prefixed with an at sign, or the special target `.SILENT` has either  
 23500 the current target as a prerequisite or has no prerequisites. If *make* is invoked without any work  
 23501 needing to be done, it shall write a message to standard output indicating that no action was  
 23502 taken. If the `-t` option is present and a file is touched, *make* shall write to standard output a  
 23503 message of unspecified format indicating that the file was touched, including the filename of the  
 23504 file.

23505 **STDERR**

23506 The standard error shall be used only for diagnostic messages.

23507 **OUTPUT FILES**

23508 Files can be created when the `-t` option is present. Additional files can also be created by the  
 23509 utilities invoked by *make*.

23510 **EXTENDED DESCRIPTION**

23511 The *make* utility attempts to perform the actions required to ensure that the specified targets are  
 23512 up-to-date. A target is considered out-of-date if it is older than any of its prerequisites or if it  
 23513 does not exist. The *make* utility shall treat all prerequisites as targets themselves and recursively  
 23514 ensure that they are up-to-date, processing them in the order in which they appear in the rule.  
 23515 The *make* utility shall use the modification times of files to determine whether the corresponding  
 23516 targets are out-of-date.

23517 After *make* has ensured that all of the prerequisites of a target are up-to-date and if the target is  
 23518 out-of-date, the commands associated with the target entry shall be executed. If there are no  
 23519 commands listed for the target, the target shall be treated as up-to-date.

23520 **Makefile Syntax**

23521 A makefile can contain rules, macro definitions (see **Macros** (on page 2815)), and comments.  
 23522 There are two kinds of rules: *inference rules* and *target rules*. The *make* utility shall contain a set of  
 23523 built-in inference rules. If the `-r` option is present, the built-in rules shall not be used and the  
 23524 suffix list shall be cleared. Additional rules of both types can be specified in a makefile. If a rule  
 23525 is defined more than once, the value of the rule shall be that of the last one specified. Macros can  
 23526 also be defined more than once, and the value of the macro is specified in **Macros** (on page  
 23527 2815). Comments start with a number sign (' # ') and continue until an unescaped <newline> is  
 23528 reached.

23529 By default, the following files shall be tried in sequence: `./makefile` and `./Makefile`. If neither  
 23530 `./makefile` or `./Makefile` are found, other implementation-defined files may also be tried. On  
 23531 XSI-conformant systems, the additional files `./s.makefile`, `SCCS/s.makefile`, `./s.Makefile`, and  
 23532 `SCCS/s.Makefile` shall also be tried.

23533 The `-f` option shall direct *make* to ignore any of these default files and use the specified argument  
 23534 as a makefile instead. If the `'-'` argument is specified, standard input shall be used.

23535 The term *makefile* is used to refer to any rules provided by the user, whether in `./makefile` or its  
 23536 variants, or specified by the `-f` option.

23537 The rules in makefiles shall consist of the following types of lines: target rules, including special  
 23538 targets (see **Target Rules** (on page 2813)), inference rules (see **Inference Rules** (on page 2816)),  
 23539 macro definitions (see **Macros** (on page 2815)), empty lines, and comments.

23540 When an escaped <newline> (one preceded by a backslash) is found anywhere in the makefile  
 23541 except in a command line, it shall be replaced, along with any leading white space on the  
 23542 following line, with a single <space>. When an escaped <newline> is found in a command line

23543 in a makefile, the command line shall contain the backslash, the <newline>, and the next line,  
23544 except that the first character of the next line shall not be included if it is a <tab>.

### 23545 **Makefile Execution**

23546 Makefile command lines shall be processed one at a time by writing the makefile command line  
23547 to the standard output (unless one of the conditions listed under '@' suppresses the writing)  
23548 and executing the command(s) in the line. A <tab> may precede the command to standard  
23549 output. Command execution shall be as if the makefile command line were the argument to the  
23550 *system()* function. The environment for the command being executed shall contain all of the  
23551 variables in the environment of *make*.

23552 By default, when *make* receives a non-zero status from the execution of a command, it shall  
23553 terminate with an error message to standard error. |

23554 Makefile command lines can have one or more of the following prefixes: a hyphen ('-'), an at  
23555 sign ('@'), or a plus sign ('+'). These shall modify the way in which *make* processes the |  
23556 command. When a command is written to standard output, the prefix shall not be included in  
23557 the output.

23558 – If the command prefix contains a hyphen, or the **-i** option is present, or the special target  
23559 **.IGNORE** has either the current target as a prerequisite or has no prerequisites, any error  
23560 found while executing the command shall be ignored.

23561 @ If the command prefix contains an at sign and the *make* utility command line **-n** option is  
23562 not specified, or the **-s** option is present, or the special target **.SILENT** has either the current  
23563 target as a prerequisite or has no prerequisites, the command shall not be written to  
23564 standard output before it is executed.

23565 + If the command prefix contains a plus sign, this indicates a makefile command line that  
23566 shall be executed even if **-n**, **-q**, or **-t** is specified.

### 23567 **Target Rules**

23568 Target rules are formatted as follows:

```
23569 target [target...]: [prerequisite...][;command]
23570 [<tab>command
23571 <tab>command
23572 ...]
```

23573 *line that does not begin with <tab>*

23574 Target entries are specified by a <blank>-separated, non-null list of targets, then a colon, then a  
23575 <blank>-separated, possibly empty list of prerequisites. Text following a semicolon, if any, and  
23576 all following lines that begin with a <tab>, are makefile command lines to be executed to update  
23577 the target. The first non-empty line that does not begin with a <tab> or '#' shall begin a new  
23578 entry. An empty or blank line, or a line beginning with '#', may begin a new entry.

23579 Applications shall select target names from the set of characters consisting solely of periods,  
23580 underscores, digits, and alphabetic characters from the portable character set (see the Base Definitions  
23581 volume of IEEE Std 1003.1-200x, Section 6.1, Portable Character Set). Implementations may  
23582 allow other characters in target names as extensions. The interpretation of targets containing the  
23583 characters '%' and '"' is implementation-defined.

23584 A target that has prerequisites, but does not have any commands, can be used to add to the  
23585 prerequisite list for that target. Only one target rule for any given target can contain commands.

- 23586 Lines that begin with one of the following are called *special targets* and control the operation of  
23587 *make*:
- 23588 **.DEFAULT** If the makefile uses this special target, the application shall ensure that it is  
23589 specified with commands, but without prerequisites. The commands shall be used  
23590 by *make* if there are no other rules available to build a target.
- 23591 **.IGNORE** Prerequisites of this special target are targets themselves; this shall cause errors  
23592 from commands associated with them to be ignored in the same manner as  
23593 specified by the `-i` option. Subsequent occurrences of **.IGNORE** shall add to the  
23594 list of targets ignoring command errors. If no prerequisites are specified, *make* shall  
23595 behave as if the `-i` option had been specified and errors from all commands  
23596 associated with all targets shall be ignored.
- 23597 **.POSIX** The application shall ensure that this special target is specified without  
23598 prerequisites or commands. If it appears as the first non-comment line in the  
23599 makefile, *make* shall process the makefile as specified by this section; otherwise, the  
23600 behavior of *make* is unspecified.
- 23601 **.PRECIOUS** Prerequisites of this special target shall not be removed if *make* receives one of the  
23602 asynchronous events explicitly described in the ASYNCHRONOUS EVENTS  
23603 section. Subsequent occurrences of **.PRECIOUS** shall add to the list of precious  
23604 files. If no prerequisites are specified, all targets in the makefile shall be treated as  
23605 if specified with **.PRECIOUS**.
- 23606 XSI **.SCCS\_GET** The application shall ensure that this special target is specified without  
23607 prerequisites. If this special target is included in a makefile, the commands  
23608 specified with this target shall replace the default commands associated with this  
23609 special target (see **Default Rules** (on page 2819)). The commands specified with  
23610 this target are used to get all SCCS files that are not found in the current directory.
- 23611 When source files are named in a dependency list, *make* shall treat them just like |  
23612 any other target. Because the source file is presumed to be present in the directory, |  
23613 there is no need to add an entry for it to the makefile. When a target has no |  
23614 dependencies, but is present in the directory, *make* shall assume that that file is up- |  
23615 to-date. If, however, an SCCS file named **SCCS/s.source\_file** is found for a target |  
23616 *source\_file*, *make* compares the timestamp of the target file with that of the |  
23617 **SCCS/s.source\_file** to assure the target is up-to-date. If the target is missing, or if |  
23618 the SCCS file is newer, *make* shall automatically issue the commands specified for |  
23619 the **.SCCS\_GET** special target to retrieve the most recent version. However, if the |  
23620 target is writable by anyone, *make* shall not retrieve a new version. |
- 23621 **.SILENT** Prerequisites of this special target are targets themselves; this shall cause  
23622 commands associated with them to not be written to the standard output before  
23623 they are executed. Subsequent occurrences of **.SILENT** shall add to the list of  
23624 targets with silent commands. If no prerequisites are specified, *make* shall behave  
23625 as if the `-s` option had been specified and no commands or touch messages  
23626 associated with any target shall be written to standard output.
- 23627 **.SUFFIXES** Prerequisites of **.SUFFIXES** shall be appended to the list of known suffixes and are  
23628 used in conjunction with the inference rules (see **Inference Rules** (on page 2816)).  
23629 If **.SUFFIXES** does not have any prerequisites, the list of known suffixes shall be  
23630 cleared.
- 23631 The special targets **.IGNORE**, **.POSIX**, **.PRECIOUS**, **.SILENT**, and **.SUFFIXES** shall be specified  
23632 without commands.

23633 Targets with names consisting of a leading period followed by the uppercase letters "POSIX"  
 23634 and then any other characters are reserved for future standardization. Targets with names  
 23635 consisting of a leading period followed by one or more uppercase letters are reserved for  
 23636 implementation extensions.

### 23637 **Macros**

23638 Macro definitions are in the form:

```
23639 string1 = [string2]
```

23640 The macro named *string1* is defined as having the value of *string2*, where *string2* is defined as all  
 23641 characters, if any, after the equal sign, up to a comment character ('#') or an unescaped  
 23642 <newline>. Any <blank>s immediately before or after the equal sign shall be ignored.

23643 Applications shall select macro names from the set of characters consisting solely of periods,  
 23644 underscores, digits, and alphabetic characters from the portable character set (see the Base Definitions  
 23645 volume of IEEE Std 1003.1-200x, Section 6.1, Portable Character Set). A macro name shall not  
 23646 contain an equals sign. Implementations may allow other characters in macro names as  
 23647 extensions.

23648 Macros can appear anywhere in the makefile. Macro expansions using the forms  $\$(string1)$  or  
 23649  $\${string1}$  shall be replaced by *string2*, as follows:

- 23650 • Macros in target lines shall be evaluated when the target line is read.
- 23651 • Macros in makefile command lines shall be evaluated when the command is executed.
- 23652 • Macros in the string before the equals sign in a macro definition shall be evaluated when the  
 23653 macro assignment is made.
- 23654 • Macros after the equals sign in a macro definition shall not be evaluated until the defined  
 23655 macro is used in a rule or command, or before the equals sign in a macro definition.

23656 The parentheses or braces are optional if *string1* is a single character. The macro \$\$ shall be  
 23657 replaced by the single character '\$'. If *string1* in a macro expansion contains a macro  
 23658 expansion, the results are unspecified.

23659 Macro expansions using the forms  $\$(string1[:subst1=[subst2]])$  or  $\${string1[:subst1=[subst2]]}$  can  
 23660 be used to replace all occurrences of *subst1* with *subst2* when the macro substitution is  
 23661 performed. The *subst1* to be replaced shall be recognized when it is a suffix at the end of a word  
 23662 in *string1* (where a *word*, in this context, is defined to be a string delimited by the beginning of  
 23663 the line, a <blank> or <newline>). If *string1* in a macro expansion contains a macro expansion,  
 23664 the results are unspecified.

23665 Macro expansions in *string1* of macro definition lines shall be evaluated when read. Macro  
 23666 expansions in *string2* of macro definition lines shall be performed when the macro identified by  
 23667 *string1* is expanded in a rule or command.

23668 Macro definitions shall be taken from the following sources, in the following logical order,  
 23669 before the makefile(s) are read.

- 23670 1. Macros specified on the *make* utility command line, in the order specified on the command  
 23671 line. It is unspecified whether the internal macros defined in **Internal Macros** (on page  
 23672 2818) are accepted from this source.
- 23673 2. Macros defined by the *MAKEFLAGS* environment variable, in the order specified in the  
 23674 environment variable. It is unspecified whether the internal macros defined in **Internal  
 23675 Macros** (on page 2818) are accepted from this source.

23676 3. The contents of the environment, excluding the *MAKEFLAGS* and *SHELL* variables and  
23677 including the variables with null values.

23678 4. Macros defined in the inference rules built into *make*.

23679 Macro definitions from these sources shall not override macro definitions from a lower-  
23680 numbered source. Macro definitions from a single source (for example, the *make* utility  
23681 command line, the *MAKEFLAGS* environment variable, or the other environment variables) shall  
23682 override previous macro definitions from the same source.

23683 Macros defined in the makefile(s) shall override macro definitions that occur before them in the  
23684 makefile(s) and macro definitions from source 4. If the *-e* option is not specified, macros defined  
23685 in the makefile(s) shall override macro definitions from source 3. Macros defined in the  
23686 makefile(s) shall not override macro definitions from source 1 or source 2.

23687 Before the makefile(s) are read, all of the *make* utility command line options (except *-f* and *-p*)  
23688 and *make* utility command line macro definitions (except any for the *MAKEFLAGS* macro), not  
23689 already included in the *MAKEFLAGS* macro, shall be added to the *MAKEFLAGS* macro, quoted  
23690 in an implementation-defined manner such that when *MAKEFLAGS* is read by another instance  
23691 of the *make* command, the original macro's value is recovered. Other implementation-defined  
23692 options and macros may also be added to the *MAKEFLAGS* macro. If this modifies the value of  
23693 the *MAKEFLAGS* macro, or, if the *MAKEFLAGS* macro is modified at any subsequent time, the  
23694 *MAKEFLAGS* environment variable shall be modified to match the new value of the  
23695 *MAKEFLAGS* macro. The result of setting *MAKEFLAGS* in the Makefile is unspecified.

23696 Before the makefile(s) are read, all of the *make* utility command line macro definitions (except the  
23697 *MAKEFLAGS* macro or the *SHELL* macro) shall be added to the environment of *make*. Other  
23698 implementation-defined variables may also be added to the environment of *make*.

23699 The *SHELL* macro shall be treated specially. It shall be provided by *make* and set to the  
23700 pathname of the shell command language interpreter (see *sh* (on page 3048)). The *SHELL*  
23701 environment variable shall not affect the value of the *SHELL* macro. If *SHELL* is defined in the  
23702 makefile or is specified on the command line, it shall replace the original value of the *SHELL*  
23703 macro, but shall not affect the *SHELL* environment variable. Other effects of defining *SHELL* in  
23704 the makefile or on the command line are implementation-defined.

## 23705 Inference Rules

23706 Inference rules are formatted as follows:

```
23707 target:
23708 <tab>command
23709 [ <tab>command ]
23710 ...
```

23711 *line that does not begin with <tab> or #*

23712 The application shall ensure that the *target* portion is a valid target name (see **Target Rules** (on  
23713 page 2813)) of the form *.s2* or *.s1.s2* (where *.s1* and *.s2* are suffixes that have been given as  
23714 prerequisites of the *.SUFFIXES* special target and *s1* and *s2* do not contain any slashes or  
23715 periods.) If there is only one period in the target, it is a single-suffix inference rule. Targets with  
23716 two periods are double-suffix inference rules. Inference rules can have only one target before the  
23717 colon.

23718 The application shall ensure that the makefile does not specify prerequisites for inference rules;  
23719 no characters other than white space shall follow the colon in the first line, except when creating  
23720 the *empty rule*, described below. Prerequisites are inferred, as described below.

23721 Inference rules can be redefined. A target that matches an existing inference rule shall overwrite  
 23722 the old inference rule. An empty rule can be created with a command consisting of simply a  
 23723 semicolon (that is, the rule still exists and is found during inference rule search, but since it is  
 23724 empty, execution has no effect). The empty rule also can be formatted as follows:

23725 `rule: ;`

23726 where zero or more <blank>s separate the colon and semicolon.

23727 The *make* utility uses the suffixes of targets and their prerequisites to infer how a target can be  
 23728 made up-to-date. A list of inference rules defines the commands to be executed. By default, *make*  
 23729 contains a built-in set of inference rules. Additional rules can be specified in the makefile.

23730 The special target **.SUFFIXES** contains as its prerequisites a list of suffixes that shall be used by  
 23731 the inference rules. The order in which the suffixes are specified defines the order in which the  
 23732 inference rules for the suffixes are used. New suffixes shall be appended to the current list by  
 23733 specifying a **.SUFFIXES** special target in the makefile. A **.SUFFIXES** target with no prerequisites  
 23734 shall clear the list of suffixes. An empty **.SUFFIXES** target followed by a new **.SUFFIXES** list is  
 23735 required to change the order of the suffixes.

23736 Normally, the user would provide an inference rule for each suffix. The inference rule to update  
 23737 a target with a suffix **.s1** from a prerequisite with a suffix **.s2** is specified as a target **.s2.s1**. The  
 23738 internal macros provide the means to specify general inference rules (see **Internal Macros** (on  
 23739 page 2818)).

23740 When no target rule is found to update a target, the inference rules shall be checked. The suffix  
 23741 of the target (**.s1**) to be built is compared to the list of suffixes specified by the **.SUFFIXES** special  
 23742 targets. If the **.s1** suffix is found in **.SUFFIXES**, the inference rules shall be searched in the order  
 23743 defined for the first **.s2.s1** rule whose prerequisite file (**\$\*.s2**) exists. If the target is out-of-date  
 23744 with respect to this prerequisite, the commands for that inference rule shall be executed.

23745 If the target to be built does not contain a suffix and there is no rule for the target, the single  
 23746 suffix inference rules shall be checked. The single-suffix inference rules define how to build a  
 23747 target if a file is found with a name that matches the target name with one of the single suffixes  
 23748 appended. A rule with one suffix **.s2** is the definition of how to build *target* from **target.s2**. The  
 23749 other suffix (**.s1**) is treated as null.

23750 XSI A tilde ('~') in the above rules refers to an SCCS file in the current directory. Thus, the rule **.c~.o**  
 23751 would transform an SCCS C-language source file into an object file (**.o**). Because the **s.** of the  
 23752 SCCS files is a prefix, it is incompatible with *make*'s suffix point of view. Hence, the '~' is a way  
 23753 of changing any file reference into an SCCS file reference.

## 23754 Libraries

23755 If a target or prerequisite contains parentheses, it shall be treated as a member of an archive  
 23756 library. For the *lib(member.o)* expression *lib* refers to the name of the archive library and *member.o*  
 23757 to the member name. The application shall ensure that the member is an object file with the **.o**  
 23758 suffix. The modification time of the expression is the modification time for the member as kept  
 23759 in the archive library; see *ar* (on page 2336). The **.a** suffix shall refer to an archive library. The  
 23760 **.s2.a** rule shall be used to update a member in the library from a file with a suffix **.s2**.

23761

**Internal Macros**

23762

The *make* utility shall maintain five internal macros that can be used in target and inference rules.

23763

In order to clearly define the meaning of these macros, some clarification of the terms *target rule*,

23764

*inference rule*, *target*, and *prerequisite* is necessary.

23765

Target rules are specified by the user in a makefile for a particular target. Inference rules are

23766

user-specified or *make*-specified rules for a particular class of target name. Explicit prerequisites

23767

are those prerequisites specified in a makefile on target lines. Implicit prerequisites are those

23768

prerequisites that are generated when inference rules are used. Inference rules are applied to

23769

implicit prerequisites or to explicit prerequisites that do not have target rules defined for them in

23770

the makefile. Target rules are applied to targets specified in the makefile.

23771

Before any target in the makefile is updated, each of its prerequisites (both explicit and implicit)

23772

shall be updated. This shall be accomplished by recursively processing each prerequisite. Upon

23773

recursion, each prerequisite shall become a target itself. Its prerequisites in turn shall be

23774

processed recursively until a target is found that has no prerequisites, at which point the

23775

recursion stops. The recursion then shall back up, updating each target as it goes.

23776

In the definitions that follow, the word *target* refers to one of:

23777

- A target specified in the makefile

23778

- An explicit prerequisite specified in the makefile that becomes the target when *make* processes it during recursion

23779

23780

- An implicit prerequisite that becomes a target when *make* processes it during recursion

23781

In the definitions that follow, the word *prerequisite* refers to one of the following:

23782

- An explicit prerequisite specified in the makefile for a particular target

23783

- An implicit prerequisite generated as a result of locating an appropriate inference rule and corresponding file that matches the suffix of the target

23784

23785

The five internal macros are:

23786

**\$@** The **\$@** shall evaluate to the full target name of the current target, or the archive filename part of a library archive target. It shall be evaluated for both target and inference rules.

23787

23788

23789

For example, in the **.c.a** inference rule, **\$@** represents the out-of-date **.a** file to be built.

23790

Similarly, in a makefile target rule to build **lib.a** from **file.c**, **\$@** represents the out-of-

23791

date **lib.a**.

23792

**\$%** The **\$%** macro shall be evaluated only when the current target is an archive library member of the form *libname(member.o)*. In these cases, **\$@** shall evaluate to *libname* and **\$%** shall evaluate to *member.o*. The **\$%** macro shall be evaluated for both target and inference rules.

23793

23794

23795

23796

For example, in a makefile target rule to build **lib.a(file.o)**, **\$%** represents **file.o**, as opposed to **\$@**, which represents **lib.a**.

23797

23798

**\$?** The **\$?** macro shall evaluate to the list of prerequisites that are newer than the current target. It shall be evaluated for both target and inference rules.

23799

23800

For example, in a makefile target rule to build *prog* from **file1.o**, **file2.o**, and **file3.o**, and

23801

where *prog* is not out of date with respect to **file1.o**, but is out of date with respect to

23802

**file2.o** and **file3.o**, **\$?** represents **file2.o** and **file3.o**.

23803        \$<        In an inference rule, the \$< macro shall evaluate to the filename whose existence  
 23804                    allowed the inference rule to be chosen for the target. In the **.DEFAULT** rule, the \$<  
 23805                    macro shall evaluate to the current target name. The meaning of the \$< macro shall be  
 23806                    otherwise unspecified. |

23807                    For example, in the **.c.a** inference rule, \$< represents the prerequisite **.c** file.

23808        \$\*        The \$\* macro shall evaluate to the current target name with its suffix deleted. It shall be  
 23809                    evaluated at least for inference rules.

23810                    For example, in the **.c.a** inference rule, \$\*.o represents the out-of-date **.o** file that  
 23811                    corresponds to the prerequisite **.c** file.

23812                    Each of the internal macros has an alternative form. When an uppercase 'D' or 'F' is appended |  
 23813                    to any of the macros, the meaning shall be changed to the *directory part* for 'D' and *filename part* |  
 23814                    for 'F'. The directory part is the path prefix of the file without a trailing slash; for the current  
 23815                    directory, the directory part is '.'. When the \$? macro contains more than one prerequisite  
 23816                    filename, the \$(?D) and \${?F} (or \${?D} and \${?F}) macros expand to a list of directory name parts  
 23817                    and filename parts respectively.

23818                    For the target *lib(member.o)* and the **s2.a** rule, the internal macros shall be defined as: |

23819        \$<        *member.s2*

23820        \$\*        *member*

23821        \$@        *lib*

23822        \$?        *member.s2*

23823        \$%        *member.o*

## 23824        **Default Rules**

23825                    The default rules for *make* shall achieve results that are the same as if the following were used.  
 23826                    Implementations that do not support the C-Language Development Utilities option may omit  
 23827                    **CC**, **CFLAGS**, **YACC**, **YFLAGS**, **LEX**, **LFLAGS**, **LDFLAGS**, and the **.c**, **.y**, and **.l** inference rules.  
 23828                    Implementations that do not support FORTRAN may omit **FC**, **FFLAGS**, and the **.f** inference  
 23829                    rules. Implementations may provide additional macros and rules.

## 23830        *SPECIAL TARGETS*

23831 XSI        .SCCS\_GET: sccs \$(SCCSFLAGS) get \$(SCCSGETFLAGS) \$@

23832

23833 XSI        .SUFFIXES: .o .c .y .l .a .sh .f .c~ .y~ .l~ .sh~ .f~

## 23834        **MACROS**

23835        MAKE=make

23836        AR=ar

23837        ARFLAGS=-rv

23838        YACC=yacc

23839        YFLAGS=

23840        LEX=lex

23841        LFLAGS=

23842        LDFLAGS=

23843        CC=c99

23844        CFLAGS=-O

23845        FC=fort77

```

23846      FFLAGS=-O 1
23847 XSI   GET=get
23848      GFLAGS=
23849      SCCSFLAGS=
23850      SCCSGETFLAGS=-s
23851
23852      SINGLE SUFFIX RULES
23853      .c:
23854          $(CC) $(CFLAGS) $(LDFLAGS) -o $@ $<
23855      .f:
23856          $(FC) $(FFLAGS) $(LDFLAGS) -o $@ $<
23857      .sh:
23858          cp $< $@
23859          chmod a+x $@
23860 XSI   .c~:
23861          $(GET) $(GFLAGS) -p $< > $*.c
23862          $(CC) $(CFLAGS) $(LDFLAGS) -o $@ $*.c
23863      .f~:
23864          $(GET) $(GFLAGS) -p $< > $*.f
23865          $(FC) $(FFLAGS) $(LDFLAGS) -o $@ $*.f
23866      .sh~:
23867          $(GET) $(GFLAGS) -p $< > $*.sh
23868          cp $*.sh $@
23869          chmod a+x $@
23870
23871      DOUBLE SUFFIX RULES
23872      .c.o:
23873          $(CC) $(CFLAGS) -c $<
23874      .f.o:
23875          $(FC) $(FFLAGS) -c $<
23876      .y.o:
23877          $(YACC) $(YFLAGS) $<
23878          $(CC) $(CFLAGS) -c y.tab.c
23879          rm -f y.tab.c
23880          mv y.tab.o $@
23881      .l.o:
23882          $(LEX) $(LFLAGS) $<
23883          $(CC) $(CFLAGS) -c lex.yy.c
23884          rm -f lex.yy.c
23885          mv lex.yy.o $@
23886      .y.c:
23887          $(YACC) $(YFLAGS) $<
23888          mv y.tab.c $@
23889      .l.c:
23890          $(LEX) $(LFLAGS) $<

```

```

23891         mv lex.yy.c $@
23892 xSI     .c~.o:
23893         $(GET) $(GFLAGS) -p $< > $*.c
23894         $(CC) $(CFLAGS) -c $*.c
23895     .f~.o:
23896         $(GET) $(GFLAGS) -p $< > $*.f
23897         $(FC) $(FFLAGS) -c $*.f
23898     .y~.o:
23899         $(GET) $(GFLAGS) -p $< > $*.y
23900         $(YACC) $(YFLAGS) $*.y
23901         $(CC) $(CFLAGS) -c y.tab.c
23902         rm -f y.tab.c
23903         mv y.tab.o $@
23904     .l~.o:
23905         $(GET) $(GFLAGS) -p $< > $*.l
23906         $(LEX) $(LFLAGS) $*.l
23907         $(CC) $(CFLAGS) -c lex.yy.c
23908         rm -f lex.yy.c
23909         mv lex.yy.o $@
23910     .y~.c:
23911         $(GET) $(GFLAGS) -p $< > $*.y
23912         $(YACC) $(YFLAGS) $*.y
23913         mv y.tab.c $@
23914     .l~.c:
23915         $(GET) $(GFLAGS) -p $< > $*.l
23916         $(LEX) $(LFLAGS) $*.l
23917         mv lex.yy.c $@
23918
23919     .c.a:
23920         $(CC) -c $(CFLAGS) $<
23921         $(AR) $(ARFLAGS) $@ $*.o
23922         rm -f $*.o
23923     .f.a:
23924         $(FC) -c $(FFLAGS) $<
23925         $(AR) $(ARFLAGS) $@ $*.o
23926         rm -f $*.o
23927 EXIT STATUS
23928     When the -q option is specified, the make utility shall exit with one of the following values:
23929         0 Successful completion.
23930         1 The target was not up-to-date.
23931     >1 An error occurred.
23932     When the -q option is not specified, the make utility shall exit with one of the following values:
23933         0 Successful completion.
23934         >0 An error occurred.

```

## 23935 CONSEQUENCES OF ERRORS

23936 Default.

## 23937 APPLICATION USAGE

23938 If there is a source file (such as *./source.c*) and there are two SCCS files corresponding to it  
 23939 (*./s.source.c* and *./SCCS/s.source.c*), on XSI-conformant systems *make* uses the SCCS file in the  
 23940 current directory. However, users are advised to use the underlying SCCS utilities (*admin*, *delta*,  
 23941 *get*, and so on) or the *sccs* utility for all source files in a given directory. If both forms are used for  
 23942 a given source file, future developers are very likely to be confused.

23943 It is incumbent upon portable makefiles to specify the **.POSIX** special target in order to  
 23944 guarantee that they are not affected by local extensions.

23945 The **-k** and **-S** options are both present so that the relationship between the command line, the  
 23946 **MAKEFLAGS** variable, and the makefile can be controlled precisely. If the **k** flag is passed in  
 23947 **MAKEFLAGS** and a command is of the form:

23948 `$(MAKE) -S foo`23949 then the default behavior is restored for the child *make*.

23950 When the **-n** option is specified, it is always added to **MAKEFLAGS**. This allows a recursive  
 23951 *make -n target* to be used to see all of the action that would be taken to update *target*.

23952 Because of widespread historical practice, interpreting a '#' number sign inside a variable as  
 23953 the start of a comment has the unfortunate side effect of making it impossible to place a number  
 23954 sign in a variable, thus forbidding something like:

23955 `CFLAGS = "-D COMMENT_CHAR='#'"`

23956 Many historical *make* utilities stop chaining together inference rules when an intermediate target  
 23957 is nonexistent. For example, it might be possible for a *make* to determine that both *.y.c* and *.c.o*  
 23958 could be used to convert a *.y* to a *.o*. Instead, in this case, *make* requires the use of a *.y.o* rule.

23959 The best way to provide portable makefiles is to include all of the rules needed in the makefile  
 23960 itself. The rules provided use only features provided by other parts of this volume of  
 23961 IEEE Std 1003.1-200x. The default rules include rules for optional commands in this volume of  
 23962 IEEE Std 1003.1-200x. Only rules pertaining to commands that are provided are needed in an  
 23963 implementation's default set.

23964 Macros used within other macros are evaluated when the new macro is used rather than when  
 23965 the new macro is defined. Therefore:

```
23966 MACRO = value1
23967 NEW   = $(MACRO)
23968 MACRO = value2
```

```
23969 target:
23970     echo $(NEW)
```

23971 would produce *value2* and not *value1* since **NEW** was not expanded until it was needed in the  
 23972 *echo* command line.

23973 Some historical applications have been known to intermix *target\_name* and *macro=name* operands  
 23974 on the command line, expecting that all of the macros are processed before any of the targets are  
 23975 dealt with. Conforming applications do not do this, although some backward compatibility  
 23976 support may be included in some implementations.

23977 The following characters in filenames may give trouble: '=', ':', '\', '\'', and '@'. For  
 23978 inference rules, the description of \$< and \$? seem similar. However, an example shows the

23979 minor difference. In a makefile containing:

23980 `foo.o: foo.h`

23981 if **foo.h** is newer than **foo.o**, yet **foo.c** is older than **foo.o**, the built-in rule to make **foo.o** from  
 23982 **foo.c** is used, with `$<` equal to **foo.c** and `$?` equal to **foo.h**. If **foo.c** is also newer than **foo.o**, `$<` is  
 23983 equal to **foo.c** and `$?` is equal to **foo.h foo.c**.

#### 23984 EXAMPLES

23985 1. The following command:

23986 `make`

23987 makes the first target found in the makefile.

23988 2. The following command:

23989 `make junk`

23990 makes the target **junk**.

23991 3. The following makefile says that **pgm** depends on two files, **a.o** and **b.o**, and that they in  
 23992 turn depend on their corresponding source files (**a.c** and **b.c**), and a common file **incl.h**:

```
23993 pgm: a.o b.o
23994     c99 a.o b.o -o pgm
23995 a.o: incl.h a.c
23996     c99 -c a.c
23997 b.o: incl.h b.c
23998     c99 -c b.c
```

23999 4. An example for making optimized **.o** files from **.c** files is:

```
24000 .c.o:
24001     c99 -c -O $*.c
```

24002 or:

```
24003 .c.o:
24004     c99 -c -O $<
```

24005 5. The most common use of the archive interface follows. Here, it is assumed that the source  
 24006 files are all C-language source:

```
24007 lib: lib(file1.o) lib(file2.o) lib(file3.o)
24008     @echo lib is now up-to-date
```

24009 The **.c.a** rule is used to make **file1.o**, **file2.o**, and **file3.o** and insert them into **lib**.

24010 The treatment of escaped `<newline>`s throughout the makefile is historical practice. For  
 24011 example, the inference rule:

```
24012 .c.o\
24013 :
```

24014 works, and the macro:

```
24015 f= bar baz\
24016     biz
24017 a:
24018     echo ==$f==
```

```

24019      echoes "==bar baz biz==".
24020      If $? were:
24021      /usr/include/stdio.h /usr/include/unistd.h foo.h
24022      then $(?D) would be:
24023      /usr/include /usr/include .
24024      and $(?F) would be:
24025      stdio.h unistd.h foo.h
24026      6. The contents of the built-in rules can be viewed by running:
24027      make -p -f /dev/null 2>/dev/null

```

#### 24028 RATIONALE

24029 The *make* utility described in this volume of IEEE Std 1003.1-200x is intended to provide the  
 24030 means for changing portable source code into executables that can be run on a  
 24031 IEEE Std 1003.1-200x-conforming system. It reflects the most common features present in  
 24032 System V and BSD *makes*.

24033 Historically, the *make* utility has been an especially fertile ground for vendor and research  
 24034 organization-specific syntax modifications and extensions. Examples include:

- 24035 • Syntax supporting parallel execution (such as from various multi-processor vendors, GNU,  
 24036 and others)
- 24037 • Additional “operators” separating targets and their prerequisites (System V, BSD, and  
 24038 others)
- 24039 • Specifying that command lines containing the strings “\${MAKE}” and “\$(MAKE)” are  
 24040 executed when the `-n` option is specified (GNU and System V)
- 24041 • Modifications of the meaning of internal macros when referencing libraries (BSD and others)
- 24042 • Using a single instance of the shell for all of the command lines of the target (BSD and others)
- 24043 • Allowing spaces as well as tabs to delimit command lines (BSD)
- 24044 • Adding C preprocessor-style “include” and “ifdef” constructs (System V, GNU, BSD, and  
 24045 others)
- 24046 • Remote execution of command lines (Sprite and others)
- 24047 • Specifying additional special targets (BSD, System V, and most others)

24048 Additionally, many vendors and research organizations have rethought the basic concepts of  
 24049 *make*, creating vastly extended, as well as completely new, syntaxes. Each of these versions of  
 24050 *make* fulfills the needs of a different community of users; it is unreasonable for this volume of  
 24051 IEEE Std 1003.1-200x to require behavior that would be incompatible (and probably inferior) to  
 24052 historical practice for such a community.

24053 In similar circumstances, when the industry has enough sufficiently incompatible formats as to  
 24054 make them irreconcilable, this volume of IEEE Std 1003.1-200x has followed one or both of two  
 24055 courses of action. Commands have been renamed (*cksum*, *echo*, and *pax*) and/or command line  
 24056 options have been provided to select the desired behavior (*grep*, *od*, and *pax*).

24057 Because the syntax specified for the *make* utility is, by and large, a subset of the syntaxes  
 24058 accepted by almost all versions of *make*, it was decided that it would be counter-productive to  
 24059 change the name. And since the makefile itself is a basic unit of portability, it would not be

24060 completely effective to reserve a new option letter, such as *make -P*, to achieve the portable  
24061 behavior. Therefore, the special target *.POSIX* was added to the makefile, allowing users to  
24062 specify “standard” behavior. This special target does not preclude extensions in the *make* utility,  
24063 nor does it preclude such extensions being used by the makefile specifying the target; it does,  
24064 however, preclude any extensions from being applied that could alter the behavior of previously  
24065 valid syntax; such extensions must be controlled via command line options or new special  
24066 targets. It is incumbent upon portable makefiles to specify the *.POSIX* special target in order to  
24067 guarantee that they are not affected by local extensions.

24068 The portable version of *make* described in this reference page is not intended to be the state-of-  
24069 the-art software generation tool and, as such, some newer and more leading-edge features have  
24070 not been included. An attempt has been made to describe the portable makefile in a manner that  
24071 does not preclude such extensions as long as they do not disturb the portable behavior described  
24072 here.

24073 When the *-n* option is specified, it is always added to *MAKEFLAGS*. This allows a recursive  
24074 *make -n target* to be used to see all of the action that would be taken to update *target*.

24075 The definition of *MAKEFLAGS* allows both the System V letter string and the BSD command line  
24076 formats. The two formats are sufficiently different to allow implementations to support both  
24077 without ambiguity.

24078 Early proposals stated that an “unquoted” number sign was treated as the start of a comment.  
24079 The *make* utility does not pay any attention to quotes. A number sign starts a comment  
24080 regardless of its surroundings.

24081 The text about “other implementation-defined pathnames may also be tried” in addition to  
24082 *./makefile* and *./Makefile* is to allow such extensions as *SCCS/s.Makefile* and other variations.  
24083 It was made an implementation-defined requirement (as opposed to unspecified behavior) to  
24084 highlight surprising implementations that might select something unexpected like  
24085 */etc/Makefile*. XSI-conformant systems also try *./s.makefile*, *SCCS/s.makefile*, *./s.Makefile*,  
24086 and *SCCS/s.Makefile*.

24087 Early proposals contained the macro *NPROC* as a means of specifying that *make* should use *n*  
24088 processes to do the work required. While this feature is a valuable extension for many systems, it  
24089 is not common usage and could require other non-trivial extensions to makefile syntax. This  
24090 extension is not required by this volume of IEEE Std 1003.1-200x, but could be provided as a  
24091 compatible extension. The macro *PARALLEL* is used by some historical systems with essentially  
24092 the same meaning (but without using a name that is a common system limit value). It is  
24093 suggested that implementors recognize the existing use of *NPROC* and/or *PARALLEL* as  
24094 extensions to *make*.

24095 The default rules are based on System V. The default *CC=* value is *c99* instead of *cc* because this  
24096 volume of IEEE Std 1003.1-200x does not standardize the utility named *cc*. Thus, every  
24097 conforming application would be required to define *CC=c99* to expect to run. There is no  
24098 advantage conferred by the hope that the makefile might hit the “preferred” compiler because  
24099 this cannot be guaranteed to work. Also, since the portable makescript can only use the *c99*  
24100 options, no advantage is conferred in terms of what the script can do. It is a quality-of-  
24101 implementation issue as to whether *c99* is as valuable as *cc*.

24102 The *-d* option to *make* is frequently used to produce debugging information, but is too  
24103 implementation-defined to add to this volume of IEEE Std 1003.1-200x.

24104 The *-p* option is not passed in *MAKEFLAGS* on most historical implementations and to change  
24105 this would cause many implementations to break without sufficiently increased portability.

24106 Commands that begin with a plus sign ('+') are executed even if the `-n` option is present. Based  
 24107 on the GNU version of *make*, the behavior of `-n` when the plus-sign prefix is encountered has  
 24108 been extended to apply to `-q` and `-t` as well. However, the System V convention of forcing  
 24109 command execution with `-n` when the command line of a target contains either of the strings  
 24110 "\$`(MAKE)`" or "\$`{MAKE}`" has not been adopted. This functionality appeared in early  
 24111 proposals, but the danger of this approach was pointed out with the following example of a  
 24112 portion of a makefile:

```
24113     subdir:
24114         cd subdir; rm all_the_files; $(MAKE)
```

24115 The loss of the System V behavior in this case is well-balanced by the safety afforded to other  
 24116 makefiles that were not aware of this situation. In any event, the command line plus-sign prefix  
 24117 can provide the desired functionality.

24118 The double colon in the target rule format is supported in BSD systems to allow more than one  
 24119 target line containing the same target name to have commands associated with it. Since this is  
 24120 not functionality described in the SVID or XPG3 it has been allowed as an extension, but not  
 24121 mandated.

24122 The default rules are provided with text specifying that the built-in rules shall be the same *as if*  
 24123 the listed set were used. The intent is that implementations should be able to use the rules  
 24124 without change, but will be allowed to alter them in ways that do not affect the primary  
 24125 behavior.

24126 The best way to provide portable makefiles is to include all of the rules needed in the makefile  
 24127 itself. The rules provided use only features provided by other portions of this volume of  
 24128 IEEE Std 1003.1-200x. The default rules include rules for optional commands in this volume of  
 24129 IEEE Std 1003.1-200x. Only rules pertaining to commands that are provided are needed in the  
 24130 default set of an implementation.

24131 One point of discussion was whether to drop the default rules list from this volume of  
 24132 IEEE Std 1003.1-200x. They provide convenience, but do not enhance portability of applications.  
 24133 The prime benefit is in portability of users who wish to type *make command* and have the  
 24134 command build from a **command.c** file.

24135 The historical *MAKESHELL* feature was omitted. In some implementations it is used to let a user  
 24136 override the shell to be used to run *make* commands. This was confusing; for a portable *make*,  
 24137 the shell should be chosen by the makefile writer or specified on the *make* command line and not by  
 24138 a user running *make*.

24139 The *make* utilities in most historical implementations process the prerequisites of a target in left-  
 24140 to-right order, and the makefile format requires this. It supports the standard idiom used in  
 24141 many makefiles that produce *yacc* programs; for example:

```
24142     foo: y.tab.o lex.o main.o
24143         $(CC) $(CFLAGS) -o $@ t.tab.o lex.o main.o
```

24144 In this example, if *make* chose any arbitrary order, the **lex.o** might not be made with the correct  
 24145 **y.tab.h**. Although there may be better ways to express this relationship, it is widely used  
 24146 historically. Implementations that desire to update prerequisites in parallel should require an  
 24147 explicit extension to *make* or the makefile format to accomplish it, as described previously.

24148 The algorithm for determining a new entry for target rules is partially unspecified. Some  
 24149 historical *makes* allow blank, empty, or comment lines within the collection of commands  
 24150 marked by leading `<tab>s`. A conforming makefile must ensure that each command starts with  
 24151 a `<tab>`, but implementations are free to ignore blank, empty, and comment lines without  
 24152 triggering the start of a new entry.

24153 The ASYNCHRONOUS EVENTS section includes having SIGTERM and SIGHUP, along with  
 24154 the more traditional SIGINT and SIGQUIT, remove the current target unless directed not to do  
 24155 so. SIGTERM and SIGHUP were added to parallel other utilities that have historically cleaned  
 24156 up their work as a result of these signals. When *make* receives any signal other than SIGQUIT, it  
 24157 is required to resend itself the signal it received so that it exits with a status that reflects the  
 24158 signal. The results from SIGQUIT are partially unspecified because, on systems that create **core**  
 24159 files upon receipt of SIGQUIT, the **core** from *make* would conflict with a core file from the  
 24160 command that was running when the SIGQUIT arrived. The main concern was to prevent  
 24161 damaged files from appearing up-to-date when *make* is rerun.

24162 The **.PRECIOUS** special target was extended to affect all targets globally (by specifying no  
 24163 prerequisites). The **.IGNORE** and **.SILENT** special targets were extended to allow prerequisites;  
 24164 it was judged to be more useful in some cases to be able to turn off errors or echoing for a list  
 24165 of targets than for the entire makefile. These extensions to the *make* in System V were made to  
 24166 match historical practice from the BSD *make*.

24167 Macros are not exported to the environment of commands to be run. This was never the case in  
 24168 any historical *make* and would have serious consequences. The environment is the same as the  
 24169 environment to *make* except that **MAKEFLAGS** and macros defined on the *make* command line  
 24170 are added.

24171 Some implementations do not use *system()* for all command lines, as required by the portable  
 24172 makefile format; as a performance enhancement, they select lines without shell metacharacters  
 24173 for direct execution by *execve()*. There is no requirement that *system()* be used specifically, but  
 24174 merely that the same results be achieved. The metacharacters typically used to bypass the direct  
 24175 *execve()* execution have been any of:

24176 = | ^ ( ) ; & < > \* ? [ ] : \$ \ ' " \ \n

24177 The default in some advanced versions of *make* is to group all the command lines for a target and  
 24178 execute them using a single shell invocation; the System V method is to pass each line  
 24179 individually to a separate shell. The single-shell method has the advantages in performance and  
 24180 the lack of a requirement for many continued lines. However, converting to this newer method  
 24181 has caused portability problems with many historical makefiles, so the behavior with the POSIX  
 24182 makefile is specified to be the same as that of System V. It is suggested that the special target  
 24183 **.ONESHELL** be used as an implementation extension to achieve the single-shell grouping for a  
 24184 target or group of targets.

24185 Novice users of *make* have had difficulty with the historical need to start commands with a  
 24186 <tab>. Since it is often difficult to discern differences between <tab>s and <space>s on terminals  
 24187 or printed listings, confusing bugs can arise. In early proposals, an attempt was made to correct  
 24188 this problem by allowing leading <blank>s instead of <tab>s. However, implementors reported  
 24189 many makefiles that failed in subtle ways following this change, and it is difficult to implement  
 24190 a *make* that unambiguously can differentiate between macro and command lines. There is  
 24191 extensive historical practice of allowing leading spaces before macro definitions. Forcing macro  
 24192 lines into column 1 would be a significant backwards-compatibility problem for some makefiles.  
 24193 Therefore, historical practice was restored.

24194 The System V **INCLUDE** feature was considered, but not included. This would treat a line that  
 24195 began in the first column and contained **INCLUDE** <filename> as an indication to read <filename>  
 24196 at that point in the makefile. This is difficult to use in a portable way, and it raises concerns  
 24197 about nesting levels and diagnostics. System V, BSD, GNU, and others have used different  
 24198 methods for including files.

24199 The System V dynamic dependency feature was not included. It would support:

24200 cat: \$\$@.c

24201 that would expand to;

24202 cat: cat.c

24203 This feature exists only in the new version of System V *make* and, while useful, is not in wide  
24204 usage. This means that macros are expanded twice for prerequisites: once at makefile parse time  
24205 and once at target update time.

24206 Consideration was given to adding metarules to the POSIX *make*. This would make `%o: %c` the  
24207 same as `.c.o:`. This is quite useful and available from some vendors, but it would cause too many  
24208 changes to this *make* to support. It would have introduced rule chaining and new substitution  
24209 rules. However, the rules for target names have been set to reserve the `'%'` and `'"'` characters.  
24210 These are traditionally used to implement metarules and quoting of target names, respectively.  
24211 Implementors are strongly encouraged to use these characters only for these purposes.

24212 A request was made to extend the suffix delimiter character from a period to any character. The  
24213 metarules feature in newer *makes* solves this problem in a more general way. This volume of  
24214 IEEE Std 1003.1-200x is staying with the more conservative historical definition.

24215 The standard output format for the `-p` option is not described because it is primarily a  
24216 debugging option and because the format is not generally useful to programs. In historical  
24217 implementations the output is not suitable for use in generating makefiles. The `-p` format has  
24218 been variable across historical implementations. Therefore, the definition of `-p` was only to  
24219 provide a consistently named option for obtaining *make* script debugging information.

24220 Some historical implementations have not cleared the suffix list with `-r`.

24221 Implementations should be aware that some historical applications have intermixed *target\_name*  
24222 and *macro=value* operands on the command line, expecting that all of the macros are processed |  
24223 before any of the targets are dealt with. Conforming applications do not do this, but some |  
24224 backwards-compatibility support may be warranted.

24225 Empty inference rules are specified with a semicolon command rather than omitting all  
24226 commands, as described in an early proposal. The latter case has no traditional meaning and is  
24227 reserved for implementation extensions, such as in GNU *make*.

#### 24228 FUTURE DIRECTIONS

24229 None.

#### 24230 SEE ALSO

24231 *ar*, *c99*, *get*, *lex*, *sh*, *yacc*, the System Interfaces volume of IEEE Std 1003.1-200x, *system()*

#### 24232 CHANGE HISTORY

24233 First released in Issue 2.

#### 24234 Issue 5

24235 FUTURE DIRECTIONS section added.

#### 24236 Issue 6

24237 This utility is now marked as part of the Software Development Utilities option.

24238 The Open Group Corrigendum U029/1 is applied, correcting a typographical error in the  
24239 SPECIAL TARGETS section.

24240 In the ENVIRONMENT VARIABLES section, the *PROJECTDIR* description is updated from  
24241 “otherwise, the home directory of a user of that name is examined” to “otherwise, the value of  
24242 *PROJECTDIR* is treated as a user name and that user’s initial working directory is examined”.

- 24243 It is specified whether the command line is related to the makefile or to the *make* command, and  
24244 the macro processing rules are updated to align with the IEEE P1003.2b draft standard.
- 24245 The normative text is reworded to avoid use of the term “must” for application requirements.
- 24246 PASC Interpretation 1003.2 #193 is applied.

24247 **NAME**

24248 man — display system documentation

24249 **SYNOPSIS**24250 man [-k] *name* . . .24251 **DESCRIPTION**

24252 The *man* utility shall write information about each of the *name* operands. If *name* is the name of a  
 24253 standard utility, *man* at a minimum shall write a message describing the syntax used by the  
 24254 standard utility, its options, and operands. If more information is available, the *man* utility shall  
 24255 provide it in an implementation-defined manner.

24256 An implementation may provide information for values of *name* other than the standard utilities.  
 24257 Standard utilities that are listed as optional and that are not supported by the implementation  
 24258 either shall cause a brief message indicating that fact to be displayed or shall cause a full display  
 24259 of information as described previously.

24260 **OPTIONS**

24261 The *man* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section  
 24262 12.2, Utility Syntax Guidelines.

24263 The following option shall be supported:

24264 **-k** Interpret *name* operands as keywords to be used in searching a utilities summary  
 24265 database that contains a brief purpose entry for each standard utility and write lines  
 24266 from the summary database that match any of the keywords. The keyword search shall  
 24267 produce results that are the equivalent of the output of the following command:

```
24268     grep -Ei '
24269     name
24270     name
24271     . . .
24272     ' summary-database
```

24273 This assumes that the *summary-database* is a text file with a single entry per line; this  
 24274 organization is not required and the example using *grep -Ei* is merely illustrative of the  
 24275 type of search intended. The purpose entry to be included in the database shall consist  
 24276 of a terse description of the purpose of the utility.

24277 **OPERANDS**

24278 The following operand shall be supported:

24279 *name* A keyword or the name of a standard utility. When **-k** is not specified and *name*  
 24280 does not represent one of the standard utilities, the results are unspecified.

24281 **STDIN**

24282 Not used.

24283 **INPUT FILES**

24284 None.

24285 **ENVIRONMENT VARIABLES**24286 The following environment variables shall affect the execution of *man*:

24287 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 24288 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
 24289 Internationalization Variables for the precedence of internationalization variables  
 24290 used to determine the values of locale categories.)

- 24291 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
24292 internationalization variables.
- 24293 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
24294 characters (for example, single-byte as opposed to multi-byte characters in  
24295 arguments and in the summary database). The value of *LC\_CTYPE* need not affect  
24296 the format of the information written about the name operands.
- 24297 **LC\_MESSAGES**  
24298 Determine the locale that should be used to affect the format and contents of  
24299 diagnostic messages written to standard error and informative messages written to  
24300 standard output.
- 24301 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 24302 **PAGER** Determine an output filtering command for writing the output to a terminal. Any  
24303 string acceptable as a *command\_string* operand to the *sh -c* command shall be valid.  
24304 When standard output is a terminal device, the reference page output shall be  
24305 piped through the command. If the *PAGER* variable is null or not set, the  
24306 command shall be either *more* or another paginator utility documented in the  
24307 system documentation.
- 24308 **ASYNCHRONOUS EVENTS**
- 24309 Default.
- 24310 **STDOUT**  
24311 The *man* utility shall write text describing the syntax of the utility *name*, its options and its  
24312 operands, or, when *-k* is specified, lines from the summary database. The format of this text is  
24313 implementation-defined.
- 24314 **STDERR**  
24315 The standard error shall be used only for diagnostic messages.
- 24316 **OUTPUT FILES**  
24317 None.
- 24318 **EXTENDED DESCRIPTION**  
24319 None.
- 24320 **EXIT STATUS**  
24321 The following exit values shall be returned:  
24322 0 Successful completion.  
24323 >0 An error occurred.
- 24324 **CONSEQUENCES OF ERRORS**  
24325 Default.
- 24326 **APPLICATION USAGE**  
24327 None.
- 24328 **EXAMPLES**  
24329 None.
- 24330 **RATIONALE**  
24331 It is recognized that the *man* utility is only of minimal usefulness as specified. The opinion of the  
24332 standard developers was strongly divided as to how much or how little information *man* should  
24333 be required to provide. They considered, however, that the provision of some portable way of  
24334 accessing documentation would aid user portability. The arguments against a fuller

24335 specification were:

- 24336 • Large quantities of documentation should not be required on a system that does not have
- 24337 excess disk space.
- 24338 • The current manual system does not present information in a manner that greatly aids user
- 24339 portability.
- 24340 • A “better help system” is currently an area in which vendors feel that they can add value to
- 24341 their POSIX implementations.

24342 The `-f` option was considered, but due to implementation differences, it was not included in this

24343 volume of IEEE Std 1003.1-200x.

24344 The description was changed to be more specific about what has to be displayed for a utility.

24345 The standard developers considered it insufficient to allow a display of only the synopsis

24346 without giving a short description of what each option and operand does.

24347 The “purpose” entry to be included in the database can be similar to the section title (less the

24348 numeric prefix) from this volume of IEEE Std 1003.1-200x for each utility. These titles are similar

24349 to those used in historical systems for this purpose.

24350 See *mailx* for rationale concerning the default paginator.

24351 The caveat in the *LC\_CTYPE* description was added because it is not a requirement that an

24352 implementation provide reference pages for all of its supported locales on each system;

24353 changing *LC\_CTYPE* does not necessarily translate the reference page into another language.

24354 This is equivalent to the current state of *LC\_MESSAGES* in IEEE Std 1003.1-200x—locale-specific

24355 messages are not yet a requirement.

24356 The historical *MANPATH* variable is not included in POSIX because no attempt is made to

24357 specify naming conventions for reference page files, nor even to mandate that they are files at |

24358 all. On some implementations they could be a true database, a hypertext file, or even fixed |

24359 strings within the *man* executable. The standard developers considered the portability of |

24360 reference pages to be outside their scope of work. However, users should be aware that

24361 *MANPATH* is implemented on a number of historical systems and that it can be used to tailor

24362 the search pattern for reference pages from the various categories (utilities, functions, file

24363 formats, and so on) when the system administrator reveals the location and conventions for

24364 reference pages on the system.

24365 The keyword search can rely on at least the text of the section titles from these utility

24366 descriptions, and the implementation may add more keywords. The term “section titles” refers

24367 to the strings such as:

24368 `man` – Display system documentation

24369 `ps` – Report process status

24370 **FUTURE DIRECTIONS**

24371 None.

24372 **SEE ALSO**

24373 *more*

24374 **CHANGE HISTORY**

24375 First released in Issue 4.

24376 **Issue 5**

24377 FUTURE DIRECTIONS section added.

24378 **NAME**

24379 mesg — permit or deny messages

24380 **SYNOPSIS**

24381 UP mesg [y|n]

24382

24383 **DESCRIPTION**

24384 The *mesg* utility shall control whether other users are allowed to send messages via *write*, *talk*, or  
24385 other utilities to a terminal device. The terminal device affected shall be determined by searching  
24386 for the first terminal in the sequence of devices associated with standard input, standard output,  
24387 and standard error, respectively. With no arguments, *mesg* shall report the current state without  
24388 changing it. Processes with appropriate privileges may be able to send messages to the terminal  
24389 independent of the current state.

24390 **OPTIONS**

24391 None.

24392 **OPERANDS**

24393 The following operands shall be supported in the POSIX locale:

24394 y Grant permission to other users to send messages to the terminal device.

24395 n Deny permission to other users to send messages to the terminal device.

24396 **STDIN**

24397 Not used.

24398 **INPUT FILES**

24399 None.

24400 **ENVIRONMENT VARIABLES**24401 The following environment variables shall affect the execution of *mesg*:

24402 *LANG* Provide a default value for the internationalization variables that are unset or null.  
24403 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
24404 Internationalization Variables for the precedence of internationalization variables  
24405 used to determine the values of locale categories.)

24406 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
24407 internationalization variables.

24408 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
24409 characters (for example, single-byte as opposed to multi-byte characters in  
24410 arguments).

24411 *LC\_MESSAGES*

24412 Determine the locale that should be used to affect the format and contents of  
24413 diagnostic messages written (by *mesg*) to standard error.

24414 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.24415 **ASYNCHRONOUS EVENTS**

24416 Default.

24417 **STDOUT**24418 If no operand is specified, *mesg* shall display the current terminal state in an unspecified format.

24419 **STDERR**

24420 The standard error shall be used only for diagnostic messages.

24421 **OUTPUT FILES**

24422 None.

24423 **EXTENDED DESCRIPTION**

24424 None.

24425 **EXIT STATUS**

24426 The following exit values shall be returned:

24427 0 Receiving messages is allowed.

24428 1 Receiving messages is not allowed.

24429 >1 An error occurred.

24430 **CONSEQUENCES OF ERRORS**

24431 Default.

24432 **APPLICATION USAGE**

24433 The mechanism by which the message status of the terminal is changed is unspecified.  
24434 Therefore, unspecified actions may cause the status of the terminal to change after *mesg* has  
24435 successfully completed. These actions may include, but are not limited to: another invocation of  
24436 the *mesg* utility, login procedures; invocation of the *stty* utility, invocation of the *chmod* utility or  
24437 *chmod()* function, and so on.

24438 **EXAMPLES**

24439 None.

24440 **RATIONALE**

24441 The terminal changed by *mesg* is that associated with the standard input, output, or error, rather  
24442 than the controlling terminal for the session. This is because users logged in more than once  
24443 should be able to change any of their login terminals without having to stop the job running in  
24444 those sessions. This is not a security problem involving the terminals of other users because  
24445 appropriate privileges would be required to affect the terminal of another user.

24446 The method of checking each of the first three file descriptors in sequence until a terminal is  
24447 found was adopted from System V.

24448 The file */dev/tty* is not specified for the terminal device because it was thought to be too  
24449 restrictive. Typical environment changes for the *n* operand are that write permissions are  
24450 removed for *others* and *group* from the appropriate device. It was decided to leave the actual  
24451 description of what is done as unspecified because of potential differences between  
24452 implementations.

24453 The format for standard output is unspecified because of differences between historical  
24454 implementations. This output is generally not useful to shell scripts (they can use the exit  
24455 status), so exact parsing of the output is unnecessary.

24456 **FUTURE DIRECTIONS**

24457 None.

24458 **SEE ALSO**

24459 *talk*, *write*

24460 **CHANGE HISTORY**

24461 First released in Issue 2.

24462 **Issue 6**

24463 This utility is now marked as part of the User Portability Utilities option.

24464 **NAME**

24465           mkdir — make directories

24466 **SYNOPSIS**24467           mkdir [-p][-m *mode*] *dir*...24468 **DESCRIPTION**24469           The *mkdir* utility shall create the directories specified by the operands, in the order specified.24470           For each *dir* operand, the *mkdir* utility shall perform actions equivalent to the *mkdir()* function  
24471           defined in the System Interfaces volume of IEEE Std 1003.1-200x, called with the following  
24472           arguments:

- 24473           1. The *dir* operand is used as the *path* argument.
- 24474           2. The value of the bitwise-inclusive OR of S\_IRWXU, S\_IRWXG, and S\_IRWXO is used as  
24475           the *mode* argument. (If the **-m** option is specified, the *mode* option-argument overrides this  
24476           default.)

24477 **OPTIONS**24478           The *mkdir* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section  
24479           12.2, Utility Syntax Guidelines.

24480           The following options shall be supported:

24481           **-m mode**     Set the file permission bits of the newly-created directory to the specified *mode*  
24482           value. The *mode* option-argument shall be the same as the *mode* operand defined  
24483           for the *chmod* utility. In the *symbolic\_mode* strings, the *op* characters '+' and '-'  
24484           shall be interpreted relative to an assumed initial mode of *a=rwx*; '+' shall add  
24485           permissions to the default mode, '-' shall delete permissions from the default  
24486           mode.

24487           **-p**           Create any missing intermediate pathname components.

24488           For each *dir* operand that does not name an existing directory, effects equivalent to  
24489           those caused by the following command shall occur:

```
24490           mkdir -p -m $(umask -S),u+wX $(dirname dir) &&
24491           mkdir [-m mode] dir
```

24492           where the **-m mode** option represents that option supplied to the original  
24493           invocation of *mkdir*, if any.

24494           Each *dir* operand that names an existing directory shall be ignored without error.

24495 **OPERANDS**

24496           The following operand shall be supported:

24497           *dir*           A pathname of a directory to be created.

24498 **STDIN**

24499           Not used.

24500 **INPUT FILES**

24501           None.

24502 **ENVIRONMENT VARIABLES**24503           The following environment variables shall affect the execution of *mkdir*:

24504           **LANG**         Provide a default value for the internationalization variables that are unset or null.  
24505           (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
24506           Internationalization Variables for the precedence of internationalization variables

- 24507 used to determine the values of locale categories.)
- 24508 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
24509 internationalization variables.
- 24510 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
24511 characters (for example, single-byte as opposed to multi-byte characters in  
24512 arguments).
- 24513 *LC\_MESSAGES*  
24514 Determine the locale that should be used to affect the format and contents of  
24515 diagnostic messages written to standard error.
- 24516 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 24517 **ASYNCHRONOUS EVENTS**
- 24518 Default.
- 24519 **STDOUT**
- 24520 Not used.
- 24521 **STDERR**
- 24522 The standard error shall be used only for diagnostic messages.
- 24523 **OUTPUT FILES**
- 24524 None.
- 24525 **EXTENDED DESCRIPTION**
- 24526 None.
- 24527 **EXIT STATUS**
- 24528 The following exit values shall be returned:
- 24529 0 All the specified directories were created successfully or the **-p** option was specified and all  
24530 the specified directories now exist.
- 24531 >0 An error occurred.
- 24532 **CONSEQUENCES OF ERRORS**
- 24533 Default.
- 24534 **APPLICATION USAGE**
- 24535 The default file mode for directories is *a=rwx* (777 on most systems) with selected permissions  
24536 removed in accordance with the file mode creation mask. For intermediate pathname  
24537 components created by *mkdir*, the mode is the default modified by *u+wx* so that the  
24538 subdirectories can always be created regardless of the file mode creation mask; if different  
24539 ultimate permissions are desired for the intermediate directories, they can be changed  
24540 afterwards with *chmod*.
- 24541 Note that some of the requested directories may have been created even if an error occurs.
- 24542 **EXAMPLES**
- 24543 None.
- 24544 **RATIONALE**
- 24545 The System V **-m** option was included to control the file mode.
- 24546 The System V **-p** option was included to create any needed intermediate directories and to  
24547 complement the functionality provided by *rmdir* for removing directories in the path prefix as  
24548 they become empty. Because no error is produced if any path component already exists, the **-p**  
24549 option is also useful to ensure that a particular directory exists.

24550 The functionality of *mkdir* is described substantially through a reference to the *mkdir()* function  
24551 in the System Interfaces volume of IEEE Std 1003.1-200x. For example, by default, the mode of  
24552 the directory is affected by the file mode creation mask in accordance with the specified  
24553 behavior of the *mkdir()* function. In this way, there is less duplication of effort required for  
24554 describing details of the directory creation.

24555 **FUTURE DIRECTIONS**

24556 None.

24557 **SEE ALSO**

24558 *rm*, *rmdir*, *umask*, the System Interfaces volume of IEEE Std 1003.1-200x, *mkdir()*

24559 **CHANGE HISTORY**

24560 First released in Issue 2.

24561 **Issue 5**

24562 FUTURE DIRECTIONS section added.

## 24563 NAME

24564 mkfifo — make FIFO special files

## 24565 SYNOPSIS

24566 mkfifo [-m *mode*] *file*...

## 24567 DESCRIPTION

24568 The *mkfifo* utility shall create the FIFO special files specified by the operands, in the order  
24569 specified.24570 For each *file* operand, the *mkfifo* utility shall perform actions equivalent to the *mkfifo*() function  
24571 defined in the System Interfaces volume of IEEE Std 1003.1-200x, called with the following  
24572 arguments:

- 24573 1. The *file* operand is used as the *path* argument.
- 24574 2. The value of the bitwise-inclusive OR of S\_IRUSR, S\_IWUSR, S\_IRGRP, S\_IWGRP,  
24575 S\_IROTH, and S\_IWOTH is used as the *mode* argument. (If the **-m** option is specified, the  
24576 value of the *mkfifo*() *mode* argument is unspecified, but the FIFO shall at no time have  
24577 permissions less restrictive than the **-m mode** option-argument.)

## 24578 OPTIONS

24579 The *mkfifo* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section  
24580 12.2, Utility Syntax Guidelines.

24581 The following option shall be supported:

24582 **-m mode** Set the file permission bits of the newly-created FIFO to the specified *mode* value.  
24583 The *mode* option-argument shall be the same as the *mode* operand defined for the  
24584 *chmod* utility. In the *symbolic\_mode* strings, the *op* characters '+' and '-' shall be  
24585 interpreted relative to an assumed initial mode of *a=rw*.

## 24586 OPERANDS

24587 The following operand shall be supported:

24588 *file* A pathname of the FIFO special file to be created.

## 24589 STDIN

24590 Not used.

## 24591 INPUT FILES

24592 None.

## 24593 ENVIRONMENT VARIABLES

24594 The following environment variables shall affect the execution of *mkfifo*:

24595 *LANG* Provide a default value for the internationalization variables that are unset or null.  
24596 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
24597 Internationalization Variables for the precedence of internationalization variables  
24598 used to determine the values of locale categories.)

24599 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
24600 internationalization variables.

24601 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
24602 characters (for example, single-byte as opposed to multi-byte characters in  
24603 arguments).

24604 *LC\_MESSAGES*

24605 Determine the locale that should be used to affect the format and contents of  
24606 diagnostic messages written to standard error.

24607 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

24608 **ASYNCHRONOUS EVENTS**

24609 Default.

24610 **STDOUT**

24611 Not used.

24612 **STDERR**

24613 The standard error shall be used only for diagnostic messages.

24614 **OUTPUT FILES**

24615 None.

24616 **EXTENDED DESCRIPTION**

24617 None.

24618 **EXIT STATUS**

24619 The following exit values shall be returned:

24620 0 All the specified FIFO special files were created successfully.

24621 >0 An error occurred.

24622 **CONSEQUENCES OF ERRORS**

24623 Default.

24624 **APPLICATION USAGE**

24625 None.

24626 **EXAMPLES**

24627 None.

24628 **RATIONALE**

24629 This utility was added to permit shell applications to create FIFO special files.

24630 The **-m** option was added to control the file mode, for consistency with the similar functionality provided the *mkdir* utility.

24632 Early proposals included a **-p** option similar to the *mkdir -p* option that created intermediate directories leading up to the FIFO specified by the final component. This was removed because it is not commonly needed and is not common practice with similar utilities.

24635 The functionality of *mkfifo* is described substantially through a reference to the *mkfifo()* function in the System Interfaces volume of IEEE Std 1003.1-200x. For example, by default, the mode of the FIFO file is affected by the file mode creation mask in accordance with the specified behavior of the *mkfifo()* function. In this way, there is less duplication of effort required for describing details of the file creation.

24640 **FUTURE DIRECTIONS**

24641 None.

24642 **SEE ALSO**

24643 *umask*, the System Interfaces volume of IEEE Std 1003.1-200x, *mkfifo()*

24644 **CHANGE HISTORY**

24645 First released in Issue 3.

## 24646 NAME

24647 more — display files on a page-by-page basis

## 24648 SYNOPSIS

24649 UP `more [-ceisu][-n number][-p command][-t tagstring][file ...]`

24650

## 24651 DESCRIPTION

24652 The *more* utility shall read files and either write them to the terminal on a page-by-page basis or  
 24653 filter them to standard output. If standard output is not a terminal device, all input files shall be  
 24654 copied to standard output in their entirety, without modification, except as specified for the `-s`  
 24655 option. If standard output is a terminal device, the files shall be written a number of lines (one  
 24656 screenful) at a time under the control of user commands. See the EXTENDED DESCRIPTION  
 24657 section.

24658 Certain block-mode terminals do not have all the capabilities necessary to support the complete  
 24659 *more* definition; they are incapable of accepting commands that are not terminated with a  
 24660 <newline>. Implementations that support such terminals shall provide an operating mode to  
 24661 *more* in which all commands can be terminated with a <newline> on those terminals. This mode:

- 24662 • Shall be documented in the system documentation
- 24663 • Shall, at invocation, inform the user of the terminal deficiency that requires the <newline>  
 24664 usage and provide instructions on how this warning can be suppressed in future invocations
- 24665 • Shall not be required for implementations supporting only fully capable terminals
- 24666 • Shall not affect commands already requiring <newline>s
- 24667 • Shall not affect users on the capable terminals from using *more* as described in this volume of  
 24668 IEEE Std 1003.1-200x

## 24669 OPTIONS

24670 The *more* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section  
 24671 12.2, Utility Syntax Guidelines.

24672 The following options shall be supported:

24673 `-c` If a screen is to be written that has no lines in common with the current screen, or  
 24674 *more* is writing its first screen, *more* shall not scroll the screen, but instead shall |  
 24675 redraw each line of the screen in turn, from the top of the screen to the bottom. In |  
 24676 addition, if *more* is writing its first screen, the screen shall be cleared. This option |  
 24677 may be silently ignored on devices with insufficient terminal capabilities. |

24678 `-e` By default, *more* shall exit immediately after writing the last line of the last file in  
 24679 the argument list. If the `-e` option is specified:

24680 1. If there is only a single file in the argument list and that file was completely  
 24681 displayed on a single screen, *more* shall exit immediately after writing the last  
 24682 line of that file.

24683 2. Otherwise, *more* shall exit only after reaching end-of-file on the last file in the  
 24684 argument list twice without an intervening operation. See the EXTENDED  
 24685 DESCRIPTION section.

24686 `-i` Perform pattern matching in searches without regard to case; see the Base  
 24687 Definitions volume of IEEE Std 1003.1-200x, Section 9.2, Regular Expression  
 24688 General Requirements .

- 24689        **-n number**   Specify the number of lines per screenful. The *number* argument is a positive  
24690        decimal integer. The **-n** option shall override any values obtained from any other  
24691        source.
- 24692        **-p command**   Each time a screen from a new file is displayed or redisplayed (including as a  
24693        result of *more* commands; for example, **:p**), execute the *more* command(s) in the  
24694        command arguments in the order specified, as if entered by the user after the first  
24695        screen has been displayed. No intermediate results shall be displayed (that is, if the  
24696        command is a movement to a screen different from the normal first screen, only |  
24697        the screen resulting from the command shall be displayed.) If any of the |  
24698        commands fail for any reason, an informational message to this effect shall be  
24699        written, and no further commands specified using the **-p** option shall be executed  
24700        for this file.
- 24701        **-s**           Behave as if consecutive empty lines were a single empty line.
- 24702        **-t tagstring**   Write the screenful of the file containing the tag named by the *tagstring* argument.  
24703        See the *ctags* utility. The tags feature represented by **-t tagstring** and the **:t**  
24704        command is optional. It shall be provided on any system that also provides a  
24705        conforming implementation of *ctags*; otherwise, the use of **-t** produces undefined  
24706        results.
- 24707        The filename resulting from the **-t** option shall be logically added as a prefix to the  
24708        list of command line files, as if specified by the user. If the tag named by the  
24709        *tagstring* argument is not found, it shall be an error, and *more* shall take no further  
24710        action.
- 24711        If the tag specifies a line number, the first line of the display shall contain the  
24712        beginning of that line. If the tag specifies a pattern, the first line of the display shall  
24713        contain the beginning of the matching text from the first line of the file that  
24714        contains that pattern. If the line does not exist in the file or matching text is not  
24715        found, an informational message to this effect shall be displayed, and *more* shall  
24716        display the default screen as if **-t** had not been specified.
- 24717        If both the **-t tagstring** and **-p command** options are given, the **-t tagstring** shall be  
24718        processed first; that is, the file and starting line for the display shall be as specified  
24719        by **-t**, and then the **-p more** command shall be executed. If the line (matching text)  
24720        specified by the **-t** command does not exist (is not found), no **-p more** command  
24721        shall be executed for this file at any time.
- 24722        **-u**           Treat a <backspace> as a printable control character, displayed as an  
24723        implementation-defined character sequence (see the EXTENDED DESCRIPTION  
24724        section), suppressing backspacing and the special handling that produces  
24725        underlined or standout mode text on some terminal types. Also, do not ignore a  
24726        <carriage-return> at the end of a line.

#### 24727 OPERANDS

24728        The following operand shall be supported:

- 24729        **file**           A pathname of an input file. If no *file* operands are specified, the standard input  
24730        shall be used. If a *file* is '-', the standard input shall be read at that point in the  
24731        sequence.

#### 24732 STDIN

24733        The standard input shall be used only if no *file* operands are specified, or if a *file* operand is '-'.

24734 **INPUT FILES**

24735 The input files being examined shall be text files. If standard output is a terminal, standard error  
 24736 shall be used to read commands from the user. If standard output is a terminal, standard error is  
 24737 not readable, and command input is needed, *more* may attempt to obtain user commands from  
 24738 the controlling terminal (for example, */dev/tty*); otherwise, *more* shall terminate with an error  
 24739 indicating that it was unable to read user commands. If standard output is not a terminal, no  
 24740 error shall result if standard error cannot be opened for reading.

24741 **ENVIRONMENT VARIABLES**

24742 The following environment variables shall affect the execution of *more*:

24743 **COLUMNS** Override the system-selected horizontal display line size. See the Base Definitions  
 24744 volume of IEEE Std 1003.1-200x, Chapter 8, Environment Variables for valid values  
 24745 and results when it is unset or null.

24746 **EDITOR** Used by the *v* command to select an editor. See the EXTENDED DESCRIPTION  
 24747 section.

24748 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 24749 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
 24750 Internationalization Variables for the precedence of internationalization variables  
 24751 used to determine the values of locale categories.)

24752 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 24753 internationalization variables.

24754 **LC\_COLLATE**

24755 Determine the locale for the behavior of ranges, equivalence classes, and multi-  
 24756 character collating elements within regular expressions.

24757 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 24758 characters (for example, single-byte as opposed to multi-byte characters in  
 24759 arguments and input files) and the behavior of character classes within regular  
 24760 expressions.

24761 **LC\_MESSAGES**

24762 Determine the locale that should be used to affect the format and contents of  
 24763 diagnostic messages written to standard error and informative messages written to  
 24764 standard output.

24765 XSI **NLSPATH** Determine the location of message catalogs for the processing of **LC\_MESSAGES**.

24766 **LINES** Override the system-selected vertical screen size, used as the number of lines in a  
 24767 screenful. See the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 8,  
 24768 Environment Variables for valid values and results when it is unset or null. The *-n*  
 24769 option shall take precedence over the **LINES** variable for determining the number  
 24770 of lines in a screenful.

24771 **MORE** Determine a string containing options described in the OPTIONS section preceded  
 24772 with hyphens and <blank>-separated as on the command line. Any command line  
 24773 options shall be processed after those in the **MORE** variable, as if the command  
 24774 line were:

24775 `more $MORE options operands`

24776 The **MORE** variable shall take precedence over the **TERM** and **LINES** variables for  
 24777 determining the number of lines in a screenful.

24778           **TERM**       Determine the name of the terminal type. If this variable is unset or null, an  
24779                           unspecified default terminal type is used.

#### 24780 **ASYNCHRONOUS EVENTS**

24781           Default.

#### 24782 **STDOUT**

24783           The standard output shall be used to write the contents of the input files.

#### 24784 **STDERR**

24785           The standard error shall be used for diagnostic messages and user commands (see the INPUT  
24786           FILES section), and, if standard output is a terminal device, to write a prompting string. The  
24787           prompting string shall appear on the screen line below the last line of the file displayed in the  
24788           current screenful. The prompt shall contain the name of the file currently being examined and  
24789           shall contain an end-of-file indication and the name of the next file, if any, when prompting at  
24790           the end-of-file. If an error or informational message is displayed, it is unspecified whether it is  
24791           contained in the prompt. If it is not contained in the prompt, it shall be displayed and then the  
24792           user shall be prompted for a continuation character, at which point another message or the user  
24793           prompt may be displayed. The prompt is otherwise unspecified. It is unspecified whether  
24794           informational messages are written for other user commands.

#### 24795 **OUTPUT FILES**

24796           None.

#### 24797 **EXTENDED DESCRIPTION**

24798           The following subsection describes the behavior of *more* when the standard output is a terminal  
24799           device. If the standard output is not a terminal device, no options other than **-s** shall have any  
24800           effect, and all input files shall be copied to standard output otherwise unmodified, at which time  
24801           *more* shall exit without further action.

24802           The number of lines available per screen shall be determined by the **-n** option, if present, or by  
24803           examining values in the environment (see the ENVIRONMENT VARIABLES section). If neither  
24804           method yields a number, an unspecified number of lines shall be used.

24805           The maximum number of lines written shall be one less than this number, because the screen  
24806           line after the last line written shall be used to write a user prompt and user input. If the number  
24807           of lines in the screen is less than two, the results are undefined. It is unspecified whether user  
24808           input is permitted to be longer than the remainder of the single line where the prompt has been  
24809           written.

24810           The number of columns available per line shall be determined by examining values in the  
24811           environment (see the ENVIRONMENT VARIABLES section), with a default value as described  
24812           in Base Definitions volume of IEEE Std 1003.1-200x, Chapter 8, Environment Variables.

24813           Lines that are longer than the display shall be folded; the length at which folding occurs is  
24814           unspecified, but should be appropriate for the output device. Folding may occur between glyphs  
24815           of single characters that take up multiple display columns.

24816           When standard output is a terminal and **-u** is not specified, *more* shall treat `<backspace>s` and  
24817           `<carriage-return>s` specially:

- 24818           • A character, followed first by a sequence of *n* `<backspace>s` (where *n* is the same as the  
24819           number of column positions that the character occupies), then by *n* underscore characters  
24820           (`'_'`), shall cause that character to be written as underlined text, if the terminal type  
24821           supports that. The *n* underscore characters, followed first by *n* `<backspace>s`, then any  
24822           character with *n* column positions, shall also cause that character to be written as underlined  
24823           text, if the terminal type supports that.

- 24824           • A sequence of *n* <backspace>s (where *n* is the same as the number of column positions that  
24825           the previous character occupies) that appears between two identical printable characters  
24826           shall cause the first of those two characters to be written as emboldened text (that is, visually  
24827           brighter, standout mode, or inverse-video mode), if the terminal type supports that, and the  
24828           second to be discarded. Immediately subsequent occurrences of <backspace>/character pairs  
24829           for that same character also shall be discarded. (For example, the sequence "a\ba\ba\ba" is  
24830           interpreted as a single emboldened 'a'.)
- 24831           • The *more* utility shall logically discard all other <backspace>s from the line as well as the  
24832           character which precedes them, if any.
- 24833           • A <carriage-return> at the end of a line shall be ignored, rather than being written as a non-  
24834           printable character, as described in the next paragraph.
- 24835           It is implementation-defined how other non-printable characters are written. Implementations  
24836           should use the same format that they use for the *ex print* command; see the OPTIONS section  
24837           within the *ed* utility. It is unspecified whether a multi-column character shall be separated if it  
24838           crosses a display line boundary; it shall not be discarded. The behavior is unspecified if the  
24839           number of columns on the display is less than the number of columns any single character in the  
24840           line being displayed would occupy.
- 24841           When each new file is displayed (or redisplayed), *more* shall write the first screen of the file.  
24842           Once the initial screen has been written, *more* shall prompt for a user command. If the execution  
24843           of the user command results in a screen that has lines in common with the current screen, and  
24844           the device has sufficient terminal capabilities, *more* shall scroll the screen; otherwise, it is  
24845           unspecified whether the screen is scrolled or redrawn.
- 24846           For all files but the last (including standard input if no file was specified, and for the last file as  
24847           well, if the *-e* option was not specified), when *more* has written the last line in the file, *more* shall  
24848           prompt for a user command. This prompt shall contain the name of the next file as well as an  
24849           indication that *more* has reached end-of-file. If the user command is *f*, <control>-F, <space>, *j*,  
24850           <newline>, *d*, <control>-D, or *s*, *more* shall display the next file. Otherwise, if displaying the last  
24851           file, *more* shall exit. Otherwise, *more* shall execute the user command specified.
- 24852           Several of the commands described in this section display a previous screen from the input  
24853           stream. In the case that text is being taken from a non-rewindable stream, such as a pipe, it is  
24854           implementation-defined how much backwards motion is supported. If a command cannot be  
24855           executed because of a limitation on backwards motion, an error message to this effect shall be  
24856           displayed, the current screen shall not change, and the user shall be prompted for another  
24857           command.
- 24858           If a command cannot be performed because there are insufficient lines to display, *more* shall alert  
24859           the terminal. If a command cannot be performed because there are insufficient lines to display or  
24860           a / command fails: if the input is the standard input, the last screen in the file may be displayed;  
24861           otherwise, the current file and screen shall not change, and the user shall be prompted for  
24862           another command.
- 24863           The interactive commands in the following sections shall be supported. Some commands can be  
24864           preceded by a decimal integer, called *count* in the following descriptions. If not specified with  
24865           the command, *count* shall default to 1. In the following descriptions, *pattern* is a basic regular  
24866           expression, as described in the Base Definitions volume of IEEE Std 1003.1-200x, Section 9.3,  
24867           Basic Regular Expressions. The term “examine” is historical usage meaning “open the file for  
24868           viewing”; for example, *more foo* would be expressed as examining file *foo*.
- 24869           In the following descriptions, unless otherwise specified, *line* is a line in the *more* display, not a  
24870           line from the file being examined.

24871 In the following descriptions, the *current position* refers to two things:

- 24872 1. The position of the current line on the screen
- 24873 2. The line number (in the file) of the current line on the screen

24874 Usually, the line on the screen corresponding to the current position is the third line on the  
 24875 screen. If this is not possible (there are fewer than three lines to display or this is the first page of  
 24876 the file, or it is the last page of the file), then the current position is either the first or last line on  
 24877 the screen as described later.

## 24878 **Help**

24879 *Synopsis:*     h

24880 Write a summary of these commands and other implementation-defined commands. The  
 24881 behavior shall be as if the *more* utility were executed with the *-e* option on a file that contained  
 24882 the summary information. The user shall be prompted as described earlier in this section when  
 24883 end-of-file is reached. If the user command is one of those specified to continue to the next file,  
 24884 *more* shall return to the file and screen state from which the **h** command was executed.

## 24885 **Scroll Forward One Screenful**

24886 *Synopsis:*     [*count*]f  
 24887                 [*count*]<control>-F

24888 Scroll forward *count* lines, with a default of one screenful. If *count* is more than the screen size,  
 24889 only the final screenful shall be written.

## 24890 **Scroll Backward One Screenful**

24891 *Synopsis:*     [*count*]b  
 24892                 [*count*]<control>-B

24893 Scroll backward *count* lines, with a default of one screenful (see the *-n* option). If *count* is more  
 24894 than the screen size, only the final screenful shall be written.

## 24895 **Scroll Forward One Line**

24896 *Synopsis:*     [*count*]<space>  
 24897                 [*count*]j  
 24898                 [*count*]<newline>

24899 Scroll forward *count* lines. The default *count* for the <space> shall be one screenful; for **j** and  
 24900 <newline>, one line. The entire *count* lines shall be written, even if *count* is more than the screen  
 24901 size.

## 24902 **Scroll Backward One Line**

24903 *Synopsis:*     [*count*]k

24904 Scroll backward *count* lines. The entire *count* lines shall be written, even if *count* is more than the  
 24905 screen size.

**24906 Scroll Forward One Half Screenful**

24907 *Synopsis:* [count]d  
24908 [count]<control>-D

24909 Scroll forward *count* lines, with a default of one half of the screen size. If *count* is specified, it  
24910 shall become the new default for subsequent **d**, <control>-D, and **u** commands.

**24911 Skip Forward One Line**

24912 *Synopsis:* [count]s

24913 Display the screenful beginning with the line *count* lines after the last line on the current screen.  
24914 If *count* would cause the current position to be such that less than one screenful would be  
24915 written, the last screenful in the file shall be written.

**24916 Scroll Backward One Half Screenful**

24917 *Synopsis:* [count]u  
24918 [count]<control>-U

24919 Scroll backward *count* lines, with a default of one half of the screen size. If *count* is specified, it  
24920 shall become the new default for subsequent **d**, <control>-D, **u**, and <control>-U commands.  
24921 The entire *count* lines shall be written, even if *count* is more than the screen size.

**24922 Go to Beginning of File**

24923 *Synopsis:* [count]g

24924 Display the screenful beginning with line *count*.

**24925 Go to End-of-File**

24926 *Synopsis:* [count]G

24927 If *count* is specified, display the screenful beginning with the line *count*. Otherwise, display the  
24928 last screenful of the file.

**24929 Refresh the Screen**

24930 *Synopsis:* r  
24931 <control>-L

24932 Refresh the screen.

**24933 Discard and Refresh**

24934 *Synopsis:* R

24935 Refresh the screen, discarding any buffered input. If the current file is non-seekable, buffered  
24936 input shall not be discarded and the **R** command shall be equivalent to the **r** command. |

**24937 Mark Position**

24938 *Synopsis:* `mletter`

24939 Mark the current position with the letter named by *letter*, where *letter* represents the name of one  
24940 of the lowercase letters of the portable character set. When a new file is examined, all marks may  
24941 be lost.

**24942 Return to Mark**

24943 *Synopsis:* `'letter`

24944 Return to the position that was previously marked with the letter named by *letter*, making that  
24945 line the current position.

**24946 Return to Previous Position**

24947 *Synopsis:* `' '`

24948 Return to the position from which the last large movement command was executed (where a  
24949 "large movement" is defined as any movement of more than a screenful of lines). If no such  
24950 movements have been made, return to the beginning of the file.

**24951 Search Forward for Pattern**

24952 *Synopsis:* `[count]/[!]pattern<newline>`

24953 Display the screenful beginning with the *count*th line containing the pattern. The search shall  
24954 start after the first line currently displayed. The null regular expression (`'/'` followed by a  
24955 `<newline>`) shall repeat the search using the previous regular expression, with a default *count*. If  
24956 the character `'!'` is included, the matching lines shall be those that do not contain the *pattern*. If  
24957 no match is found for the *pattern*, a message to that effect shall be displayed.

**24958 Search Backward for Pattern**

24959 *Synopsis:* `[count]?[!]pattern<newline>`

24960 Display the screenful beginning with the *count*th previous line containing the pattern. The  
24961 search shall start on the last line before the first line currently displayed. The null regular  
24962 expression (`'?'` followed by a `<newline>`) shall repeat the search using the previous regular  
24963 expression, with a default *count*. If the character `'!'` is included, matching lines shall be those  
24964 that do not contain the *pattern*.

24965 If no match is found for the *pattern*, a message to that effect shall be displayed.

**24966 Repeat Search**

24967 *Synopsis:* `[count]n`

24968 Repeat the previous search for *count*th line containing the last *pattern* (or not containing the last  
24969 *pattern*, if the previous search was `"!/"` or `"?!"`).

**24970 Repeat Search in Reverse**

24971 *Synopsis:* [count]N

24972 Repeat the search in the opposite direction of the previous search for the *count*th line containing  
24973 the last *pattern* (or not containing the last *pattern*, if the previous search was "/" or "?!").

**24974 Examine New File**

24975 *Synopsis:* :e [filename]<newline>

24976 Examine a new file. If the *filename* argument is not specified, the current file (see the :n and :p  
24977 commands below) shall be re-examined. The *filename* shall be subjected to the process of shell  
24978 word expansions (see Section 2.6 (on page 2238)); if more than a single pathname results, the  
24979 effects are unspecified. If *filename* is a number sign ('#'), the previously examined file shall be  
24980 re-examined. If *filename* is not accessible for any reason (including that it is a non-seekable file),  
24981 an error message to this effect shall be displayed and the current file and screen shall not change.

**24982 Examine Next File**

24983 *Synopsis:* [count]:n

24984 Examine the next file. If a number *count* is specified, the *count*th next file shall be examined. If  
24985 *filename* refers to a non-seekable file, the results are unspecified.

**24986 Examine Previous File**

24987 *Synopsis:* [count]:p

24988 Examine the previous file. If a number *count* is specified, the *count*th previous file shall be  
24989 examined. If *filename* refers to a non-seekable file, the results are unspecified.

**24990 Go to Tag**

24991 *Synopsis:* :t tagstring<newline>

24992 If the file containing the tag named by the *tagstring* argument is not the current file, examine the  
24993 file, as if the :e command was executed with that file as the argument. Otherwise, or in addition,  
24994 display the screenful beginning with the tag, as described for the -t option (see the OPTIONS  
24995 section). If the *ctags* utility is not supported by the system, the use of :t produces undefined  
24996 results.

**24997 Invoke Editor**

24998 *Synopsis:* v

24999 Invoke an editor to edit the current file being examined. If standard input is being examined, the  
25000 results are unspecified. The name of the editor shall be taken from the environment variable  
25001 *EDITOR*, or shall default to *vi*. If the last pathname component in *EDITOR* is either *vi* or *ex*, the  
25002 editor shall be invoked with a -c *linenumber* command line argument, where *linenumber* is the  
25003 line number of the file line containing the display line currently displayed as the first line of the  
25004 screen. It is implementation-defined whether line-setting options are passed to editors other  
25005 than *vi* and *ex*.

25006 When the editor exits, *more* shall resume with the same file and screen as when the editor was  
25007 invoked.

25008 **Display Position**

25009 *Synopsis:* =  
 25010 <control>-G

25011 Write a message for which the information references the first byte of the line after the last line of  
 25012 the file on the screen. This message shall include the name of the file currently being examined,  
 25013 its number relative to the total number of files there are to examine, the line number in the file,  
 25014 the byte number and the total bytes in the file, and what percentage of the file precedes the  
 25015 current position. If *more* is reading from standard input, or the file is shorter than a single screen,  
 25016 the line number, the byte number, the total bytes, and the percentage need not be written.

25017 **Quit**

25018 *Synopsis:* q  
 25019 :q  
 25020 ZZ

25021 Exit *more*.

25022 **EXIT STATUS**

25023 The following exit values shall be returned:

25024 0 Successful completion.  
 25025 >0 An error occurred.

25026 **CONSEQUENCES OF ERRORS**

25027 If an error is encountered accessing a file when using the **:n** command, *more* shall attempt to  
 25028 examine the next file in the argument list, but the final exit status shall be affected. If an error is  
 25029 encountered accessing a file via the **:p** command, *more* shall attempt to examine the previous file  
 25030 in the argument list, but the final exit status shall be affected. If an error is encountered accessing  
 25031 a file via the **:e** command, *more* shall remain in the current file and the final exit status shall not  
 25032 be affected.

25033 **APPLICATION USAGE**

25034 When the standard output is not a terminal, only the **-s** filter-modification option is effective.  
 25035 This is based on historical practice. For example, a typical implementation of *man* pipes its  
 25036 output through *more -s* to squeeze excess white space for terminal users. When *man* is piped to  
 25037 *lp*, however, it is undesirable for this squeezing to happen.

25038 **EXAMPLES**

25039 The **-p** allows arbitrary commands to be executed at the start of each file. Examples are:

25040 *more -p G file1 file2*  
 25041 Examine each file starting with its last screenful.

25042 *more -p 100 file1 file2*  
 25043 Examine each file starting with line 100 in the current position (usually the third line, so line  
 25044 98 would be the first line written).

25045 *more -p /100 file1 file2*  
 25046 Examine each file starting with the first line containing the string "100" in the current  
 25047 position

25048 **RATIONALE**

25049 The *more* utility, available in BSD and BSD-derived systems, was chosen as the prototype for the  
 25050 POSIX file display program since it is more widely available than either the public-domain  
 25051 program *less* or than *pg*, a pager provided in System V. The 4.4 BSD *more* is the model for the

25052 features selected; it is almost fully upward-compatible from the 4.3 BSD version in wide use and  
25053 has become more amenable for *vi* users. Several features originally derived from various file  
25054 editors, found in both *less* and *pg*, have been added to this volume of IEEE Std 1003.1-200x as  
25055 they have proved extremely popular with users.

25056 There are inconsistencies between *more* and *vi* that result from historical practice. For example,  
25057 the single-character commands **h**, **f**, **b**, and `<space>` are screen movers in *more*, but cursor  
25058 movers in *vi*. These inconsistencies were maintained because the cursor movements are not  
25059 applicable to *more* and the powerful functionality achieved without the use of the control key  
25060 justifies the differences.

25061 The tags interface has been included in a program that is not a text editor because it promotes  
25062 another degree of consistent operation with *vi*. It is conceivable that the paging environment of  
25063 *more* would be superior for browsing source code files in some circumstances.

25064 The operating mode referred to for block-mode terminals effectively adds a `<newline>` to each  
25065 Synopsis line that currently has none. So, for example, `d<newline>` would page one screenful.  
25066 The mode could be triggered by a command line option, environment variable, or some other  
25067 method. The details are not imposed by this volume of IEEE Std 1003.1-200x because there are so  
25068 few systems known to support such terminals. Nevertheless, it was considered that all systems  
25069 should be able to support *more* given the exception cited for this small community of terminals  
25070 because, in comparison to *vi*, the cursor movements are few and the command set relatively  
25071 amenable to the optional `<newline>s`.

25072 Some versions of *more* provide a shell escaping mechanism similar to the `ex !` command. The  
25073 standard developers did not consider that this was necessary in a paginator, particularly given  
25074 the wide acceptance of multiple window terminals and job control features. (They chose to  
25075 retain such features in the editors and *mailx* because the shell interaction also gives an  
25076 opportunity to modify the editing buffer, which is not applicable to *more*).

25077 The `-p` (position) option replaces the `+` command because of the Utility Syntax Guidelines. In  
25078 early proposals, it took a *pattern* argument, but historical *less* provided the *more* general facility of  
25079 a command. It would have been desirable to use the same `-c` as *ex* and *vi*, but the letter was  
25080 already in use.

25081 The text stating “from a non-rewindable stream ... implementations may limit the amount of  
25082 backwards motion supported” would allow an implementation that permitted no backwards  
25083 motion beyond text already on the screen. It was not possible to require a minimum amount of  
25084 backwards motion that would be effective for all conceivable device types. The implementation  
25085 should allow the user to back up as far as possible, within device and reasonable memory  
25086 allocation constraints.

25087 Historically, non-printable characters were displayed using the ARPA standard mappings,  
25088 which are as follows:

- 25089 1. Printable characters are left alone.
- 25090 2. Control characters less than `\177` are represented as followed by the character offset from  
25091 the `'@'` character in the ASCII map; for example, `\007` is represented as `'G'`.
- 25092 3. `\177` is represented as followed by `'?'`.

25093 The display of characters having their eighth bit set was less standard. Existing implementations  
25094 use hex (`0x00`), octal (`\000`), and a meta-bit display. (The latter displayed characters with their  
25095 eighth bit set as the two characters `"M-`," followed by the seven bit display as described  
25096 previously.) The latter probably has the best claim to historical practice because it was used with  
25097 the `-v` option of 4 BSD and 4 BSD-derived versions of the *cat* utility since 1980.

- 25098 No specific display format is required by IEEE Std 1003.1-200x. Implementations are encouraged  
25099 to conform to historic practice in the absence of any strong reason to diverge.
- 25100 **FUTURE DIRECTIONS**
- 25101 None.
- 25102 **SEE ALSO**
- 25103 *ctags, ed, ex, vi*
- 25104 **CHANGE HISTORY**
- 25105 First released in Issue 4.
- 25106 **Issue 5**
- 25107 FUTURE DIRECTIONS section added.
- 25108 **Issue 6**
- 25109 This utility is now marked as part of the User Portability Utilities option.
- 25110 The obsolescent SYNOPSIS is removed.
- 25111 The utility has been extensively reworked for alignment with the IEEE P1003.2b draft standard:
- 25112
  - Changes have been made as result of IEEE PASC Interpretations 1003.2 #37 and #109.
- 25113
  - The *more* utility should be able to handle underlined and emboldened displays of characters
- 25114 that are wider than a single column position.

## 25115 NAME

25116 mv — move files

## 25117 SYNOPSIS

25118 mv [-fi] *source\_file target\_file*25119 mv [-fi] *source\_file... target\_file*

## 25120 DESCRIPTION

25121 In the first synopsis form, the *mv* utility shall move the file named by the *source\_file* operand to  
 25122 the *destination* specified by the *target\_file*. This first synopsis form is assumed when the final  
 25123 operand does not name an existing directory and is not a symbolic link referring to an existing  
 25124 directory.

25125 In the second synopsis form, *mv* shall move each file named by a *source\_file* operand to a  
 25126 *destination* file in the existing directory named by the *target\_dir* operand, or referenced if  
 25127 *target\_dir* is a symbolic link referring to an existing directory. The *destination* path for each  
 25128 *source\_file* shall be the concatenation of the target directory, a single slash character, and the last  
 25129 pathname component of the *source\_file*. This second form is assumed when the final operand  
 25130 names an existing directory.

25131 If any operand specifies an existing file of a type not specified by the System Interfaces volume  
 25132 of IEEE Std 1003.1-200x, the behavior is implementation-defined.

25133 For each *source\_file* the following steps shall be taken:

25134 1. If the destination path exists, the *-f* option is not specified, and either of the following  
 25135 conditions is true:

25136 a. The permissions of the destination path do not permit writing and the standard input  
 25137 is a terminal.

25138 b. The *-i* option is specified.

25139 the *mv* utility shall write a prompt to standard error and read a line from standard input. If  
 25140 the response is not affirmative, *mv* shall do nothing more with the current *source\_file* and  
 25141 go on to any remaining *source\_files*.

25142 2. The *mv* utility shall perform actions equivalent to the *rename()* function defined in the  
 25143 System Interfaces volume of IEEE Std 1003.1-200x, called with the following arguments:

25144 a. The *source\_file* operand is used as the *old* argument.

25145 b. The destination path is used as the *new* argument.

25146 If this succeeds, *mv* shall do nothing more with the current *source\_file* and go on to any  
 25147 remaining *source\_files*. If this fails for any reasons other than those described for the *errno*  
 25148 [EXDEV] in the System Interfaces volume of IEEE Std 1003.1-200x, *mv* shall write a  
 25149 diagnostic message to standard error, do nothing more with the current *source\_file*, and go  
 25150 on to any remaining *source\_files*.

25151 3. If the destination path exists, and it is a file of type directory and *source\_file* is not a file of  
 25152 type directory, or it is a file not of type directory and *source\_file* is a file of type directory,  
 25153 *mv* shall write a diagnostic message to standard error, do nothing more with the current  
 25154 *source\_file*, and go on to any remaining *source\_files*.

25155 4. If the destination path exists, *mv* shall attempt to remove it. If this fails for any reason, *mv*  
 25156 shall write a diagnostic message to standard error, do nothing more with the current  
 25157 *source\_file*, and go on to any remaining *source\_files*.

25158 5. The file hierarchy rooted in *source\_file* shall be duplicated as a file hierarchy rooted in the  
 25159 *destination* path. If *source\_file* or any of the files below it in the hierarchy are symbolic links,  
 25160 the links themselves shall be duplicated, including their contents, rather than any files to  
 25161 which they refer. The following characteristics of each file in the file hierarchy shall be  
 25162 duplicated:

- 25163 • The time of last data modification and time of last access
- 25164 • The user ID and group ID
- 25165 • The file mode

25166 If the user ID, group ID, or file mode of a regular file cannot be duplicated, the file mode  
 25167 bits S\_ISUID and S\_ISGID shall not be duplicated.

25168 When files are duplicated to another file system, the implementation may require that the  
 25169 process invoking *mv* has read access to each file being duplicated.

25170 If the duplication of the file hierarchy fails for any reason, *mv* shall write a diagnostic  
 25171 message to standard error, do nothing more with the current *source\_file*, and go on to any  
 25172 remaining *source\_files*.

25173 If the duplication of the file characteristics fails for any reason, *mv* shall write a diagnostic  
 25174 message to standard error, but this failure shall not cause *mv* to modify its exit status.

25175 6. The file hierarchy rooted in *source\_file* shall be removed. If this fails for any reason, *mv* shall  
 25176 write a diagnostic message to the standard error, do nothing more with the current  
 25177 *source\_file*, and go on to any remaining *source\_files*.

#### 25178 OPTIONS

25179 The *mv* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section  
 25180 12.2, Utility Syntax Guidelines.

25181 The following options shall be supported:

- 25182 **-f** Do not prompt for confirmation if the destination path exists. Any previous  
 25183 occurrences of the **-i** option is ignored.
- 25184 **-i** Prompt for confirmation if the destination path exists. Any previous occurrences of  
 25185 the **-f** option is ignored.

25186 Specifying more than one of the **-f** or **-i** options shall not be considered an error. The last option  
 25187 specified shall determine the behavior of *mv*.

#### 25188 OPERANDS

25189 The following operands shall be supported:

- 25190 *source\_file* A pathname of a file or directory to be moved.
- 25191 *target\_file* A new pathname for the file or directory being moved.
- 25192 *target\_dir* A pathname of an existing directory into which to move the input files.

#### 25193 STDIN

25194 The standard input shall be used to read an input line in response to each prompt specified in |  
 25195 the STDERR section. Otherwise, the standard input shall not be used. |

#### 25196 INPUT FILES

25197 The input files specified by each *source\_file* operand can be of any file type.

25198 **ENVIRONMENT VARIABLES**

25199 The following environment variables shall affect the execution of *mv*:

25200 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 25201 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
 25202 Internationalization Variables for the precedence of internationalization variables  
 25203 used to determine the values of locale categories.)

25204 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 25205 internationalization variables.

25206 *LC\_COLLATE*

25207 Determine the locale for the behavior of ranges, equivalence classes and multi-  
 25208 character collating elements used in the extended regular expression defined for  
 25209 the **yesexpr** locale keyword in the *LC\_MESSAGES* category.

25210 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 25211 characters (for example, single-byte as opposed to multi-byte characters in  
 25212 arguments and input files), the behavior of character classes used in the extended  
 25213 regular expression defined for the **yesexpr** locale keyword in the *LC\_MESSAGES*  
 25214 category.

25215 *LC\_MESSAGES*

25216 Determine the locale for the processing of affirmative responses that should be  
 25217 used to affect the format and contents of diagnostic messages written to standard  
 25218 error.

25219 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

25220 **ASYNCHRONOUS EVENTS**

25221 Default.

25222 **STDOUT**

25223 Not used.

25224 **STDERR**

25225 Prompts shall be written to the standard error under the conditions specified in the  
 25226 DESCRIPTION section. The prompts shall contain the *destination* pathname, but their format is  
 25227 otherwise unspecified. Otherwise, the standard error shall be used only for diagnostic messages.

25228 **OUTPUT FILES**

25229 The output files may be of any file type.

25230 **EXTENDED DESCRIPTION**

25231 None.

25232 **EXIT STATUS**

25233 The following exit values shall be returned:

25234 0 All input files were moved successfully.

25235 >0 An error occurred.

25236 **CONSEQUENCES OF ERRORS**

25237 If the copying or removal of *source\_file* is prematurely terminated by a signal or error, *mv* may  
 25238 leave a partial copy of *source\_file* at the source or destination. The *mv* utility shall not modify  
 25239 both *source\_file* and the destination path simultaneously; termination at any point shall leave  
 25240 either *source\_file* or the destination path complete.

25241 **APPLICATION USAGE**

25242 Some implementations mark for update the *st\_ctime* field of renamed files and some do not. |  
 25243 Applications which make use of the *st\_ctime* field may behave differently with respect to |  
 25244 renamed files unless they are designed to allow for either behavior. |

25245 **EXAMPLES**

25246 If the current directory contains only files **a** (of any type defined by the System Interfaces |  
 25247 volume of IEEE Std 1003.1-200x), **b** (also of any type), and a directory **c**:

25248 `mv a b c`

25249 `mv c d`

25250 results with the original files **a** and **b** residing in the directory **d** in the current directory.

25251 **RATIONALE**

25252 Early proposals diverged from the SVID and BSD historical practice in that they required that |  
 25253 when the destination path exists, the `-f` option is not specified, and input is not a terminal, *mv* |  
 25254 fails. This was done for compatibility with *cp*. The current text returns to historical practice. It |  
 25255 should be noted that this is consistent with the *rename()* function defined in the System |  
 25256 Interfaces volume of IEEE Std 1003.1-200x, which does not require write permission on the |  
 25257 target.

25258 For absolute clarity, paragraph (1), describing the behavior of *mv* when prompting for |  
 25259 confirmation, should be interpreted in the following manner:

```
25260 if (exists AND (NOT f_option) AND
25261     ((not_writable AND input_is_terminal) OR i_option))
```

25262 The `-i` option exists on BSD systems, giving applications and users a way to avoid accidentally |  
 25263 unlinking files when moving others. When the standard input is not a terminal, the 4.3 BSD *mv* |  
 25264 deletes all existing destination paths without prompting, even when `-i` is specified; this is |  
 25265 inconsistent with the behavior of the 4.3 BSD *cp* utility, which always generates an error when |  
 25266 the file is unwritable and the standard input is not a terminal. The standard developers decided |  
 25267 that use of `-i` is a request for interaction, so when the *destination* path exists, the utility takes |  
 25268 instructions from whatever responds to standard input.

25269 The *rename()* function is able to move directories within the same file system. Some historical |  
 25270 versions of *mv* have been able to move directories, but not to a different file system. The |  
 25271 standard developers considered that this was an annoying inconsistency, so this volume of |  
 25272 IEEE Std 1003.1-200x requires directories to be able to be moved even across file systems. There |  
 25273 is no `-R` option to confirm that moving a directory is actually intended, since such an option was |  
 25274 not required for moving directories in historical practice. Requiring the application to specify it |  
 25275 sometimes, depending on the destination, seemed just as inconsistent. The semantics of the |  
 25276 *rename()* function were preserved as much as possible. For example, *mv* is not permitted to |  
 25277 “rename” files to or from directories, even though they might be empty and removable.

25278 Historic implementations of *mv* did not exit with a non-zero exit status if they were unable to |  
 25279 duplicate any file characteristics when moving a file across file systems, nor did they write a |  
 25280 diagnostic message for the user. The former behavior has been preserved to prevent scripts from |  
 25281 breaking; a diagnostic message is now required, however, so that users are alerted that the file |  
 25282 characteristics have changed.

25283 The exact format of the interactive prompts is unspecified. Only the general nature of the |  
 25284 contents of prompts are specified because implementations may desire more descriptive |  
 25285 prompts than those used on historical implementations. Therefore, an application not using the |  
 25286 `-f` option or using the `-i` option relies on the system to provide the most suitable dialog directly |  
 25287 with the user, based on the behavior specified.

25288           When *mv* is dealing with a single file system and *source\_file* is a symbolic link, the link itself is  
25289           moved as a consequence of the dependence on the *rename()* functionality, per the  
25290           DESCRIPTION. Across file systems, this has to be made explicit.

25291 **FUTURE DIRECTIONS**

25292           None.

25293 **SEE ALSO**

25294           *cp*, *ln*

25295 **CHANGE HISTORY**

25296           First released in Issue 2.

25297 **Issue 6**

25298           The *mv* utility is changed to describe processing of symbolic links as specified in the  
25299           IEEE P1003.2b draft standard. |

25300           The APPLICATION USAGE section is added. |

25301 **NAME**

25302           newgrp — change to a new group

25303 **SYNOPSIS**

25304 UP       newgrp [-l][group]

25305

25306 **DESCRIPTION**

25307       The *newgrp* utility shall create a new shell execution environment with a new real and effective  
 25308 group identification. Of the attributes listed in Section 2.12 (on page 2263), the new shell  
 25309 execution environment shall retain the working directory, file creation mask, and exported  
 25310 variables from the previous environment (that is, open files, traps, unexported variables, alias  
 25311 definitions, shell functions, and *set* options may be lost). All other aspects of the process  
 25312 environment that are preserved by the *exec* family of functions defined in the System Interfaces  
 25313 volume of IEEE Std 1003.1-200x shall also be preserved by *newgrp*; whether other aspects are  
 25314 preserved is unspecified.

25315       A failure to assign the new group identifications (for example, for security or password-related  
 25316 reasons) shall not prevent the new shell execution environment from being created.

25317       The *newgrp* utility shall affect the supplemental groups for the process as follows:

- 25318       • On systems where the effective group ID is normally in the supplementary group list (or  
 25319 whenever the old effective group ID actually is in the supplementary group list):

- 25320       — If the new effective group ID is also in the supplementary group list, *newgrp* shall change  
 25321 the effective group ID.

- 25322       — If the new effective group ID is not in the supplementary group list, *newgrp* shall add the  
 25323 new effective group ID to the list, if there is room to add it.

- 25324       • On systems where the effective group ID is not normally in the supplementary group list (or  
 25325 whenever the old effective group ID is not in the supplementary group list):

- 25326       — If the new effective group ID is in the supplementary group list, *newgrp* shall delete it.

- 25327       — If the old effective group ID is not in the supplementary list, *newgrp* shall add it if there is  
 25328 room.

25329       **Note:**     The System Interfaces volume of IEEE Std 1003.1-200x does not specify whether the effective  
 25330 group ID of a process is included in its supplementary group list.

25331       With no operands, *newgrp* shall change the effective group back to the groups identified in the  
 25332 user's user entry, and shall set the list of supplementary groups to that set in the user's group  
 25333 database entries.

25334       If a password is required for the specified group, and the user is not listed as a member of that  
 25335 group in the group database, the user shall be prompted to enter the correct password for that  
 25336 group. If the user is listed as a member of that group, no password shall be requested. If no  
 25337 password is required for the specified group, it is implementation-defined whether users not  
 25338 listed as members of that group can change to that group. Whether or not a password is  
 25339 required, implementation-defined system accounting or security mechanisms may impose  
 25340 additional authorization restrictions that may cause *newgrp* to write a diagnostic message and  
 25341 suppress the changing of the group identification.

25342 **OPTIONS**

25343       The *newgrp* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section  
 25344 12.2, Utility Syntax Guidelines.

25345 The following option shall be supported:

25346 **-l** (The letter ell.) Change the environment to what would be expected if the user  
25347 actually logged in again.

#### 25348 OPERANDS

25349 The following operand shall be supported:

25350 *group* A group name from the group database or a non-negative numeric group ID.  
25351 Specifies the group ID to which the real and effective group IDs shall be set. If  
25352 *group* is a non-negative numeric string and exists in the group database as a group  
25353 name (see *getgrnam()*), the numeric group ID associated with that group name  
25354 shall be used as the group ID.

#### 25355 STDIN

25356 Not used.

#### 25357 INPUT FILES

25358 The file */dev/tty* shall be used to read a single line of text for password checking, when one is  
25359 required.

#### 25360 ENVIRONMENT VARIABLES

25361 The following environment variables shall affect the execution of *newgrp*:

25362 *LANG* Provide a default value for the internationalization variables that are unset or null.  
25363 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
25364 Internationalization Variables for the precedence of internationalization variables  
25365 used to determine the values of locale categories.)

25366 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
25367 internationalization variables.

25368 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
25369 characters (for example, single-byte as opposed to multi-byte characters in  
25370 arguments).

#### 25371 LC\_MESSAGES

25372 Determine the locale that should be used to affect the format and contents of  
25373 diagnostic messages written to standard error.

25374 *XSI NLS\_PATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

#### 25375 ASYNCHRONOUS EVENTS

25376 Default.

#### 25377 STDOUT

25378 Not used.

#### 25379 STDERR

25380 The standard error shall be used for diagnostic messages and a prompt string for a password, if |  
25381 one is required. Diagnostic messages may be written in cases where the exit status is not |  
25382 available. See the EXIT STATUS section. |

#### 25383 OUTPUT FILES

25384 None.

#### 25385 EXTENDED DESCRIPTION

25386 None.

25387 **EXIT STATUS**

25388 If *newgrp* succeeds in creating a new shell execution environment, whether or not the group  
 25389 identification was changed successfully, the exit status shall be the exit status of the shell.  
 25390 Otherwise, the following exit value shall be returned:

25391 >0 An error occurred.

25392 **CONSEQUENCES OF ERRORS**

25393 The invoking shell may terminate.

25394 **APPLICATION USAGE**

25395 There is no convenient way to enter a password into the Group Database. Use of group  
 25396 passwords is not encouraged, because by their very nature they encourage poor security  
 25397 practices. Group passwords may disappear in the future.

25398 A common implementation of *newgrp* is that the current shell uses *exec* to overlay itself with  
 25399 *newgrp*, which in turn overlays itself with a new shell after changing group. On some  
 25400 implementations, however, this may not occur and *newgrp* may be invoked as a subprocess.

25401 The *newgrp* command is intended only for use from an interactive terminal. It does not offer a  
 25402 useful interface for the support of applications.

25403 The exit status of *newgrp* is generally inapplicable. If *newgrp* is used in a script, in most cases it  
 25404 successfully invokes a new shell and the rest of the original shell script is bypassed when the  
 25405 new shell exits. Used interactively, *newgrp* displays diagnostic messages to indicate problems.  
 25406 But usage such as:

```
25407 newgrp foo
25408 echo $?
```

25409 is not useful because the new shell might not have access to any status *newgrp* may have  
 25410 generated (and most historical systems do not provide this status). A zero status echoed here  
 25411 does not necessarily indicate that the user has changed to the new group successfully. Following  
 25412 *newgrp* with the *id* command provides a portable means of determining whether the group  
 25413 change was successful or not.

25414 **EXAMPLES**

25415 None.

25416 **RATIONALE**

25417 Most historical implementations use one of the *exec* functions to implement the behavior of  
 25418 *newgrp*. Errors detected before the *exec* leave the environment unchanged, while errors detected  
 25419 after the *exec* leave the user in a changed environment. While it would be useful to have *newgrp*  
 25420 issue a diagnostic message to tell the user that the environment changed, it would be  
 25421 inappropriate to require this change to some historical implementations.

25422 The password mechanism is allowed in the group database, but how this would be  
 25423 implemented is not specified.

25424 The *newgrp* utility was retained in this volume of IEEE Std 1003.1-200x, even given the existence  
 25425 of the multiple group permissions feature in the System Interfaces volume of  
 25426 IEEE Std 1003.1-200x, for several reasons. First, in some implementations, the group ownership  
 25427 of a newly created file is determined by the group of the directory in which the file is created, as  
 25428 allowed by the System Interfaces volume of IEEE Std 1003.1-200x; on other implementations, the  
 25429 group ownership of a newly created file is determined by the effective group ID. On  
 25430 implementations of the latter type, *newgrp* allows files to be created with a specific group  
 25431 ownership. Finally, many implementations use the real group ID in accounting, and on such  
 25432 systems, *newgrp* allows the accounting identity of the user to be changed.

25433 **FUTURE DIRECTIONS**

25434 None.

25435 **SEE ALSO**25436 *sh*, the System Interfaces volume of IEEE Std 1003.1-200x, *exec*25437 **CHANGE HISTORY**

25438 First released in Issue 2.

25439 **Issue 6**

25440 This utility is now marked as part of the User Portability Utilities option.

25441 The obsolescent SYNOPSIS is removed.

25442 The text describing supplemental groups is no longer conditional on {NGROUPS\_MAX} being greater than 1. This is because {NGROUPS\_MAX} now has a minimum value of 8. This is a FIPS

25444 requirement.

25445 **NAME**25446           *nice* — invoke a utility with an altered nice value25447 **SYNOPSIS**25448 UP       *nice* [-n *increment*] *utility* [*argument...*]

25449

25450 **DESCRIPTION**

25451       The *nice* utility shall invoke a utility, requesting that it be run with a different nice value (see the  
 25452       Base Definitions volume of IEEE Std 1003.1-200x, Section 3.239, Nice Value). With no options  
 25453       and only if the user has appropriate privileges, the executed utility shall be run with a nice value  
 25454       that is some implementation-defined quantity less than or equal to the nice value of the current  
 25455       process. If the user lacks appropriate privileges to affect the nice value in the requested manner,  
 25456       the *nice* utility shall not affect the nice value; in this case, a warning message may be written to  
 25457       standard error, but this shall not prevent the invocation of *utility* or affect the exit status.

25458 **OPTIONS**

25459       The *nice* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section  
 25460       12.2, Utility Syntax Guidelines.

25461       The following option is supported:

25462       **-n *increment*** A positive or negative decimal integer which shall have the same effect on the |  
 25463       execution of the utility as if the utility had called the *nice*() function with the |  
 25464       numeric value of the *increment* option-argument. |

25465 **OPERANDS**

25466       The following operands shall be supported:

25467       ***utility***       The name of a utility that is to be invoked. If the *utility* operand names any of the  
 25468       special built-in utilities in Section 2.14 (on page 2266), the results are undefined.

25469       ***argument***     Any string to be supplied as an argument when invoking the utility named by the  
 25470       *utility* operand.

25471 **STDIN**

25472       Not used.

25473 **INPUT FILES**

25474       None.

25475 **ENVIRONMENT VARIABLES**25476       The following environment variables shall affect the execution of *nice*:

25477       ***LANG***        Provide a default value for the internationalization variables that are unset or null.  
 25478       (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
 25479       Internationalization Variables for the precedence of internationalization variables  
 25480       used to determine the values of locale categories.)

25481       ***LC\_ALL***       If set to a non-empty string value, override the values of all the other  
 25482       internationalization variables.

25483       ***LC\_CTYPE***     Determine the locale for the interpretation of sequences of bytes of text data as  
 25484       characters (for example, single-byte as opposed to multi-byte characters in  
 25485       arguments).

25486       ***LC\_MESSAGES***

25487       Determine the locale that should be used to affect the format and contents of  
 25488       diagnostic messages written to standard error.

|       |     |                                                 |                                                                                                                                                                |
|-------|-----|-------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 25489 | XSI | <b>NLSPATH</b>                                  | Determine the location of message catalogs for the processing of <i>LC_MESSAGES</i> .                                                                          |
| 25490 |     | <b>PATH</b>                                     | Determine the search path used to locate the utility to be invoked. See the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 8, Environment Variables. |
| 25491 |     |                                                 |                                                                                                                                                                |
| 25492 |     | <b>ASYNCHRONOUS EVENTS</b>                      |                                                                                                                                                                |
| 25493 |     |                                                 | Default.                                                                                                                                                       |
| 25494 |     | <b>STDOUT</b>                                   |                                                                                                                                                                |
| 25495 |     |                                                 | Not used.                                                                                                                                                      |
| 25496 |     | <b>STDERR</b>                                   |                                                                                                                                                                |
| 25497 |     |                                                 | The standard error shall be used only for diagnostic messages.                                                                                                 |
| 25498 |     | <b>OUTPUT FILES</b>                             |                                                                                                                                                                |
| 25499 |     |                                                 | None.                                                                                                                                                          |
| 25500 |     | <b>EXTENDED DESCRIPTION</b>                     |                                                                                                                                                                |
| 25501 |     |                                                 | None.                                                                                                                                                          |
| 25502 |     | <b>EXIT STATUS</b>                              |                                                                                                                                                                |
| 25503 |     |                                                 | If the <i>utility</i> utility is invoked, the exit status of <i>nice</i> shall be the exit status of <i>utility</i> ; otherwise,                               |
| 25504 |     |                                                 | the <i>nice</i> utility shall exit with one of the following values:                                                                                           |
| 25505 |     | 1-125                                           | An error occurred in the <i>nice</i> utility.                                                                                                                  |
| 25506 |     | 126                                             | The utility specified by <i>utility</i> was found but could not be invoked.                                                                                    |
| 25507 |     | 127                                             | The utility specified by <i>utility</i> could not be found.                                                                                                    |
| 25508 |     | <b>CONSEQUENCES OF ERRORS</b>                   |                                                                                                                                                                |
| 25509 |     |                                                 | Default.                                                                                                                                                       |
| 25510 |     | <b>APPLICATION USAGE</b>                        |                                                                                                                                                                |
| 25511 |     |                                                 | The only guaranteed portable uses of this utility are:                                                                                                         |
| 25512 |     | <i>nice utility</i>                             |                                                                                                                                                                |
| 25513 |     |                                                 | Run <i>utility</i> with the default lower nice value.                                                                                                          |
| 25514 |     | <i>nice -n &lt;positive integer&gt; utility</i> |                                                                                                                                                                |
| 25515 |     |                                                 | Run <i>utility</i> with a lower nice value.                                                                                                                    |
| 25516 |     |                                                 | On some implementations they have no discernible effect on the invoked utility and on some                                                                     |
| 25517 |     |                                                 | others they are exactly equivalent.                                                                                                                            |
| 25518 |     |                                                 | Historical systems have frequently supported the <i>&lt;positive integer&gt;</i> up to 20. Since there is no                                                   |
| 25519 |     |                                                 | error penalty associated with guessing a number that is too high, users without access to the                                                                  |
| 25520 |     |                                                 | system conformance document (to see what limits are actually in place) could use the historical                                                                |
| 25521 |     |                                                 | 1 to 20 range or attempt to use very large numbers if the job should be truly low priority.                                                                    |
| 25522 |     |                                                 | The nice value value of a process can be displayed using the command:                                                                                          |
| 25523 |     |                                                 | <code>ps -o nice</code>                                                                                                                                        |
| 25524 |     |                                                 | The <i>command</i> , <i>env</i> , <i>nice</i> , <i>nohup</i> , <i>time</i> , and <i>xargs</i> utilities have been specified to use exit code 127 if            |
| 25525 |     |                                                 | an error occurs so that applications can distinguish “failure to find a utility” from “invoked                                                                 |
| 25526 |     |                                                 | utility exited with an error indication”. The value 127 was chosen because it is not commonly                                                                  |
| 25527 |     |                                                 | used for other meanings; most utilities use small values for “normal error conditions” and the                                                                 |
| 25528 |     |                                                 | values above 128 can be confused with termination due to receipt of a signal. The value 126 was                                                                |
| 25529 |     |                                                 | chosen in a similar manner to indicate that the utility could be found, but not invoked. Some                                                                  |
| 25530 |     |                                                 | scripts produce meaningful error messages differentiating the 126 and 127 cases. The distinction                                                               |
| 25531 |     |                                                 | between exit codes 126 and 127 is based on KornShell practice that uses 127 when all attempts to                                                               |
| 25532 |     |                                                 | <i>exec</i> the utility fail with [ENOENT], and uses 126 when any attempt to <i>exec</i> the utility fails for                                                 |

25533 any other reason.

25534 **EXAMPLES**

25535 None.

25536 **RATIONALE**

25537 Due to the text about the limits of the *nice* value being implementation-defined, *nice* is not  
25538 actually required to change the *nice* value of the executed command; the limits could be zero  
25539 differences from the system default, although the implementor is required to document this fact  
25540 in the conformance document.

25541 The 4.3 BSD version of *nice* does not check if *increment* is a valid decimal integer. The command  
25542 *nice -x utility*, for example, would be treated the same as the command *nice --1 utility*. If the  
25543 user does not have appropriate privileges, this results in a “permission denied” error. This is  
25544 considered a bug.

25545 When a user without appropriate privileges gives a negative *increment*, System V treats it like  
25546 the command *nice -0 utility*, while 4.3 BSD writes a “permission denied” message and does not  
25547 run the utility. Neither was considered clearly superior, so the behavior was left unspecified.

25548 The C shell has a built-in version of *nice* that has a different interface from the one described in  
25549 this volume of IEEE Std 1003.1-200x.

25550 The term “utility” is used, rather than “command”, to highlight the fact that shell compound  
25551 commands, pipelines, and so on, cannot be used. Special built-ins also cannot be used.  
25552 However, “utility” includes user application programs and shell scripts, not just utilities defined  
25553 in this volume of IEEE Std 1003.1-200x.

25554 Historical implementations of *nice* provide a *nice* value range of 40 or 41 discrete steps, with the  
25555 default *nice* value being the midpoint of that range. By default, they lower the *nice* value of the  
25556 executed utility by 10.

25557 Some historical documentation states that the *increment* value must be within a fixed range. This  
25558 is misleading; the valid *increment* values on any invocation are determined by the current  
25559 process *nice* value, which is not always the default.

25560 The definition of *nice* value is not intended to suggest that all processes in a system have  
25561 priorities that are comparable. Scheduling policy extensions such as the realtime priorities in the  
25562 System Interfaces volume of IEEE Std 1003.1-200x make the notion of a single underlying  
25563 priority for all scheduling policies problematic. Some implementations may implement the *nice*-  
25564 related features to affect all processes on the system, others to affect just the general time-  
25565 sharing activities implied by this volume of IEEE Std 1003.1-200x, and others may have no effect  
25566 at all. Because of the use of “implementation-defined” in *nice* and *renice*, a wide range of  
25567 implementation strategies are possible.

25568 **FUTURE DIRECTIONS**

25569 None.

25570 **SEE ALSO**

25571 *renice*, the System Interfaces volume of IEEE Std 1003.1-200x, *nice()*

25572 **CHANGE HISTORY**

25573 First released in Issue 4.

25574 **Issue 6**

25575 This utility is now marked as part of the User Portability Utilities option.

25576 The obsolescent SYNOPSIS is removed.

## 25577 NAME

25578 nl — line numbering filter

## 25579 SYNOPSIS

```
25580 xSI nl [-p][-b type][-d delim][-f type][-h type][-i incr][-l num][-n format]
25581 [-s sep][-v startnum][-w width][file]
25582
```

## 25583 DESCRIPTION

25584 The *nl* utility shall read lines from the named *file* or the standard input if no *file* is named and  
 25585 shall reproduce the lines to standard output. Lines shall be numbered on the left. Additional  
 25586 functionality may be provided in accordance with the command options in effect.

25587 The *nl* utility views the text it reads in terms of logical pages. Line numbering shall be reset at  
 25588 the start of each logical page. A logical page consists of a header, a body, and a footer section.  
 25589 Empty sections are valid. Different line numbering options are independently available for  
 25590 header, body, and footer (for example, no numbering of header and footer lines while  
 25591 numbering blank lines only in the body).

25592 The starts of logical page sections shall be signaled by input lines containing nothing but the  
 25593 following delimiter characters:

25594

25595

25596

25597

| Line   | Start of |
|--------|----------|
| \:\:\: | Header   |
| \:\:   | Body     |
| \:     | Footer   |

25598 Unless otherwise specified, *nl* shall assume the text being read is in a single logical page body.

## 25599 OPTIONS

25600 The *nl* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2,  
 25601 Utility Syntax Guidelines. Only one file can be named.

25602 The following options shall be supported:

25603 **-b type** Specify which logical page body lines shall be numbered. Recognized *types* and  
 25604 their meaning are:

25605 **a** Number all lines.

25606 **t** Number only non-empty lines.

25607 **n** No line numbering.

25608 **pstring** Number only lines that contain the basic regular expression specified in  
 25609 *string*.

25610 The default *type* for logical page body shall be **t** (text lines numbered).

25611 **-d delim** Specify the delimiter characters that indicate the start of a logical page section.  
 25612 These can be changed from the default characters "\:" to two user-specified  
 25613 characters. If only one character is entered, the second character shall remain the  
 25614 default character ' : '.

25615 **-f type** Specify the same as **b type** except for footer. The default for logical page footer  
 25616 shall be **n** (no lines numbered).

25617 **-h type** Specify the same as **b type** except for header. The default *type* for logical page  
 25618 header shall be **n** (no lines numbered).



25660 **ASYNCHRONOUS EVENTS**

25661 Default.

25662 **STDOUT**

25663 The standard output shall be a text file in the following format:

25664 "%s%s%s", &lt;line number&gt;, &lt;separator&gt;, &lt;input line&gt;

25665 where &lt;line number&gt; is one of the following numeric formats:

25666 %6d When the **rn** format is used (the default; see **-n**).25667 %06d When the **rz** format is used.25668 %-6d When the **ln** format is used.

25669 &lt;empty&gt; When line numbers are suppressed for a portion of the page; the &lt;separator&gt; is also

25670 suppressed.

25671 In the preceding list, the number 6 is the default width; the **-w** option can change this value.25672 **STDERR**

25673 The standard error shall be used only for diagnostic messages.

25674 **OUTPUT FILES**

25675 None.

25676 **EXTENDED DESCRIPTION**

25677 None.

25678 **EXIT STATUS**

25679 The following exit values shall be returned:

25680 0 Successful completion.

25681 &gt;0 An error occurred.

25682 **CONSEQUENCES OF ERRORS**

25683 Default.

25684 **APPLICATION USAGE**25685 In using the **-d delim** option, care should be taken to escape characters that have special meaning  
25686 to the command interpreter.25687 **EXAMPLES**

25688 The command:

25689 nl -v 10 -i 10 -d \!+ file1

25690 numbers *file1* starting at line number 10 with an increment of 10. The logical page delimiter is  
25691 " !+". Note that the ' ! ' has to be escaped when using *csh* as a command interpreter because of  
25692 its history substitution syntax. For *ksh* and *sh* the escape is not necessary, but does not do any  
25693 harm.25694 **RATIONALE**

25695 None.

25696 **FUTURE DIRECTIONS**

25697 None.

25698 **SEE ALSO**

25699 *pr*

25700 **CHANGE HISTORY**

25701 First released in Issue 2.

25702 **Issue 5**

25703 The option `[-f type]` is added to the SYNOPSIS. The option descriptions are presented in  
25704 alphabetic order. The description of `-bt` is changed to “Number only non-empty lines”.

25705 **Issue 6**

25706 The obsolescent behavior allowing the options to be intermingled with the optional *file* operand  
25707 is removed.

## 25708 NAME

25709 nm — write the name list of an object file (DEVELOPMENT)

## 25710 SYNOPSIS

25711 UP SD XSI nm [-APv][-efox][ -g | -u][-t *format*] *file...*

25712

## 25713 DESCRIPTION

25714 This utility shall be provided on systems that support both the User Portability Utilities option  
25715 and the Software Development Utilities option. On other systems it is optional. Certain options  
25716 are only available on XSI-conformant systems.

25717 The *nm* utility shall display symbolic information appearing in the object file, executable file or  
25718 object-file library named by *file*. If no symbolic information is available for a valid input file, the  
25719 *nm* utility shall report that fact, but not consider it an error condition.

25720 XSI The default base used when numeric values are written is unspecified. On XSI-conformant  
25721 systems, it shall be decimal.

## 25722 OPTIONS

25723 The *nm* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section  
25724 12.2, Utility Syntax Guidelines.

25725 The following options shall be supported:

25726 **-A** Write the full pathname or library name of an object on each line.

25727 XSI **-e** Write only external (global) and static symbol information.

25728 XSI **-f** Produce full output. Write redundant symbols (**.text**, **.data**, and **.bss**), normally  
25729 suppressed.

25730 **-g** Write only external (global) symbol information.

25731 XSI **-o** Write numeric values in octal (equivalent to **-t o**).

25732 **-P** Write information in a portable output format, as specified in the STDOUT section.

25733 **-t *format*** Write each numeric value in the specified format. The format shall be dependent  
25734 on the single character used as the *format* option-argument:

25735 XSI d The offset is written in decimal (default).

25736 o The offset is written in octal.

25737 x The offset is written in hexadecimal.

25738 **-u** Write only undefined symbols.

25739 **-v** Sort output by value instead of alphabetically.

25740 XSI **-x** Write numeric values in hexadecimal (equivalent to **-t x**).

## 25741 OPERANDS

25742 The following operand shall be supported:

25743 *file* A pathname of an object file, executable file, or object-file library.

## 25744 STDIN

25745 See the INPUT FILES section.

25746 **INPUT FILES**

25747 The input file shall be an object file, an object-file library whose format is the same as those  
 25748 produced by the *ar* utility for link editing, or an executable file. The *nm* utility may accept  
 25749 additional implementation-defined object library formats for the input file.

25750 **ENVIRONMENT VARIABLES**

25751 The following environment variables shall affect the execution of *nm*:

25752 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 25753 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
 25754 Internationalization Variables for the precedence of internationalization variables  
 25755 used to determine the values of locale categories.)

25756 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 25757 internationalization variables.

25758 *LC\_COLLATE*

25759 Determine the locale for character collation information for the symbol-name and  
 25760 symbol-value collation sequences.

25761 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 25762 characters (for example, single-byte as opposed to multi-byte characters in  
 25763 arguments).

25764 *LC\_MESSAGES*

25765 Determine the locale that should be used to affect the format and contents of  
 25766 diagnostic messages written to standard error.

25767 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

25768 **ASYNCHRONOUS EVENTS**

25769 Default.

25770 **STDOUT**

25771 If symbolic information is present in the input files, then for each file or for each member of an  
 25772 archive, the *nm* utility shall write the following information to standard output. By default, the  
 25773 format is unspecified, but the output shall be sorted alphabetically by symbol name:

- 25774 • Library or object name, if *-A* is specified
- 25775 • Symbol name
- 25776 • Symbol type, which shall either be one of the following single characters or an  
 25777 implementation-defined type represented by a single character:
  - 25778 A Global absolute symbol.
  - 25779 a Local absolute symbol.
  - 25780 B Global “bss” (that is, uninitialized data space) symbol.
  - 25781 b Local bss symbol.
  - 25782 D Global data symbol.
  - 25783 d Local data symbol.
  - 25784 T Global text symbol.
  - 25785 t Local text symbol.
  - 25786 U Undefined symbol.

- 25787           • Value of the symbol
- 25788           • The size associated with the symbol, if applicable
- 25789           This information may be supplemented by additional information specific to the  
25790           implementation.
- 25791           If the **-P** option is specified, the previous information shall be displayed using the following  
25792           portable format. The three versions differ depending on whether **-t d**, **-t o**, or **-t x** was specified,  
25793           respectively:
- 25794           "*%s%s %s %d %d\n*", *<library/object name>*, *<name>*, *<type>*,  
25795           *<value>*, *<size>*
- 25796           "*%s%s %s %o %o\n*", *<library/object name>*, *<name>*, *<type>*,  
25797           *<value>*, *<size>*
- 25798           "*%s%s %s %x %x\n*", *<library/object name>*, *<name>*, *<type>*,  
25799           *<value>*, *<size>*
- 25800           where
- 25801           *<library/object name>* shall be formatted as follows:
- 25802           • If **-A** is not specified, *<library/object name>* shall be an empty string.
- 25803           • If **-A** is specified and the corresponding *file* operand does not name a library:
- 25804           *"%s: ", <file>*
- 25805           • If **-A** is specified and the corresponding *file* operand names a library. In this case, *<object file>*  
25806           shall name the object file in the library containing the symbol being described:
- 25807           *"%s[%s]: ", <file>, <object file>*
- 25808           If **-A** is not specified, then if more than one *file* operand is specified or if only one *file* operand is  
25809           specified and it names a library, *nm* shall write a line identifying the object containing the  
25810           following symbols before the lines containing those symbols, in the form:
- 25811           • If the corresponding *file* operand does not name a library:
- 25812           *"%s:\n", <file>*
- 25813           • If the corresponding *file* operand names a library; in this case, *<object file>* shall be the name  
25814           of the file in the library containing the following symbols:
- 25815           *"%s[%s]:\n", <file>, <object file>*
- 25816           If **-P** is specified, but **-t** is not, the format shall be as if **-t x** had been specified.
- 25817   **STDERR**
- 25818           The standard error shall be used only for diagnostic messages. |
- 25819   **OUTPUT FILES**
- 25820           None.
- 25821   **EXTENDED DESCRIPTION**
- 25822           None.
- 25823   **EXIT STATUS**
- 25824           The following exit values shall be returned:
- 25825           0   Successful completion.

25826 >0 An error occurred.

#### 25827 CONSEQUENCES OF ERRORS

25828 Default.

#### 25829 APPLICATION USAGE

25830 Mechanisms for dynamic linking make this utility less meaningful when applied to an  
25831 executable file because a dynamically linked executable may omit numerous library routines  
25832 that would be found in a statically linked executable.

#### 25833 EXAMPLES

25834 None.

#### 25835 RATIONALE

25836 Historical implementations of *nm* have used different bases for numeric output and supplied  
25837 different default types of symbols that were reported. The *-t format* option, similar to that used  
25838 in *od* and *strings*, can be used to specify the numeric base; *-g* and *-u* can be used to restrict the  
25839 amount of output or the types of symbols included in the output.

25840 The compromise of using *-t format versus* using *-d*, *-o*, and other similar options was necessary  
25841 because of differences in the meaning of *-o* between implementations. The *-o* option from BSD  
25842 has been provided here as *-A* to avoid confusion with the *-o* from System V (which has been  
25843 provided here as *-t* and as *-o* on XSI-conformant systems).

25844 The option list was significantly reduced from that provided by historical implementations.

25845 The *nm* description is a subset of both the System V and BSD *nm* utilities with no specified  
25846 default output.

25847 It was recognized that mechanisms for dynamic linking make this utility less meaningful when  
25848 applied to an executable file (because a dynamically linked executable file may omit numerous  
25849 library routines that would be found in a statically linked executable file), but the value of *nm*  
25850 during software development was judged to outweigh other limitations.

25851 The default output format of *nm* is not specified because of differences in historical  
25852 implementations. The *-P* option was added to allow some type of portable output format. After  
25853 a comparison of the different formats used in SunOS, BSD, SVR3, and SVR4, it was decided to  
25854 create one that did not match the current format of any of these four systems. The format  
25855 devised is easy to parse by humans, easy to parse in shell scripts, and does not need to vary  
25856 depending on locale (because no English descriptions are included). All of the systems currently  
25857 have the information available to use this format.

25858 The format given in *nm* STDOUT uses spaces between the fields, which may be any number of  
25859 <blank>s required to align the columns. The single-character types were selected to match  
25860 historical practice, and the requirement that implementation additions also be single characters  
25861 made parsing the information easier for shell scripts.

#### 25862 FUTURE DIRECTIONS

25863 None.

#### 25864 SEE ALSO

25865 *ar*, *c99*

#### 25866 CHANGE HISTORY

25867 First released in Issue 2.

25868 **Issue 6**

25869

25870

This utility is now marked as supported when both the User Portability Utilities option and the Software Development Utilities option are supported.

25871 **NAME**

25872           nohup — invoke a utility immune to hangups

25873 **SYNOPSIS**25874           nohup *utility* [*argument...*]25875 **DESCRIPTION**

25876           The *nohup* utility shall invoke the utility named by the *utility* operand with arguments supplied  
 25877 as the *argument* operands. At the time the named *utility* is invoked, the SIGHUP signal shall be  
 25878 set to be ignored.

25879           If the standard output is a terminal, all output written by the named *utility* to its standard output  
 25880 shall be appended to the end of the file **nohup.out** in the current directory. If **nohup.out** cannot  
 25881 be created or opened for appending, the output shall be appended to the end of the file  
 25882 **nohup.out** in the directory specified by the *HOME* environment variable. If neither file can be  
 25883 created or opened for appending, *utility* shall not be invoked. If a file is created, the file's  
 25884 permission bits shall be set to S\_IRUSR | S\_IWUSR.

25885           If the standard error is a terminal, all output written by the named *utility* to its standard error  
 25886 shall be redirected to the same file descriptor as the standard output.

25887 **OPTIONS**

25888           None.

25889 **OPERANDS**

25890           The following operands shall be supported:

25891           *utility*       The name of a utility that is to be invoked. If the *utility* operand names any of the  
 25892 special built-in utilities in Section 2.14 (on page 2266), the results are undefined.

25893           *argument*     Any string to be supplied as an argument when invoking the utility named by the  
 25894 *utility* operand.

25895 **STDIN**

25896           Not used.

25897 **INPUT FILES**

25898           None.

25899 **ENVIRONMENT VARIABLES**25900           The following environment variables shall affect the execution of *nohup*:

25901           *HOME*         Determine the pathname of the user's home directory: if the output file **nohup.out**  
 25902 cannot be created in the current directory, the *nohup* utility shall use the directory  
 25903 named by *HOME* to create the file.

25904           *LANG*         Provide a default value for the internationalization variables that are unset or null.  
 25905 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
 25906 Internationalization Variables for the precedence of internationalization variables  
 25907 used to determine the values of locale categories.)

25908           *LC\_ALL*        If set to a non-empty string value, override the values of all the other  
 25909 internationalization variables.

25910           *LC\_CTYPE*    Determine the locale for the interpretation of sequences of bytes of text data as  
 25911 characters (for example, single-byte as opposed to multi-byte characters in  
 25912 arguments).

25913           *LC\_MESSAGES*

25914           Determine the locale that should be used to affect the format and contents of

- 25915 diagnostic messages written to standard error.
- 25916 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 25917 **PATH** Determine the search path that is used to locate the utility to be invoked. See the  
25918 Base Definitions volume of IEEE Std 1003.1-200x, Chapter 8, Environment  
25919 Variables.
- 25920 **ASYNCHRONOUS EVENTS**
- 25921 The *nohup* utility shall take the standard action for all signals except that *SIGHUP* shall be  
25922 ignored.
- 25923 **STDOUT**
- 25924 If the standard output is not a terminal, the standard output of *nohup* shall be the standard  
25925 output generated by the execution of the *utility* specified by the operands. Otherwise, nothing  
25926 shall be written to the standard output.
- 25927 **STDERR**
- 25928 If the standard output is a terminal, a message shall be written to the standard error, indicating  
25929 the name of the file to which the output is being appended. The name of the file shall be either  
25930 **nohup.out** or **\$HOME/nohup.out**.
- 25931 **OUTPUT FILES**
- 25932 If the standard output is a terminal, all output written by the named *utility* to the standard  
25933 output and standard error is appended to the file **nohup.out**, which is created if it does not  
25934 already exist.
- 25935 **EXTENDED DESCRIPTION**
- 25936 None.
- 25937 **EXIT STATUS**
- 25938 The following exit values shall be returned:
- 25939 126 The utility specified by *utility* was found but could not be invoked.
- 25940 127 An error occurred in the *nohup* utility or the utility specified by *utility* could not be  
25941 found.
- 25942 Otherwise, the exit status of *nohup* shall be that of the utility specified by the *utility* operand.
- 25943 **CONSEQUENCES OF ERRORS**
- 25944 Default.
- 25945 **APPLICATION USAGE**
- 25946 The *command*, *env*, *nice*, *nohup*, *time*, and *xargs* utilities have been specified to use exit code 127 if  
25947 an error occurs so that applications can distinguish “failure to find a utility” from “invoked  
25948 utility exited with an error indication”. The value 127 was chosen because it is not commonly  
25949 used for other meanings; most utilities use small values for “normal error conditions” and the  
25950 values above 128 can be confused with termination due to receipt of a signal. The value 126 was  
25951 chosen in a similar manner to indicate that the utility could be found, but not invoked. Some  
25952 scripts produce meaningful error messages differentiating the 126 and 127 cases. The distinction  
25953 between exit codes 126 and 127 is based on KornShell practice that uses 127 when all attempts to  
25954 *exec* the utility fail with [ENOENT], and uses 126 when any attempt to *exec* the utility fails for  
25955 any other reason.
- 25956 **EXAMPLES**
- 25957 It is frequently desirable to apply *nohup* to pipelines or lists of commands. This can be done by  
25958 placing pipelines and command lists in a single file; this file can then be invoked as a utility, and  
25959 the *nohup* applies to everything in the file.

25960 Alternatively, the following command can be used to apply *nohup* to a complex command:

```
25961 nohup sh -c 'complex-command-line'
```

25962 **RATIONALE**

25963 The 4.3 BSD version ignores SIGTERM and SIGHUP, and if **./nohup.out** cannot be used, it fails  
25964 instead of trying to use **\$HOME/nohup.out**.

25965 The *cs* utility has a built-in version of *nohup* that acts differently from the POSIX Shell and  
25966 Utilities *nohup*.

25967 The term *utility* is used, rather than *command*, to highlight the fact that shell compound  
25968 commands, pipelines, special built-ins, and so on, cannot be used directly. However, *utility*  
25969 includes user application programs and shell scripts, not just the standard utilities.

25970 Historical versions of the *nohup* utility use default file creation semantics. Some more recent  
25971 versions use the permissions specified here as an added security precaution.

25972 Some historical implementations ignore SIGQUIT in addition to SIGHUP; others ignore  
25973 SIGTERM. An early proposal allowed, but did not require, SIGQUIT to be ignored. Several  
25974 reviewers objected that *nohup* should only modify the handling of SIGHUP as required by this  
25975 volume of IEEE Std 1003.1-200x.

25976 **FUTURE DIRECTIONS**

25977 None.

25978 **SEE ALSO**

25979 *sh*, the System Interfaces volume of IEEE Std 1003.1-200x, *signal()*

25980 **CHANGE HISTORY**

25981 First released in Issue 2.

## 25982 NAME

25983 od — dump files in various formats

## 25984 SYNOPSIS

25985 od [-v][-A *address\_base*][-j *skip*][-N *count*][-t *type\_string*].  
 25986 [*file*...]

25987 XSI od [-bcdosx][*file*] [[+]offset[.][b]]

25988

## 25989 DESCRIPTION

25990 The *od* utility shall write the contents of its input files to standard output in a user-specified  
 25991 format.

## 25992 OPTIONS

25993 The *od* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2,  
 25994 XSI Utility Syntax Guidelines, except that the order of presentation of the **-t** options and the  
 25995 **-bcdosx** options is significant.

25996 The following options shall be supported:

25997 **-A** *address\_base*

25998 Specify the input offset base. See the EXTENDED DESCRIPTION section. The  
 25999 application shall ensure that the *address\_base* option-argument is a character. The  
 26000 characters 'd', 'o', and 'x' specify that the offset base shall be written in  
 26001 decimal, octal, or hexadecimal, respectively. The character 'n' specifies that the  
 26002 offset shall not be written.

26003 XSI **-b** Interpret bytes in octal. This shall be equivalent to **-t o1**.

26004 XSI **-c** Interpret bytes as characters specified by the current setting of the *LC\_CTYPE*  
 26005 category. Certain non-graphic characters appear as C escapes: "NUL=\0",  
 26006 "BS=\b", "FF=\f", "NL=\n", "CR=\r", "HT=\t"; others appear as 3-digit octal  
 26007 numbers.

26008 XSI **-d** Interpret *words* (two-byte units) in unsigned decimal. This shall be equivalent to  
 26009 **-t u2**.

26010 **-j** *skip* Jump over *skip* bytes from the beginning of the input. The *od* utility shall read or  
 26011 seek past the first *skip* bytes in the concatenated input files. If the combined input  
 26012 is not at least *skip* bytes long, the *od* utility shall write a diagnostic message to  
 26013 standard error and exit with a non-zero exit status.

26014 By default, the *skip* option-argument shall be interpreted as a decimal number.  
 26015 With a leading 0x or 0X, the offset shall be interpreted as a hexadecimal number;  
 26016 otherwise, with a leading '0', the offset shall be interpreted as an octal number.  
 26017 Appending the character 'b', 'k', or 'm' to offset shall cause it to be interpreted  
 26018 as a multiple of 512, 1024, or 1048576 bytes, respectively. If the *skip* number is  
 26019 hexadecimal, any appended 'b' shall be considered to be the final hexadecimal  
 26020 digit.

26021 **-N** *count* Format no more than *count* bytes of input. By default, *count* shall be interpreted as  
 26022 a decimal number. With a leading 0x or 0X, *count* shall be interpreted as a  
 26023 hexadecimal number; otherwise, with a leading '0', it shall be interpreted as an  
 26024 octal number. If *count* bytes of input (after successfully skipping, if **-j** *skip*  
 26025 is specified) are not available, it shall not be considered an error; the *od* utility shall  
 26026 format the input that is available.

|           |                                                |                                                                                                                 |  |
|-----------|------------------------------------------------|-----------------------------------------------------------------------------------------------------------------|--|
| 26027 XSI | <b>-o</b>                                      | Interpret <i>words</i> (two-byte units) in octal. This shall be equivalent to <b>-t o2</b> .                    |  |
| 26028 XSI | <b>-s</b>                                      | Interpret <i>words</i> (two-byte units) in signed decimal. This shall be equivalent to                          |  |
| 26029     |                                                | <b>-t d2</b> .                                                                                                  |  |
| 26030     | <b>-t</b> <i>type_string</i>                   |                                                                                                                 |  |
| 26031     |                                                | Specify one or more output types. See the EXTENDED DESCRIPTION section. The                                     |  |
| 26032     |                                                | application shall ensure that the <i>type_string</i> option-argument is a string specifying                     |  |
| 26033     |                                                | the types to be used when writing the input data. The string shall consist of the                               |  |
| 26034     |                                                | type specification characters a, c, d, f, o, u, and x, specifying named character,                              |  |
| 26035     |                                                | character, signed decimal, floating point, octal, unsigned decimal, and                                         |  |
| 26036     |                                                | hexadecimal, respectively. The type specification characters d, f, o, u, and x can be                           |  |
| 26037     |                                                | followed by an optional unsigned decimal integer that specifies the number of                                   |  |
| 26038     |                                                | bytes to be transformed by each instance of the output type. The type specification                             |  |
| 26039     |                                                | character f can be followed by an optional F, D, or L indicating that the conversion                            |  |
| 26040     |                                                | should be applied to an item of type <b>float</b> , <b>double</b> , or <b>long double</b> , respectively.       |  |
| 26041     |                                                | The type specification characters d, o, u and x can be followed by an optional C, S,                            |  |
| 26042     |                                                | I, or L indicating that the conversion should be applied to an item of type <b>char</b> ,                       |  |
| 26043     |                                                | <b>short</b> , <b>int</b> , or <b>long</b> , respectively. Multiple types can be concatenated within the        |  |
| 26044     |                                                | same <i>type_string</i> and multiple <b>-t</b> options can be specified. Output lines shall be                  |  |
| 26045     |                                                | written for each type specified in the order in which the type specification                                    |  |
| 26046     |                                                | characters are specified.                                                                                       |  |
| 26047     | <b>-v</b>                                      | Write all input data. Without the <b>-v</b> option, any number of groups of output lines,                       |  |
| 26048     |                                                | which would be identical to the immediately preceding group of output lines                                     |  |
| 26049     |                                                | (except for the byte offsets), shall be replaced with a line containing only an                                 |  |
| 26050     |                                                | asterisk ( ' * ' ).                                                                                             |  |
| 26051 XSI | <b>-x</b>                                      | Interpret <i>words</i> (two-byte units) in hexadecimal. This shall be equivalent to <b>-t x2</b> .              |  |
| 26052 XSI |                                                | Multiple types can be specified by using multiple <b>-bcdostx</b> options. Output lines are written for         |  |
| 26053     |                                                | each type specified in the order in which the types are specified.                                              |  |
| 26054     | <b>OPERANDS</b>                                |                                                                                                                 |  |
| 26055     |                                                | The following operands shall be supported:                                                                      |  |
| 26056     | <i>file</i>                                    | A pathname of a file to be read. If no <i>file</i> operands are specified, the standard input                   |  |
| 26057     |                                                | shall be used.                                                                                                  |  |
| 26058     |                                                | If there are no more than two operands, none of the <b>-A</b> , <b>-j</b> , <b>-N</b> , or <b>-t</b> options is |  |
| 26059     |                                                | specified, and either of the following is true: the first character of the last operand                         |  |
| 26060     |                                                | is a plus sign ( ' + ' ), or there are two operands and the first character of the last                         |  |
| 26061 XSI |                                                | operand is numeric; the last operand shall be interpreted as an offset operand on                               |  |
| 26062     |                                                | XSI-conformant systems. Under these conditions, the results are unspecified on                                  |  |
| 26063     |                                                | systems that are not XSI-conformant systems.                                                                    |  |
| 26064 XSI | <b>[+]</b> <i>offset</i> <b>[.]</b> <b>[b]</b> | The <i>offset</i> operand specifies the offset in the file where dumping is to commence.                        |  |
| 26065     |                                                | This operand is normally interpreted as octal bytes. If ' . ' is appended, the offset                           |  |
| 26066     |                                                | shall be interpreted in decimal. If ' b ' is appended, the offset shall be interpreted                          |  |
| 26067     |                                                | in units of 512 bytes.                                                                                          |  |
| 26068     | <b>STDIN</b>                                   |                                                                                                                 |  |
| 26069     |                                                | The standard input shall be used only if no <i>file</i> operands are specified. See the INPUT FILES             |  |
| 26070     |                                                | section.                                                                                                        |  |

26071 **INPUT FILES**

26072           The input files can be any file type.

26073 **ENVIRONMENT VARIABLES**

26074           The following environment variables shall affect the execution of *od*:

26075           **LANG**       Provide a default value for the internationalization variables that are unset or null.  
 26076                       (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
 26077                       Internationalization Variables for the precedence of internationalization variables  
 26078                       used to determine the values of locale categories.)

26079           **LC\_ALL**     If set to a non-empty string value, override the values of all the other  
 26080                       internationalization variables.

26081           **LC\_CTYPE**   Determine the locale for the interpretation of sequences of bytes of text data as  
 26082                       characters (for example, single-byte as opposed to multi-byte characters in  
 26083                       arguments and input files).

26084           **LC\_MESSAGES**

26085                       Determine the locale that should be used to affect the format and contents of  
 26086                       diagnostic messages written to standard error.

26087           **LC\_NUMERIC**

26088                       Determine the locale for selecting the radix character used when writing floating-  
 26089                       point formatted output.

26090 **XSI**       **NLSPATH**   Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

26091 **ASYNCHRONOUS EVENTS**

26092           Default.

26093 **STDOUT**

26094           See the EXTENDED DESCRIPTION section.

26095 **STDERR**

26096           The standard error shall be used only for diagnostic messages. |

26097 **OUTPUT FILES**

26098           None.

26099 **EXTENDED DESCRIPTION**

26100           The *od* utility shall copy sequentially each input file to standard output, transforming the input  
 26101 **XSI**           data according to the output types specified by the **-t** options or the **-bcdox** options. If no  
 26102           output type is specified, the default output shall be as if **-t oS** had been specified.

26103           The number of bytes transformed by the output type specifier *c* may be variable depending on |  
 26104           the *LC\_CTYPE* category.

26105           The default number of bytes transformed by output type specifiers *d*, *f*, *o*, *u*, and *x* corresponds |  
 26106           to the various C-language types as follows. If the *c99* compiler is present on the system, these  
 26107           specifiers shall correspond to the sizes used by default in that compiler. Otherwise, these sizes  
 26108           may vary among systems that conform to IEEE Std 1003.1-200x.

26109           

- For the type specifier characters *d*, *o*, *u*, and *x*, the default number of bytes shall correspond |  
 26110           to the size of the underlying implementation's basic integer type. For these specifier  
 26111           characters, the implementation shall support values of the optional number of bytes to be  
 26112           converted corresponding to the number of bytes in the C-language types **char**, **short**, **int**, and  
 26113           **long**. These numbers can also be specified by an application as the characters '*C*', '*S*', '*I*',  
 26114           and '*L*', respectively. The implementation shall also support the values 1, 2, 4, and 8, even if  
 26115           it provides no C-Language types of those sizes. The implementation shall support the

26116 decimal value corresponding to the C-language type **long long**. The byte order used when  
 26117 interpreting numeric values is implementation-defined, but shall correspond to the order in  
 26118 which a constant of the corresponding type is stored in memory on the system.

26119 • For the type specifier character  $\epsilon$ , the default number of bytes shall correspond to the number  
 26120 of bytes in the underlying implementation's basic double precision floating-point data type.  
 26121 The implementation shall support values of the optional number of bytes to be converted  
 26122 corresponding to the number of bytes in the C-language types **float**, **double**, and **long**  
 26123 **double**. These numbers can also be specified by an application as the characters 'F', 'D',  
 26124 and 'L', respectively.

26125 The type specifier character  $\alpha$  specifies that bytes shall be interpreted as named characters from  
 26126 the International Reference Version (IRV) of the ISO/IEC 646:1991 standard. Only the least  
 26127 significant seven bits of each byte shall be used for this type specification. Bytes with the values  
 26128 listed in the following table shall be written using the corresponding names for those characters.

26129 **Table 4-12** Named Characters in *od*

| Value | Name | Value | Name | Value | Name      | Value | Name |
|-------|------|-------|------|-------|-----------|-------|------|
| \000  | nul  | \001  | soh  | \002  | stx       | \003  | etx  |
| \004  | eot  | \005  | enq  | \006  | ack       | \007  | bel  |
| \010  | bs   | \011  | ht   | \012  | lf or nl* | \013  | vt   |
| \014  | ff   | \015  | cr   | \016  | so        | \017  | si   |
| \020  | dle  | \021  | dc1  | \022  | dc2       | \023  | dc3  |
| \024  | dc4  | \025  | nak  | \026  | syn       | \027  | etb  |
| \030  | can  | \031  | em   | \032  | sub       | \033  | esc  |
| \034  | fs   | \035  | gs   | \036  | rs        | \037  | us   |
| \040  | sp   | \177  | del  |       |           |       |      |

26141 **Note:** The "\012" value may be written either as lf or nl.

26142 The type specifier character  $\epsilon$  specifies that bytes shall be interpreted as characters specified by  
 26143 the current setting of the *LC\_CTYPE* locale category. Characters listed in the table in the Base  
 26144 Definitions volume of IEEE Std 1003.1-200x, Chapter 5, File Format Notation ('\\', '\a', '\b',  
 26145 '\f', '\n', '\r', '\t', '\v') shall be written as the corresponding escape sequences, except  
 26146 that backslash shall be written as a single backslash and a NUL shall be written as '\0'. Other  
 26147 non-printable characters shall be written as one three-digit octal number for each byte in the  
 26148 character. If the size of a byte on the system is greater than nine bits, the format used for non-  
 26149 printable characters is implementation-defined. Printable multi-byte characters shall be written  
 26150 in the area corresponding to the first byte of the character; the two-character sequence "\*\*\*"  
 26151 shall be written in the area corresponding to each remaining byte in the character, as an  
 26152 indication that the character is continued. When either the  $-j$  *skip* or  $-N$  *count* option is specified  
 26153 along with the  $\epsilon$  type specifier, and this results in an attempt to start or finish in the middle of a  
 26154 multi-byte character, the result is implementation-defined.

26155 The input data shall be manipulated in blocks, where a block is defined as a multiple of the least  
 26156 common multiple of the number of bytes transformed by the specified output types. If the least  
 26157 common multiple is greater than 16, the results are unspecified. Each input block shall be  
 26158 written as transformed by each output type, one per written line, in the order that the output  
 26159 types were specified. If the input block size is larger than the number of bytes transformed by  
 26160 the output type, the output type shall sequentially transform the parts of the input block, and  
 26161 the output from each of the transformations shall be separated by one or more <blank>s.

26162 If, as a result of the specification of the `-N` option or end-of-file being reached on the last input  
 26163 file, input data only partially satisfies an output type, the input shall be extended sufficiently  
 26164 with null bytes to write the last byte of the input.

26165 Unless `-A n` is specified, the first output line produced for each input block shall be preceded by  
 26166 the input offset, cumulative across input files, of the next byte to be written. The format of the  
 26167 input offset is unspecified; however, it shall not contain any `<blank>`s, shall start at the first  
 26168 character of the output line, and shall be followed by one or more `<blank>`s. In addition, the  
 26169 offset of the byte following the last byte written shall be written after all the input data has been  
 26170 processed, but shall not be followed by any `<blank>`s.

26171 If no `-A` option is specified, the input offset base is unspecified.

#### 26172 EXIT STATUS

26173 The following exit values shall be returned:

26174 0 All input files were processed successfully.

26175 >0 An error occurred.

#### 26176 CONSEQUENCES OF ERRORS

26177 Default.

#### 26178 APPLICATION USAGE

26179 XSI-conformant applications are warned not to use filenames starting with `'+'` or a first  
 26180 operand starting with a numeric character so that the old functionality can be maintained by  
 26181 implementations, unless they specify one of the `-A`, `-j`, or `-N` options. To guarantee that one of  
 26182 these filenames is always interpreted as a filename, an application could always specify the  
 26183 address base format with the `-A` option.

#### 26184 EXAMPLES

26185 If a file containing 128 bytes with decimal values zero to 127, in increasing order, is supplied as  
 26186 standard input to the command:

```
26187 od -A d -t a
```

26188 on an implementation using an input block size of 16 bytes, the standard output, independent of  
 26189 the current locale setting, would be similar to:

```
26190 0000000 nul soh stx etx eot enq ack bel bs ht nl vt ff cr so si
26191 0000016 dle dc1 dc2 dc3 dc4 nak syn etb can em sub esc fs gs rs us
26192 0000032 sp ! " # $ % & ' ( ) * + , - . /
26193 0000048 0 1 2 3 4 5 6 7 8 9 : ; < = > ?
26194 0000064 @ A B C D E F G H I J K L M N O
26195 0000080 P Q R S T U V W X Y Z [ \ ] ^ _
26196 0000096 ` a b c d e f g h i j k l m n o
26197 0000112 p q r s t u v w x y z { | } ~ del
26198 0000128
```

26199 Note that this volume of IEEE Std 1003.1-200x allows `nl` or `lf` to be used as the name for the  
 26200 ISO/IEC 646:1991 standard IRV character with decimal value 10. The IRV names this character  
 26201 `lf` (line feed), but traditional implementations have referred to this character as newline (`nl`) and  
 26202 the POSIX locale character set symbolic name for the corresponding character is a `<newline>`.

26203 The command:

```
26204 od -A o -t o2x2x -n 18
```

26205 on a system with 32-bit words and an implementation using an input block size of 16 bytes  
 26206 could write 18 bytes in approximately the following format:

```

26207      0000000 032056 031440 041123 042040 052516 044530 020043 031464
26208              342e 3320 4253 4420 554e 4958 2023 3334
26209              342e3320 42534420 554e4958 20233334
26210      0000020 032472
26211              353a
26212              353a0000
26213      0000022

```

26214 The command:

```
26215 od -A d -t f -t o4 -t x4 -n 24 -j 0x15
```

26216 on a system with 64-bit doubles (for example, IEEE Std 754-1985 double precision floating-point  
26217 format) would skip 21 bytes of input data and then write 24 bytes in approximately the  
26218 following format:

```

26219      0000000 1.0000000000000000e+00 1.5735000000000000e+01
26220              07774000000 00000000000 10013674121 35341217270
26221              3ff00000 00000000 402f3851 eb851eb8
26222      0000016 1.4066823000000000e+02
26223              10030312542 04370303230
26224              40619562 23e18698
26225      0000024

```

#### 26226 RATIONALE

26227 The *od* utility went through several names in early proposals, including *hd*, *xd*, and most recently  
26228 *hexdump*. There were several objections to all of these based on the following reasons:

- 26229 • The *hd* and *xd* names conflicted with historical utilities that behaved differently.
- 26230 • The *hexdump* description was much more complex than needed for a simple dump utility.
- 26231 • The *od* utility has been available on all historical implementations and there was no need to  
26232 create a new name for a utility so similar to the historical *od* utility.

26233 The original reasons for not standardizing historical *od* were also fairly widespread. Those  
26234 reasons are given below along with rationale explaining why the standard developers believe  
26235 that this version does not suffer from the indicated problem:

- 26236 • The BSD and System V versions of *od* have diverged, and the intersection of features  
26237 provided by both does not meet the needs of the user community. In fact, the System V  
26238 version only provides a mechanism for dumping octal bytes and **shorts**, signed and unsigned  
26239 decimal **shorts**, hexadecimal **shorts**, and ASCII characters. BSD added the ability to dump  
26240 **floats**, **doubles**, named ASCII characters, and octal, signed decimal, unsigned decimal, and  
26241 hexadecimal **longs**. The version presented here provides more normalized forms for  
26242 dumping bytes, **shorts**, **ints**, and **longs** in octal, signed decimal, unsigned decimal, and  
26243 hexadecimal; **float**, **double**, and **long double**; and named ASCII as well as current locale  
26244 characters.
- 26245 • It would not be possible to come up with a compatible superset of the BSD and System V  
26246 flags that met the requirements of the standard developers. The historical default *od* output is  
26247 the specified default output of this utility. None of the option letters chosen for this version  
26248 of *od* conflict with any of the options to historical versions of *od*.
- 26249 • On systems with different sizes for **short**, **int**, and **long**, there was no way to ask for dumps  
26250 of **ints**, even in the BSD version. Because of the way options are named, the name space  
26251 could not be extended to solve these problems. This is why the **-t** option was added (with  
26252 type specifiers more closely matched to the *printf()* formats used in the rest of this volume of

26253 IEEE Std 1003.1-200x) and the optional field sizes were added to the `d`, `f`, `o`, `u`, and `x` type  
 26254 specifiers. It is also one of the reasons why the historical practice was not mandated as a  
 26255 required obsolescent form of `od`. (Although the old versions of `od` are not listed as an  
 26256 obsolescent form, implementations are urged to continue to recognize the older forms for  
 26257 several more years.) The `a`, `c`, `f`, `o`, and `x` types match the meaning of the corresponding  
 26258 format characters in the historical implementations of `od` except for the default sizes of the  
 26259 fields converted. The `d` format is signed in this volume of IEEE Std 1003.1-200x to match the  
 26260 `printf()` notation. (Historical versions of `od` used `d` as a synonym for `u` in this version. The  
 26261 System V implementation uses `s` for signed decimal; BSD uses `i` for signed decimal and `s` for  
 26262 null-terminated strings.) Other than `d` and `u`, all of the type specifiers match format  
 26263 characters in the historical BSD version of `od`.

26264 The sizes of the C-language types **char**, **short**, **int**, **long**, **float**, **double**, and **long double** are  
 26265 used even though it is recognized that there may be zero or more than one compiler for the C  
 26266 language on an implementation and that they may use different sizes for some of these types.  
 26267 (For example, one compiler might use 2 bytes **shorts**, 2 bytes **ints**, and 4 bytes **longs**, while  
 26268 another compiler (or an option to the same compiler) uses 2 bytes **shorts**, 4 bytes **ints**, and 4  
 26269 bytes **longs**.) Nonetheless, there has to be a basic size known by the implementation for  
 26270 these types, corresponding to the values reported by invocations of the `getconf` utility when  
 26271 called with `system_var` operands {`UCHAR_MAX`}, {`USHORT_MAX`}, {`UINT_MAX`}, and  
 26272 {`ULONG_MAX`} for the types **char**, **short**, **int**, and **long**, respectively. There are similar  
 26273 constants required by the ISO C standard, but not required by the System Interfaces volume  
 26274 of IEEE Std 1003.1-200x or this volume of IEEE Std 1003.1-200x. They are {`FLT_MANT_DIG`},  
 26275 {`DBL_MANT_DIG`}, and {`LDBL_MANT_DIG`} for the types **float**, **double**, and **long double**,  
 26276 respectively. If the optional `c99` utility is provided by the implementation and used as  
 26277 specified by this volume of IEEE Std 1003.1-200x, these are the sizes that would be provided.  
 26278 If an option is used that specifies different sizes for these types, there is no guarantee that the  
 26279 `od` utility is able to interpret binary data output by such a program correctly.

26280 This volume of IEEE Std 1003.1-200x requires that the numeric values of these lengths be  
 26281 recognized by the `od` utility and that symbolic forms also be recognized. Thus, a conforming  
 26282 application can always look at an array of **unsigned long** data elements using `od -t uL`.

- 26283 • The method of specifying the format for the address field based on specifying a starting  
 26284 offset in a file unnecessarily tied the two together. The `-A` option now specifies the address  
 26285 base and the `-S` option specifies a starting offset.

- 26286 • It would be difficult to break the dependence on U.S. ASCII to achieve an internationalized  
 26287 utility. It does not seem to be any harder for `od` to dump characters in the current locale than  
 26288 it is for the `ed` or `sed` `l` commands. The `c` type specifier does this without difficulty and is  
 26289 completely compatible with the historical implementations of the `c` format character when  
 26290 the current locale uses a superset of the ISO/IEC 646:1991 standard as a codeset. The `a` type  
 26291 specifier (from the BSD `a` format character) was left as a portable means to dump ASCII (or  
 26292 more correctly ISO/IEC 646:1991 standard (IRV)) so that headers produced by `pax` could be  
 26293 deciphered even on systems that do not use the ISO/IEC 646:1991 standard as a subset of  
 26294 their base codeset.

26295 The use of `"**"` as an indication of continuation of a multi-byte character in `c` specifier output  
 26296 was chosen based on seeing an implementation that uses this method. The continuation bytes  
 26297 have to be marked in a way that is not ambiguous with another single-byte or multi-byte  
 26298 character.

26299 An early proposal used `-S` and `-n`, respectively, for the `-j` and `-N` options eventually selected.  
 26300 These were changed to avoid conflicts with historical implementations.

- 26301 The original standard specified `-t o2` as the default when no output type was given. This was  
26302 changed to `-t oS` (the length of a **short**) to accommodate a supercomputer implementation that  
26303 historically used 64 bits as its default (and that defined shorts as 64 bits). This change should not  
26304 affect conforming applications. The requirement to support lengths of 1, 2, and 4 was added at  
26305 the same time to address an historical implementation that had no two-byte data types in its C  
26306 compiler.
- 26307 The use of a basic integer data type is intended to allow the implementation to choose a word  
26308 size commonly used by applications on that architecture.
- 26309 **FUTURE DIRECTIONS**
- 26310 All option and operand interfaces marked as extensions may be withdrawn in a future issue.
- 26311 **SEE ALSO**
- 26312 *c99, sed*
- 26313 **CHANGE HISTORY**
- 26314 First released in Issue 2.
- 26315 **Issue 5**
- 26316 In the description of the `-c` option, the phrase “This is equivalent to `-t c.`” is deleted.
- 26317 The FUTURE DIRECTIONS section has been modified.
- 26318 **Issue 6**
- 26319 The *od* utility is changed to remove the assumption that **short** was a two-byte entity, as per the  
26320 revisions in the IEEE P1003.2b draft standard.
- 26321 The normative text is reworded to avoid use of the term “must” for application requirements.

## 26322 NAME

26323 paste — merge corresponding or subsequent lines of files

## 26324 SYNOPSIS

26325 paste [-s][-d *list*] *file*...

## 26326 DESCRIPTION

26327 The *paste* utility shall concatenate the corresponding lines of the given input files, and writes the  
26328 resulting lines to standard output.

26329 The default operation of *paste* shall concatenate the corresponding lines of the input files. The  
26330 <newline> of every line except the line from the last input file shall be replaced with a <tab>.

26331 If an end-of-file condition is detected on one or more input files, but not all input files, *paste* shall  
26332 behave as though empty lines were read from the files on which end-of-file was detected, unless  
26333 the **-s** option is specified.

## 26334 OPTIONS

26335 The *paste* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section  
26336 12.2, Utility Syntax Guidelines.

26337 The following options shall be supported:

26338 **-d list** Unless a backslash character appears in *list*, each character in *list* is an element  
26339 specifying a delimiter character. If a backslash character appears in *list*, the  
26340 backslash character and one or more characters following it are an element  
26341 specifying a delimiter character as described below. These elements specify one or  
26342 more delimiters to use, instead of the default <tab>, to replace the <newline> of  
26343 the input lines. The elements in *list* shall be used circularly; that is, when the list is  
26344 exhausted the first element from the list is reused. When the **-s** option is specified:

- 26345 • The last <newline> in a file shall not be modified.
- 26346 • The delimiter shall be reset to the first element of list after each *file* operand is  
26347 processed.

26348 When the **-s** option is not specified:

- 26349 • The <newline>s in the file specified by the last *file* operand shall not be  
26350 modified.
- 26351 • The delimiter shall be reset to the first element of list each time a line is  
26352 processed from each file.

26353 If a backslash character appears in *list*, it and the character following it shall be  
26354 used to represent the following delimiter characters:

26355 \n <newline>.

26356 \t <tab>.

26357 \\ Backslash character.

26358 \0 Empty string (not a null character). If '\0' is immediately followed by the  
26359 character 'x', the character 'X', or any character defined by the *LC\_CTYPE*  
26360 **digit** keyword (see the Base Definitions volume of IEEE Std 1003.1-200x,  
26361 Chapter 7, Locale), the results are unspecified.

26362 If any other characters follow the backslash, the results are unspecified.

26363 **-s** Concatenate all of the lines of each separate input file in command line order. The  
26364 <newline> of every line except the last line in each input file shall be replaced with

26365 the <tab>, unless otherwise specified by the **-d** option.

#### 26366 OPERANDS

26367 The following operand shall be supported:

26368 *file* A pathname of an input file. If *'-'* is specified for one or more of the *files*, the  
 26369 standard input shall be used; the standard input shall be read one line at a time,  
 26370 circularly, for each instance of *'-'*. Implementations shall support pasting of at  
 26371 least 12 *file* operands.

#### 26372 STDIN

26373 The standard input shall be used only if one or more *file* operands is *'-'*. See the INPUT FILES  
 26374 section.

#### 26375 INPUT FILES

26376 The input files shall be text files, except that line lengths shall be unlimited.

#### 26377 ENVIRONMENT VARIABLES

26378 The following environment variables shall affect the execution of *paste*:

26379 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 26380 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
 26381 Internationalization Variables for the precedence of internationalization variables  
 26382 used to determine the values of locale categories.)

26383 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 26384 internationalization variables.

26385 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 26386 characters (for example, single-byte as opposed to multi-byte characters in  
 26387 arguments and input files).

26388 *LC\_MESSAGES*

26389 Determine the locale that should be used to affect the format and contents of  
 26390 diagnostic messages written to standard error.

26391 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

#### 26392 ASYNCHRONOUS EVENTS

26393 Default.

#### 26394 STDOUT

26395 Concatenated lines of input files shall be separated by the <tab> (or other characters under the  
 26396 control of the **-d** option) and terminated by a <newline>.

#### 26397 STDERR

26398 The standard error shall be used only for diagnostic messages.

#### 26399 OUTPUT FILES

26400 None.

#### 26401 EXTENDED DESCRIPTION

26402 None.

#### 26403 EXIT STATUS

26404 The following exit values shall be returned:

26405 0 Successful completion.

26406 >0 An error occurred.

26407 **CONSEQUENCES OF ERRORS**

26408 If one or more input files cannot be opened when the `-s` option is not specified, a diagnostic  
 26409 message shall be written to standard error, but no output is written to standard output. If the `-s`  
 26410 option is specified, the *paste* utility shall provide the default behavior described in Section 1.11  
 26411 (on page 2221).

26412 **APPLICATION USAGE**

26413 When the escape sequences of the *list* option-argument are used in a shell script, they must be  
 26414 quoted; otherwise, the shell treats the `'\'` as a special character.

26415 Conforming applications should only use the specific backslash escaped delimiters presented in  
 26416 this volume of IEEE Std 1003.1-200x. Historical implementations treat `'\x'`, where `'x'` is not in  
 26417 this list, as `'x'`, but future implementations are free to expand this list to recognize other  
 26418 common escapes similar to those accepted by *printf* and other standard utilities.

26419 Most of the standard utilities work on text files. The *cut* utility can be used to turn files with  
 26420 arbitrary line lengths into a set of text files containing the same data. The *paste* utility can be used  
 26421 to create (or recreate) files with arbitrary line lengths. For example, if *file* contains long lines:

```
26422 cut -b 1-500 -n file > file1
26423 cut -b 501- -n file > file2
```

26424 creates **file1** (a text file) with lines no longer than 500 bytes (plus the `<newline>`) and **file2** that  
 26425 contains the remainder of the data from *file*. Note that **file2** is not a text file if there are lines in  
 26426 *file* that are longer than `500 + {LINE_MAX}` bytes. The original file can be recreated from **file1**  
 26427 and **file2** using the command:

```
26428 paste -d "\0" file1 file2 > file
```

26429 The commands:

```
26430 paste -d "\0" ...
26431 paste -d " " ...
```

26432 are not necessarily equivalent; the latter is not specified by this volume of IEEE Std 1003.1-200x  
 26433 and may result in an error. The construct `'\0'` is used to mean “no separator” because  
 26434 historical versions of *paste* did not follow the syntax guidelines, and the command:

```
26435 paste -d " " ...
```

26436 could not be handled properly by *getopt()*.

26437 **EXAMPLES**

26438 1. Write out a directory in four columns:

```
26439 ls | paste - - - -
```

26440 2. Combine pairs of lines from a file into single lines:

```
26441 paste -s -d "\t\n" file
```

26442 **RATIONALE**

26443 None.

26444 **FUTURE DIRECTIONS**

26445 None.

26446 **SEE ALSO**

26447 *cut, grep, pr*

26448 **CHANGE HISTORY**

26449 First released in Issue 2.

26450 **Issue 6**

26451 The normative text is reworded to avoid use of the term “must” for application requirements.

## 26452 NAME

26453 patch — apply changes to files

## 26454 SYNOPSIS

```
26455 UP patch [-blNR][ -c | -e | -n][-d dir][-D define][-i patchfile]
26456 [-o outfile][-p num][-r rejectfile][file]
```

26457

## 26458 DESCRIPTION

26459 The *patch* utility shall read a source (patch) file containing any of the three forms of difference (diff) listings produced by the *diff* utility (normal, context or in the style of *ed*) and apply those differences to a file. By default, *patch* shall read from the standard input.

26462 The *patch* utility shall attempt to determine the type of the *diff* listing, unless overruled by a *-c*, *-e*, or *-n* option.

26464 If the patch file contains more than one patch, *patch* shall attempt to apply each of them as if they came from separate patch files. (In this case, the application shall ensure that the name of the patch file is determinable for each *diff* listing.)

## 26467 OPTIONS

26468 The *patch* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2, Utility Syntax Guidelines.

26470 The following options shall be supported:

26471 **-b** Save a copy of the original contents of each modified file, before the differences are applied, in a file of the same name with the suffix **.orig** appended to it. If the file already exists, it shall be overwritten; if multiple patches are applied to the same file, the **.orig** file shall be written only for the first patch. When the *-o outfile* option is also specified, *file.orig* shall not be created but, if *outfile* already exists, *outfile.orig* shall be created.

26477 **-c** Interpret the patch file as a context difference (the output of the utility *diff* when the *-c* or *-C* options are specified).

26479 **-d dir** Change the current directory to *dir* before processing as described in the EXTENDED DESCRIPTION section.

26481 **-D define** Mark changes with one of the following C preprocessor constructs:

26482 #ifdef define

26483 ...

26484 #endif

26485 #ifndef define

26486 ...

26487 #endif

26488 optionally combined with the C preprocessor construct **#else**.

26489 **-e** Interpret the patch file as an *ed* script, rather than a *diff* script.

26490 **-i patchfile** Read the patch information from the file named by the pathname *patchfile*, rather than the standard input.

26492 **-l** (The letter ell.) Cause any sequence of <blank>s in the difference script to match any sequence of <blank>s in the input file. Other characters shall be matched exactly.

- 26495        **-n**           Interpret the script as a normal difference.
- 26496        **-N**           Ignore patches where the differences have already been applied to the file; by  
26497            default, already-applied patches shall be rejected.
- 26498        **-o outfile**    Instead of modifying the files (specified by the *file* operand or the difference  
26499            listings) directly, write a copy of the file referenced by each patch, with the  
26500            appropriate differences applied, to *outfile*. Multiple patches for a single file shall  
26501            be applied to the intermediate versions of the file created by any previous patches,  
26502            and shall result in multiple, concatenated versions of the file being written to  
26503            *outfile*.
- 26504        **-p num**        For all pathnames in the patch file that indicate the names of files to be patched,  
26505            delete *num* pathname components from the beginning of each pathname. If the  
26506            pathname in the patch file is absolute, any leading slashes shall be considered the  
26507            first component (that is, **-p 1** shall remove the leading slashes). Specifying **-p 0**  
26508            shall cause the full pathname to be used. If **-p** is not specified, only the basename  
26509            (the final pathname component) shall be used.
- 26510        **-R**            Reverse the sense of the patch script; that is, assume that the difference script was  
26511            created from the new version to the old version. The **-R** option cannot be used  
26512            with *ed* scripts. The *patch* utility shall attempt to reverse each portion of the script  
26513            before applying it. Rejected differences shall be saved in swapped format. If this  
26514            option is not specified, and until a portion of the patch file is successfully applied,  
26515            *patch* attempts to apply each portion in its reversed sense as well as in its normal  
26516            sense. If the attempt is successful, the user shall be prompted to determine  
26517            whether the **-R** option should be set.
- 26518        **-r rejectfile**   Override the default reject filename. In the default case, the reject file shall have the  
26519            same name as the output file, with the suffix **.rej** appended to it; see **Patch**  
26520            **Application** (on page 2893).
- 26521   **OPERANDS**  
26522        The following operand shall be supported:
- 26523        *file*            A pathname of a file to patch.
- 26524   **STDIN**  
26525        See the INPUT FILES section.
- 26526   **INPUT FILES**  
26527        Input files shall be text files.
- 26528   **ENVIRONMENT VARIABLES**  
26529        The following environment variables shall affect the execution of *patch*:
- 26530        **LANG**            Provide a default value for the internationalization variables that are unset or null.  
26531            (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
26532            Internationalization Variables for the precedence of internationalization variables  
26533            used to determine the values of locale categories.)
- 26534        **LC\_ALL**         If set to a non-empty string value, override the values of all the other  
26535            internationalization variables.
- 26536        **LC\_CTYPE**        Determine the locale for the interpretation of sequences of bytes of text data as  
26537            characters (for example, single-byte as opposed to multi-byte characters in  
26538            arguments and input files).

- 26539            *LC\_MESSAGES*
- 26540                    Determine the locale that should be used to affect the format and contents of
- 26541                    diagnostic messages written to standard error and informative messages written to
- 26542                    standard output.
- 26543 XSI            *NLSPATH*    Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 26544            *LC\_TIME*    Determine the locale for recognizing the format of file timestamps written by the
- 26545                    *diff* utility in a context-difference input file.
- 26546 **ASYNCHRONOUS EVENTS**
- 26547            Default.
- 26548 **STDOUT**
- 26549            Not used.
- 26550 **STDERR**
- 26551            The standard error shall be used for diagnostic and informational messages.
- 26552 **OUTPUT FILES**
- 26553            The output of the *patch* utility, the save files (**.orig** suffixes) and the reject files (**.rej** suffixes) shall
- 26554            be text files.
- 26555 **EXTENDED DESCRIPTION**
- 26556            A patchfile may contain patching instructions for more than one file; filenames shall be
- 26557            determined as specified in **Filename Determination** (on page 2893). When the **-b** option is
- 26558            specified, for each patched file, the original shall be saved in a file of the same name with the
- 26559            suffix **.orig** appended to it.
- 26560            For each patched file, a reject file may also be created as noted in **Patch Application** (on page
- 26561            2893). In the absence of a **-r** option, the name of this file shall be formed by appending the suffix
- 26562            **.rej** to the original filename.
- 26563            **Patchfile Format**
- 26564            The patch file shall contain zero or more lines of header information followed by one or more
- 26565            patches. Each patch shall contain zero or more lines of filename identification in the format
- 26566            produced by *diff -c*, and one or more sets of *diff* output, which are customarily called *hunks*.
- 26567            The *patch* utility shall recognize the following expression in the header information:
- 26568            **Index:** *pathname*
- 26569                    The file to be patched is named *pathname*.
- 26570            If all lines (including headers) within a patch begin with the same leading sequence of <blank>s,
- 26571            the *patch* utility shall remove this sequence before proceeding. Within each patch, if the type of
- 26572            difference is context, the *patch* utility shall recognize the following expressions:
- 26573            **\*\*\*** *filename timestamp*
- 26574                    The patches arose from *filename*.
- 26575            **---** *filename timestamp*
- 26576                    The patches should be applied to *filename*.
- 26577            Each hunk within a patch shall be the *diff* output to change a line range within the original file.
- 26578            The line numbers for successive hunks within a patch shall occur in ascending order.

26579 **Filename Determination**

26580 If no *file* operand is specified, *patch* shall perform the following steps to determine the filename  
26581 to use:

- 26582 1. If the type of *diff* is context, the *patch* utility shall delete pathname components (as  
26583 specified by the **-p** option) from the filename on the line beginning with "\*\*\*\*", then test  
26584 for the existence of this file relative to the current directory (or the directory specified with  
26585 the **-d** option). If the file exists, the *patch* utility shall use this filename.
- 26586 2. If the type of *diff* is context, the *patch* utility shall delete the pathname components (as  
26587 specified by the **-p** option) from the filename on the line beginning with "---", then test  
26588 for the existence of this file relative to the current directory (or the directory specified with  
26589 the **-d** option). If the file exists, the *patch* utility shall use this filename.
- 26590 3. If the header information contains a line beginning with the string **Index:**, the *patch* utility  
26591 shall delete pathname components (as specified by the **-p** option) from this line, then test  
26592 for the existence of this file relative to the current directory (or the directory specified with  
26593 the **-d** option). If the file exists, the *patch* utility shall use this filename.
- 26594 XSI 4. If an **SCCS** directory exists in the current directory, *patch* shall attempt to perform a *get -e*  
26595 **SCCS/s.filename** command to retrieve an editable version of the file. If the file exists, the  
26596 *patch* utility shall use this filename.
- 26597 5. The *patch* utility shall write a prompt to standard output and request a filename  
26598 interactively from the controlling terminal (for example, **/dev/tty**).

26599 **Patch Application**

26600 If the **-c**, **-e**, or **-n** option is present, the *patch* utility shall interpret information within each hunk  
26601 as a context difference, an *ed* difference or a normal difference, respectively. In the absence of  
26602 any of these options, the *patch* utility shall determine the type of difference based on the format  
26603 of information within the hunk.

26604 For each hunk, the *patch* utility shall begin to search for the place to apply the patch at the line  
26605 number at the beginning of the hunk, plus or minus any offset used in applying the previous  
26606 hunk. If lines matching the hunk context are not found, *patch* shall scan both forwards and  
26607 backwards at least 1 000 bytes for a set of lines that match the hunk context.

26608 If no such place is found and it is a context difference, then another scan shall take place,  
26609 ignoring the first and last line of context. If that fails, the first two and last two lines of context  
26610 shall be ignored and another scan shall be made. Implementations may search more extensively  
26611 for installation locations.

26612 If no location can be found, the *patch* utility shall append the hunk to the reject file. The rejected  
26613 hunk shall be written in context-difference format regardless of the format of the patch file. If the  
26614 input was a normal or *ed-style* difference, the reject file may contain differences with zero lines  
26615 of context. The line numbers on the hunks in the reject file may be different from the line  
26616 numbers in the patch file since they shall reflect the approximate locations for the failed hunks in  
26617 the new file rather than the old one.

26618 If the type of patch is an *ed* diff, the implementation may accomplish the patching by invoking  
26619 the *ed* utility.

26620 **EXIT STATUS**

26621 The following exit values shall be returned:

- 26622 0 Successful completion.

26623           1   One or more lines were written to a reject file.

26624           >1   An error occurred.

#### 26625 CONSEQUENCES OF ERRORS

26626           Patches that cannot be correctly placed in the file shall be written to a reject file.

#### 26627 APPLICATION USAGE

26628           The **-R** option does not work with *ed* scripts because there is too little information to reconstruct the reverse operation.

26630           The **-p** option makes it possible to customize a patchfile to local user directory structures without manually editing the patchfile. For example, if the filename in the patch file was:

26632           /*curds/whey/src/blurfl/blurfl.c*

26633           Setting **-p 0** gives the entire pathname unmodified; **-p 1** gives:

26634           *curds/whey/src/blurfl/blurfl.c*

26635           without the leading slash, **-p 4** gives:

26636           *blurfl/blurfl.c*

26637           and not specifying **-p** at all gives:

26638           *blurfl.c* .

#### 26639 EXAMPLES

26640           None.

#### 26641 RATIONALE

26642           Some of the functionality in historical *patch* implementations was not specified. The following documents those features present in historical implementations that have not been specified.

26644           A deleted piece of functionality was the '+' pseudo-option allowing an additional set of options and a patch file operand to be given. This was seen as being insufficiently useful to standardize.

26646           In historical implementations, if the string "Prereq:" appeared in the header, the *patch* utility would search for the corresponding version information (the string specified in the header, delimited by <blank>s or the beginning or end of a line or the file) anywhere in the original file. This was deleted as too simplistic and insufficiently trustworthy a mechanism to standardize. For example, if:

26651           Prereq: 1.2

26652           were in the header, the presence of a delimited 1.2 anywhere in the file would satisfy the prerequisite.

26654           The following options were dropped from historical implementations of *patch* as insufficiently useful to standardize:

26656           **-b**           The **-b** option historically provided a method for changing the name extension of the backup file from the default **.orig**. This option has been modified and retained in this volume of IEEE Std 1003.1-200x.

26659           **-F**           The **-F** option specified the number of lines of a context diff to ignore when searching for a place to install a patch.

26661           **-f**           The **-f** option historically caused *patch* not to request additional information from the user.

- 26663        **-r**            The **-r** option historically provided a method of overriding the extension of the  
26664                    reject file from the default **.rej**.
- 26665        **-s**            The **-s** option historically caused *patch* to work silently unless an error occurred.
- 26666        **-x**            The **-x** option historically set internal debugging flags.
- 26667        In some file system implementations, the saving of a **.orig** file may produce unwanted results. In  
26668        the case of 12, 13, or 14-character filenames (on file systems supporting 14-character maximum  
26669        filenames), the **.orig** file overwrites the new file. The reject file may also exceed this filename  
26670        limit. It was suggested, due to some historical practice, that a tilde ('~') suffix be used instead  
26671        of **.orig** and some other character instead of the **.rej** suffix. This was rejected because it is not  
26672        obvious to the user which file is which. The suffixes **.orig** and **.rej** are clearer and more  
26673        understandable.
- 26674        The **-b** option has the opposite sense in some historical implementations—do not save the **.orig**  
26675        file. The default case here is not to save the files, making *patch* behave more consistently with the  
26676        other standard utilities.
- 26677        The **-w** option in early proposals was changed to **-I** to match historical practice.
- 26678        The **-N** option was included because without it, a non-interactive application cannot reject  
26679        previously applied patches. For example, if a user is piping the output of *diff* into the *patch*  
26680        utility, and the user only wants to patch a file to a newer version non-interactively, the **-N**  
26681        option is required.
- 26682        Changes to the **-I** option description were proposed to allow matching across <newline>s in  
26683        addition to just <blank>s. Since this is not historical practice, and since some ambiguities could  
26684        result, it is suggested that future developments in this area utilize another option letter, such as  
26685        **-L**.
- 26686        **FUTURE DIRECTIONS**
- 26687                    None.
- 26688        **SEE ALSO**
- 26689                    *ed*, *diff*
- 26690        **CHANGE HISTORY**
- 26691                    First released in Issue 4.
- 26692        **Issue 5**
- 26693                    **FUTURE DIRECTIONS** section added.
- 26694        **Issue 6**
- 26695                    This utility is now marked as part of the User Portability Utilities option.
- 26696                    The description of the **-D** option and the steps in **Filename Determination** (on page 2893) are  
26697                    changed to match historical practice as defined in the IEEE P1003.2b draft standard.
- 26698                    The normative text is reworded to avoid use of the term “must” for application requirements.

## 26699 NAME

26700 pathchk — check pathnames

## 26701 SYNOPSIS

26702 pathchk [-p] *pathname*...

## 26703 DESCRIPTION

26704 The *pathchk* utility shall check that one or more pathnames are valid (that is, they could be used  
 26705 to access or create a file without causing syntax errors) and portable (that is, no filename  
 26706 truncation results). More extensive portability checks are provided by the **-p** option.

26707 By default, the *pathchk* utility shall check each component of each *pathname* operand based on the  
 26708 underlying file system. A diagnostic shall be written for each *pathname* operand that:

- 26709 • Is longer than {PATH\_MAX} bytes (see **Pathname Variable Values** in the Base Definitions  
 26710 volume of IEEE Std 1003.1-200x, Chapter 13, Headers, <limits.h>)
- 26711 • Contains any component longer than {NAME\_MAX} bytes in its containing directory
- 26712 • Contains any component in a directory that is not searchable
- 26713 • Contains any character in any component that is not valid in its containing directory

26714 The format of the diagnostic message is not specified, but shall indicate the error detected and  
 26715 the corresponding *pathname* operand.

26716 It shall not be considered an error if one or more components of a *pathname* operand do not exist  
 26717 as long as a file matching the pathname specified by the missing components could be created  
 26718 that does not violate any of the checks specified above.

## 26719 OPTIONS

26720 The *pathchk* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section  
 26721 12.2, Utility Syntax Guidelines.

26722 The following option shall be supported:

- 26723 **-p** Instead of performing checks based on the underlying file system, write a  
 26724 diagnostic for each *pathname* operand that:
  - 26725 • Is longer than {\_POSIX\_PATH\_MAX} bytes (see **Minimum Values** in the Base  
 26726 Definitions volume of IEEE Std 1003.1-200x, Chapter 13, Headers, <limits.h>)
  - 26727 • Contains any component longer than {\_POSIX\_NAME\_MAX} bytes
  - 26728 • Contains any character in any component that is not in the portable filename  
 26729 character set

## 26730 OPERANDS

26731 The following operand shall be supported:

26732 *pathname* A pathname to be checked.

## 26733 STDIN

26734 Not used.

## 26735 INPUT FILES

26736 None.

## 26737 ENVIRONMENT VARIABLES

26738 The following environment variables shall affect the execution of *pathchk*:

26739 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 26740 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,

26741 Internationalization Variables for the precedence of internationalization variables  
 26742 used to determine the values of locale categories.)

26743 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 26744 internationalization variables.

26745 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 26746 characters (for example, single-byte as opposed to multi-byte characters in  
 26747 arguments).

26748 **LC\_MESSAGES**  
 26749 Determine the locale that should be used to affect the format and contents of  
 26750 diagnostic messages written to standard error.

26751 XSI **NLSPATH** Determine the location of message catalogs for the processing of **LC\_MESSAGES**.

26752 **ASYNCHRONOUS EVENTS**  
 26753 Default.

26754 **STDOUT**  
 26755 Not used.

26756 **STDERR**  
 26757 The standard error shall be used only for diagnostic messages.

26758 **OUTPUT FILES**  
 26759 None.

26760 **EXTENDED DESCRIPTION**  
 26761 None.

26762 **EXIT STATUS**  
 26763 The following exit values shall be returned:

26764 0 All *pathname* operands passed all of the checks.

26765 >0 An error occurred.

26766 **CONSEQUENCES OF ERRORS**  
 26767 Default.

26768 **APPLICATION USAGE**  
 26769 The *test* utility can be used to determine whether a given pathname names an existing file; it  
 26770 does not, however, give any indication of whether or not any component of the pathname was  
 26771 truncated in a directory where the **\_POSIX\_NO\_TRUNC** feature is not in effect. The *pathchk*  
 26772 utility does not check for file existence; it performs checks to determine whether a pathname  
 26773 does exist or could be created with no pathname component truncation.

26774 The *noclobber* option in the shell (see the *set* (on page 2287) special built-in) can be used to  
 26775 atomically create a file. As with all file creation semantics in the System Interfaces volume of  
 26776 IEEE Std 1003.1-200x, it guarantees atomic creation, but still depends on applications to agree on  
 26777 conventions and cooperate on the use of files after they have been created.

26778 **EXAMPLES**  
 26779 To verify that all pathnames in an imported data interchange archive are legitimate and  
 26780 unambiguous on the current system:

```
26781 pax -f archive | sed -e '/ == ./s///' | xargs pathchk
26782 if [ $? -eq 0 ]
26783 then
26784     pax -r -f archive
```

```

26785     else
26786         echo Investigate problems before importing files.
26787         exit 1
26788     fi

26789     To verify that all files in the current directory hierarchy could be moved to any system
26790     conforming to the System Interfaces volume of IEEE Std 1003.1-200x that also supports the pax
26791     utility:

26792     find . -print | xargs pathchk -p
26793     if [ $? -eq 0 ]
26794     then
26795         pax -w -f archive .
26796     else
26797         echo Portable archive cannot be created.
26798         exit 1
26799     fi

26800     To verify that a user-supplied pathname names a readable file and that the application can create
26801     a file extending the given path without truncation and without overwriting any existing file:

26802     case $- in
26803         *C*)     reset="";;
26804         *)     reset="set +C"
26805                set -C;;
26806     esac
26807     test -r "$path" && pathchk "$path.out" &&
26808         rm "$path.out" > "$path.out"
26809     if [ $? -ne 0 ]; then
26810         printf "%s: %s not found or %s.out fails \
26811     creation checks.\n" $0 "$path" "$path"
26812         $reset      # Reset the noclobber option in case a trap
26813                    # on EXIT depends on it.
26814         exit 1
26815     fi
26816     $reset
26817     PROCESSING < "$path" > "$path.out"

26818     The following assumptions are made in this example:

26819     1.  PROCESSING represents the code that is used by the application to use $path once it is
26820     verified that $path.out works as intended.

26821     2.  The state of the noclobber option is unknown when this code is invoked and should be set
26822     on exit to the state it was in when this code was invoked. (The reset variable is used in this
26823     example to restore the initial state.)

26824     3.  Note the usage of:

26825     rm "$path.out" > "$path.out"

26826     a.  The pathchk command has already verified, at this point, that $path.out is not
26827     truncated.

26828     b.  With the noclobber option set, the shell verifies that $path.out does not already exist
26829     before invoking rm.

```

- 26830 c. If the shell succeeded in creating **\$path.out**, *rm* removes it so that the application can  
 26831 create the file again in the **PROCESSING** step.
- 26832 d. If the **PROCESSING** step wants the file to exist already when it is invoked, the:
- 26833 `rm "$path.out" > "$path.out"`
- 26834 should be replaced with:
- 26835 `> "$path.out"`
- 26836 which verifies that the file did not already exist, but leaves **\$path.out** in place for use  
 26837 by **PROCESSING**.

#### 26838 RATIONALE

26839 The *pathchk* utility is new, commissioned for this volume of IEEE Std 1003.1-200x. It, along with  
 26840 the *set -C(noclobber)* option added to the shell, replaces the *mktemp*, *validnam*, and *create* utilities  
 26841 that appeared in early proposals. All of these utilities were attempts to solve several common  
 26842 problems:

- 26843 • Verify the validity (for several different definitions of “valid”) of a pathname supplied by a  
 26844 user, generated by an application, or imported from an external source.
- 26845 • Atomically create a file.
- 26846 • Perform various string handling functions to generate a temporary filename.

26847 The *create* utility, included in an early proposal, provided checking and atomic creation in a  
 26848 single invocation of the utility; these are orthogonal issues and need not be grouped into a single  
 26849 utility. Note that the *noclobber* option also provides a way of creating a lock for process  
 26850 synchronization; since it provides an atomic *create*, there is no race between a test for existence  
 26851 and the following creation if it did not exist.

26852 Having a function like *tmpnam()* in the ISO C standard is important in many high-level  
 26853 languages. The shell programming language, however, has built-in string manipulation  
 26854 facilities, making it very easy to construct temporary filenames. The names needed obviously  
 26855 depend on the application, but are frequently of a form similar to:

26856 `$TMPDIR/application_abbreviation$$ .suffix`

26857 In cases where there is likely to be contention for a given suffix, a simple shell *for* or *while* loop  
 26858 can be used with the shell *noclobber* option to create a file without risk of collisions, as long as  
 26859 applications trying to use the same filename name space are cooperating on the use of files after  
 26860 they have been created.

#### 26861 FUTURE DIRECTIONS

26862 None.

#### 26863 SEE ALSO

26864 *test*, Section 2.7 (on page 2244)

#### 26865 CHANGE HISTORY

26866 First released in Issue 4.

## 26867 NAME

26868 pax — portable archive interchange

## 26869 SYNOPSIS

26870 pax [-cdnv][[-H|-L][[-f *archive*][[-s *replstr*]]...[*pattern*...]26871 pax -r[-cdiknuv][[-H|-L][[-f *archive*][[-o *options*]]...[-p *string*]]...  
26872 [-s *replstr*]]...[*pattern*...]26873 pax -w[-dituvX][[-H|-L][[-b *blocksize*][[-a][[-f *archive*][[-o *options*]]...]  
26874 [-s *replstr*]]...[-x *format*][*file*...]26875 pax -r -w[-diklntuvX][[-H|-L][[-p *string*]]...[-s *replstr*]]...  
26876 [*file*...] *directory*

## 26877 DESCRIPTION

26878 The *pax* utility shall read, write, and write lists of the members of archive files and copy  
26879 directory hierarchies. A variety of archive formats shall be supported; see the *-x format* option.26880 The action to be taken depends on the presence of the *-r* and *-w* options. The four combinations  
26881 of *-r* and *-w* are referred to as the four modes of operation: **list**, **read**, **write**, and **copy** modes,  
26882 corresponding respectively to the four forms shown in the SYNOPSIS section.26883 **list** In **list** mode (when neither *-r* nor *-w* are specified), *pax* shall write the names of  
26884 the members of the archive file read from the standard input, with pathnames  
26885 matching the specified patterns, to standard output. If a named file is of type  
26886 directory, the file hierarchy rooted at that file shall be listed as well.26887 **read** In **read** mode (when *-r* is specified, but *-w* is not), *pax* shall extract the members of  
26888 the archive file read from the standard input, with pathnames matching the  
26889 specified patterns. If an extracted file is of type directory, the file hierarchy rooted  
26890 at that file shall be extracted as well. The extracted files shall be created performing  
26891 pathname resolution with the directory in which *pax* was invoked as the current  
26892 working directory.26893 If an attempt is made to extract a directory when the directory already exists, this  
26894 shall not be considered to be an error. If an attempt is made to extract a FIFO when  
26895 the FIFO already exists, this shall not be considered to be an error.26896 The ownership, access, and modification times, and file mode of the restored files  
26897 are discussed under the *-p* option.26898 **write** In **write** mode (when *-w* is specified, but *-r* is not), *pax* shall write the contents of  
26899 the *file* operands to the standard output in an archive format. If no *file* operands are  
26900 specified, a list of files to copy, one per line, shall be read from the standard input.  
26901 A file of type directory shall include all of the files in the file hierarchy rooted at the  
26902 file.26903 **copy** In **copy** mode (when both *-r* and *-w* are specified), *pax* shall copy the *file* operands  
26904 to the destination directory.26905 If no *file* operands are specified, a list of files to copy, one per line, shall be read  
26906 from the standard input. A file of type directory shall include all of the files in the  
26907 file hierarchy rooted at the file.26908 The effect of the **copy** shall be as if the copied files were written to an archive file  
26909 and then subsequently extracted, except that there may be hard links between the  
26910 original and the copied files. If the destination directory is a subdirectory of one of  
26911 the files to be copied, the results are unspecified. If the destination directory is a

26912 file of a type not defined by the System Interfaces volume of IEEE Std 1003.1-200x,  
 26913 the results are implementation-defined; otherwise, it shall be an error for the file  
 26914 named by the *directory* operand not to exist, not be writable by the user, or not be a  
 26915 file of type *directory*.

26916 In **read** or **copy** modes, if intermediate directories are necessary to extract an archive member,  
 26917 *pax* shall perform actions equivalent to the *mkdir()* function defined in the System Interfaces  
 26918 volume of IEEE Std 1003.1-200x, called with the following arguments:

- 26919 • The intermediate directory used as the *path* argument
- 26920 • The value of the bitwise-inclusive OR of S\_IRWXU, S\_IRWXG, and S\_IRWXO as the *mode*  
 26921 argument

26922 If any specified *pattern* or *file* operands are not matched by at least one file or archive member,  
 26923 *pax* shall write a diagnostic message to standard error for each one that did not match and exit  
 26924 with a non-zero exit status.

26925 The archive formats described in the EXTENDED DESCRIPTION section shall be automatically  
 26926 detected on input. The default output archive format shall be implementation-defined.

26927 A single archive can span multiple files. The *pax* utility shall determine, in an implementation-  
 26928 defined manner, what file to read or write as the next file.

26929 If the selected archive format supports the specification of linked files, it shall be an error if these  
 26930 files cannot be linked when the archive is extracted. For archive formats that do not store file  
 26931 contents with each name that causes a hard link, if the file that contains the data is not extracted  
 26932 during this *pax* session, either the data shall be restored from the original file, or a diagnostic  
 26933 message shall be displayed with the name of a file that can be used to extract the data. In  
 26934 traversing directories, *pax* shall detect infinite loops; that is, entering a previously visited  
 26935 directory that is an ancestor of the last file visited. When it detects an infinite loop, *pax* shall  
 26936 write a diagnostic message to standard error and shall terminate.

#### 26937 OPTIONS

26938 The *pax* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section  
 26939 12.2, Utility Syntax Guidelines, except that the order of presentation of the **-o**, **-p**, and **-s** options  
 26940 is significant.

26941 The following options shall be supported:

- 26942 **-r** Read an archive file from standard input.
- 26943 **-w** Write files to the standard output in the specified archive format.
- 26944 **-a** Append files to the end of the archive. It is implementation-defined which devices  
 26945 on the system support appending. Additional file formats unspecified by this  
 26946 volume of IEEE Std 1003.1-200x may impose restrictions on appending.
- 26947 **-b *blocksize*** Block the output at a positive decimal integer number of bytes per write to the  
 26948 archive file. Devices and archive formats may impose restrictions on blocking.  
 26949 Blocking shall be automatically determined on input. Conforming applications |  
 26950 shall not specify a *blocksize* value larger than 32 256. Default blocking when |  
 26951 creating archives depends on the archive format. (See the **-x** option below.)
- 26952 **-c** Match all file or archive members except those specified by the *pattern* or *file*  
 26953 operands.
- 26954 **-d** Cause files of type *directory* being copied or archived or archive members of type  
 26955 *directory* being extracted or listed to match only the file or archive member itself  
 26956 and not the file hierarchy rooted at the file.

|       |                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|-------|--------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 26957 | <b>-f</b> <i>archive</i> | Specify the pathname of the input or output archive, overriding the default standard input (in <b>list</b> or <b>read</b> modes) or standard output ( <b>write</b> mode).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 26958 |                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 26959 | <b>-H</b>                | If a symbolic link referencing a file of type directory is specified on the command line, <i>pax</i> shall archive the file hierarchy rooted in the file referenced by the link, using the name of the link as the root of the file hierarchy. Otherwise, if a symbolic link referencing a file of any other file type which <i>pax</i> can normally archive is specified on the command line, then <i>pax</i> shall archive the file referenced by the link, using the name of the link. The default behavior shall be to archive the symbolic link itself.                                                                                                                                                                                                                                                                           |
| 26960 |                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 26961 |                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 26962 |                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 26963 |                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 26964 |                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 26965 |                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 26966 | <b>-i</b>                | Interactively rename files or archive members. For each archive member matching a <i>pattern</i> operand or file matching a <i>file</i> operand, a prompt shall be written to the file <b>/dev/tty</b> . The prompt shall contain the name of the file or archive member, but the format is otherwise unspecified. A line shall then be read from <b>/dev/tty</b> . If this line is blank, the file or archive member shall be skipped. If this line consists of a single period, the file or archive member shall be processed with no modification to its name. Otherwise, its name shall be replaced with the contents of the line. The <i>pax</i> utility shall immediately exit with a non-zero exit status if end-of-file is encountered when reading a response or if <b>/dev/tty</b> cannot be opened for reading and writing. |
| 26967 |                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 26968 |                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 26969 |                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 26970 |                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 26971 |                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 26972 |                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 26973 |                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 26974 |                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 26975 |                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 26976 |                          | The results of extracting a hard link to a file that has been renamed during extraction are unspecified.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 26977 |                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 26978 | <b>-k</b>                | Prevent the overwriting of existing files.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 26979 | <b>-l</b>                | (The letter ell.) In <b>copy</b> mode, hard links shall be made between the source and destination file hierarchies whenever possible. If specified in conjunction with <b>-H</b> or <b>-L</b> , when a symbolic link is encountered, the hard link created in the destination file hierarchy shall be to the file referenced by the symbolic link. If specified when neither <b>-H</b> nor <b>-L</b> is specified, when a symbolic link is encountered, the implementation shall create a hard link to the symbolic link in the source file hierarchy or copy the symbolic link to the destination.                                                                                                                                                                                                                                   |
| 26980 |                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 26981 |                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 26982 |                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 26983 |                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 26984 |                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 26985 |                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 26986 | <b>-L</b>                | If a symbolic link referencing a file of type directory is specified on the command line or encountered during the traversal of a file hierarchy, <i>pax</i> shall archive the file hierarchy rooted in the file referenced by the link, using the name of the link as the root of the file hierarchy. Otherwise, if a symbolic link referencing a file of any other file type which <i>pax</i> can normally archive is specified on the command line or encountered during the traversal of a file hierarchy, <i>pax</i> shall archive the file referenced by the link, using the name of the link. The default behavior shall be to archive the symbolic link itself.                                                                                                                                                                |
| 26987 |                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 26988 |                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 26989 |                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 26990 |                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 26991 |                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 26992 |                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 26993 |                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 26994 | <b>-n</b>                | Select the first archive member that matches each <i>pattern</i> operand. No more than one archive member shall be matched for each pattern (although members of type directory shall still match the file hierarchy rooted at that file).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 26995 |                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 26996 |                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 26997 | <b>-o</b> <i>options</i> | Provide information to the implementation to modify the algorithm for extracting or writing files. The value of <i>options</i> shall consist of one or more comma-separated keywords of the form:                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 26998 |                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 26999 |                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 27000 |                          | <i>keyword</i> [[:]= <i>value</i> ][, <i>keyword</i> [[:]= <i>value</i> ], ...]                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 27001 |                          | Some keywords apply only to certain file formats, as indicated with each description. Use of keywords that are inapplicable to the file format being processed produces undefined results.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 27002 |                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 27003 |                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |

27004 Keywords in the *options* argument shall be a string that would be a valid portable  
 27005 filename as described in the Base Definitions volume of IEEE Std 1003.1-200x,  
 27006 Section 3.276, Portable Filename Character Set.

27007 **Note:** Keywords are not expected to be filenames, merely to follow the same character  
 27008 composition rules as portable filenames.

27009 Keywords can be preceded with white space. The *value* field shall consist of zero or  
 27010 more characters; within *value*, the application shall precede any literal comma with  
 27011 a backslash, which shall be ignored, but preserves the comma as part of *value*. A  
 27012 comma as the final character, or a comma followed solely by white space as the  
 27013 final characters, in *options* shall be ignored. Multiple **-o** options can be specified; if  
 27014 keywords given to these multiple **-o** options conflict, the keywords and values  
 27015 appearing later in command line sequence shall take precedence and the earlier  
 27016 shall be silently ignored. The following keyword values of *options* shall be  
 27017 supported for the file formats as indicated:

27018 **delete=pattern**

27019 (Applicable only to the **-x pax** format.) When used in **write** or **copy** mode, *pax*  
 27020 shall omit from extended header records that it produces any keywords  
 27021 matching the string pattern. When used in **read** or **list** mode, *pax* shall ignore  
 27022 any keywords matching the string pattern in the extended header records. In  
 27023 both cases, matching shall be performed using the pattern matching notation  
 27024 described in Section 2.13.1 (on page 2264) and Section 2.13.2 (on page 2264).  
 27025 For example:

27026 **-o delete=security.\***

27027 would suppress security-related information. See **pax Extended Header** (on  
 27028 page 2913) for extended header record keyword usage.

27029 **exthdr.name=string**

27030 (Applicable only to the **-x pax** format.) This keyword allows user control over  
 27031 the name that is written into the **ustar** header blocks for the extended header  
 27032 produced under the circumstances described in **pax Header Block** (on page  
 27033 2912). The name shall be the contents of *string*, after the following character  
 27034 substitutions have been made:

| <i>string</i><br>Includes: | Replaced By:                                                                                                       |
|----------------------------|--------------------------------------------------------------------------------------------------------------------|
| %d                         | The directory name of the file, equivalent to the result of the <i>dirname</i> utility on the translated pathname. |
| %f                         | The filename of the file, equivalent to the result of the <i>basename</i> utility on the translated pathname.      |
| %%                         | A '%' character.                                                                                                   |

27042 Any other '%' characters in *string* produce undefined results.

27043 If no **-o exthdr.name=string** is specified, *pax* shall use the following default  
 27044 value:

27045 %d/PaxHeaders/%f

27046 **globexthdr.name=string**

27047 (Applicable only to the **-x pax** format.) When used in **write** or **copy** mode with  
 27048 the appropriate options, *pax* shall create global extended header records with  
 27049 **ustar** header blocks that will be treated as regular files by previous versions of

27050  
27051  
27052

*pax*. This keyword allows user control over the name that is written into the **ustar** header blocks for global extended header records. The name shall be the contents of string, after the following character substitutions have been made:

27053  
27054  
27055  
27056  
27057

| <i>string</i><br><b>Includes:</b> | <b>Replaced By:</b>                                                                                                |
|-----------------------------------|--------------------------------------------------------------------------------------------------------------------|
| %n                                | An integer that represents the sequence number of the global extended header record in the archive, starting at 1. |
| %%                                | A '%' character.                                                                                                   |

27058

Any other '%' characters in *string* produce undefined results.

27059  
27060

If no **-o globexthdr.name=string** is specified, *pax* shall use the following default value:

27061

\$TMPDIR/GlobalHead.%n

27062  
27063

where \$TMPDIR represents the value of the *TMPDIR* environment variable. If *TMPDIR* is not set, *pax* shall use **/tmp**.

27064

**invalid=action**

27065  
27066  
27067  
27068  
27069  
27070

(Applicable only to the **-x pax** format.) This keyword allows user control over the action *pax* takes upon encountering values in an extended header record that, in **read** or **copy** mode, are invalid in the destination hierarchy or, in **list** mode, cannot be written in the codeset and current locale of the implementation. The following are invalid values that shall be recognized by *pax*:

27071  
27072  
27073

— In **read** or **copy** mode, a filename or link name that contains character encodings invalid in the destination hierarchy. (For example, the name may contain embedded NULs.)

27074  
27075  
27076

— In **read** or **copy** mode, a filename or link name that is longer than the maximum allowed in the destination hierarchy (for either a pathname component or the entire pathname).

27077  
27078  
27079

— In **list** mode, any character string value (filename, link name, user name, and so on) that cannot be written in the codeset and current locale of the implementation.

27080  
27081

The following mutually-exclusive values of the *action* argument are supported:

27082  
27083  
27084  
27085

**bypass** In **read** or **copy** mode, *pax* shall bypass the file, causing no change to the destination hierarchy. In **list** mode, *pax* shall write all requested valid values for the file, but its method for writing invalid values is unspecified.

27086  
27087  
27088  
27089

**rename** In **read** or **copy** mode, *pax* shall act as if the **-i** option were in effect for each file with invalid filename or link name values, allowing the user to provide a replacement name interactively. In **list** mode, *pax* shall behave identically to the **bypass** action.

27090  
27091  
27092  
27093  
27094

**UTF-8** When used in **read**, **copy**, or **list** mode and a filename, link name, owner name, or any other field in an extended header record cannot be translated from the *pax* UTF-8 codeset format to the codeset and current locale of the implementation, *pax* shall use the actual UTF-8 encoding for the name.



27142 the group name will be forced to a new value for all files read from the  
27143 archive.

27144 The precedences of **-o** keywords over various fields in the archive are described in  
27145 **pax Extended Header Keyword Precedence** (on page 2916).

27146 **-p string** Specify one or more file characteristic options (privileges). The *string* option-  
27147 argument shall be a string specifying file characteristics to be retained or discarded  
27148 on extraction. The string shall consist of the specification characters *a*, *e*, *m*, *o*, and  
27149 *p*. Other implementation-defined characters can be included. Multiple  
27150 characteristics can be concatenated within the same string and multiple **-p** options  
27151 can be specified. The meaning of the specification characters are as follows:

27152 *a* Do not preserve file access times.

27153 *e* Preserve the user ID, group ID, file mode bits (see the Base Definitions volume  
27154 of IEEE Std 1003.1-200x, Section 3.168, File Mode Bits), access time,  
27155 modification time, and any other implementation-defined file characteristics.

27156 *m* Do not preserve file modification times.

27157 *o* Preserve the user ID and group ID.

27158 *p* Preserve the file mode bits. Other implementation-defined file mode attributes  
27159 may be preserved.

27160 In the preceding list, “preserve” indicates that an attribute stored in the archive  
27161 shall be given to the extracted file, subject to the permissions of the invoking  
27162 process. The access and modification times of the file shall be preserved unless  
27163 otherwise specified with the **-p** option or not stored in the archive. All attributes  
27164 that are not preserved shall be determined as part of the normal file creation action  
27165 (see Section 1.7.1.4 (on page 2204)).

27166 If neither the *e* nor the *o* specification character is specified, or the user ID and  
27167 group ID are not preserved for any reason, *pax* shall not set the *S\_ISUID* and  
27168 *S\_ISGID* bits of the file mode.

27169 If the preservation of any of these items fails for any reason, *pax* shall write a  
27170 diagnostic message to standard error. Failure to preserve these items shall affect  
27171 the final exit status, but shall not cause the extracted file to be deleted.

27172 If file characteristic letters in any of the *string* option-arguments are duplicated or  
27173 conflict with each other, the ones given last shall take precedence. For example, if  
27174 **-p eme** is specified, file modification times are preserved.

27175 **-s replstr** Modify file or archive member names named by *pattern* or *file* operands according  
27176 to the substitution expression *replstr*, using the syntax of the *ed* utility. The  
27177 concepts of “address” and “line” are meaningless in the context of the *pax* utility,  
27178 and shall not be supplied. The format shall be:

27179 `-s /old/new/[gp]`

27180 where as in *ed*, *old* is a basic regular expression and *new* can contain an ampersand,  
27181 ‘\n’ (where *n* is a digit) backreferences, or subexpression matching. The *old* string  
27182 also shall be permitted to contain <newline>s.

27183 Any non-null character can be used as a delimiter (‘/’ shown here). Multiple **-s**  
27184 expressions can be specified; the expressions shall be applied in the order  
27185 specified, terminating with the first successful substitution. The optional trailing  
27186 ‘g’ is as defined in the *ed* utility. The optional trailing ‘p’ shall cause successful

27187 substitutions to be written to standard error. File or archive member names that  
 27188 substitute to the empty string shall be ignored when reading and writing archives.

27189 **-t** When reading files from the file system, and if the user has the permissions  
 27190 required by *utime()* to do so, set the access time of each file read to the access time  
 27191 that it had before being read by *pax*.

27192 **-u** Ignore files that are older (having a less recent file modification time) than a pre-  
 27193 existing file or archive member with the same name. In **read** mode, an archive  
 27194 member with the same name as a file in the file system shall be extracted if the  
 27195 archive member is newer than the file. In **write** mode, an archive file member with  
 27196 the same name as a file in the file system shall be superseded if the file is newer  
 27197 than the archive member. If **-a** is also specified, this is accomplished by appending  
 27198 to the archive; otherwise, it is unspecified whether this is accomplished by actual  
 27199 replacement in the archive or by appending to the archive. In **copy** mode, the file in  
 27200 the destination hierarchy shall be replaced by the file in the source hierarchy or by  
 27201 a link to the file in the source hierarchy if the file in the source hierarchy is newer.

27202 **-v** In **list** mode, produce a verbose table of contents (see the STDOUT section).  
 27203 Otherwise, write archive member pathnames to standard error (see the STDERR  
 27204 section).

27205 **-x format** Specify the output archive format. The *pax* utility shall support the following  
 27206 formats:

27207 **cpio** The *cpio* interchange format; see the EXTENDED DESCRIPTION  
 27208 section. The default *blocksize* for this format for character special  
 27209 archive files shall be 5120. Implementations shall support all  
 27210 *blocksize* values less than or equal to 32 256 that are multiples of 512.

27211 **pax** The *pax* interchange format; see the EXTENDED DESCRIPTION  
 27212 section. The default *blocksize* for this format for character special  
 27213 archive files shall be 5120. Implementations shall support all  
 27214 *blocksize* values less than or equal to 32 256 that are multiples of 512.

27215 **ustar** The *tar* interchange format; see the EXTENDED DESCRIPTION  
 27216 section. The default *blocksize* for this format for character special  
 27217 archive files shall be 10 240. Implementations shall support all  
 27218 *blocksize* values less than or equal to 32 256 that are multiples of 512.

27219 Implementation-defined formats shall specify a default block size as well as any  
 27220 other block sizes supported for character special archive files.

27221 Any attempt to append to an archive file in a format different from the existing  
 27222 archive format shall cause *pax* to exit immediately with a non-zero exit status.

27223 In **copy** mode, if no **-x format** is specified, *pax* shall behave as if **-xpax** were  
 27224 specified.

27225 **-X** When traversing the file hierarchy specified by a pathname, *pax* shall not descend  
 27226 into directories that have a different device ID (*st\_dev*; see the System Interfaces  
 27227 volume of IEEE Std 1003.1-200x, *stat()*).

27228 The options that operate on the names of files or archive members (**-c**, **-i**, **-n**, **-s**, **-u**, and **-v**)  
 27229 shall interact as follows. In **read** mode, the archive members shall be selected based on the user-  
 27230 specified *pattern* operands as modified by the **-c**, **-n**, and **-u** options. Then, any **-s** and **-i** options  
 27231 shall modify, in that order, the names of the selected files. The **-v** option shall write names  
 27232 resulting from these modifications.

27233 In **write** mode, the files shall be selected based on the user-specified pathnames as modified by  
 27234 the **-n** and **-u** options. Then, any **-s** and **-i** options shall modify, in that order, the names of  
 27235 these selected files. The **-v** option shall write names resulting from these modifications.

27236 If both the **-u** and **-n** options are specified, *pax* shall not consider a file selected unless it is newer  
 27237 than the file to which it is compared.

### 27238 List Mode Format Specifications

27239 In **list** mode with the **-o listopt=format** option, the *format* argument shall be applied for each  
 27240 selected file. The *pax* utility shall append a <newline> to the **listopt** output for each selected file.  
 27241 The format argument shall be used as the *format* string described in the Base Definitions volume  
 27242 of IEEE Std 1003.1-200x, Chapter 5, File Format Notation, with the exceptions 1. through 5.  
 27243 defined in the EXTENDED DESCRIPTION section of *printf*, plus the following exceptions:

27244 6. The sequence (*keyword*) can occur before a format conversion specifier. The conversion  
 27245 argument is defined by the value of *keyword*. The implementation shall support the  
 27246 following keywords:

27247 — Any of the Field Name entries in Table 4-13 (on page 2917) and Table 4-15 (on page  
 27248 2920). The implementation may support the *cpio* keywords without the leading **c\_** in  
 27249 addition to the form required by Table 4-16 (on page 2921).

27250 — Any keyword defined for the extended header in **pax Extended Header** (on page 2913).

27251 — Any keyword provided as an implementation-defined extension within the extended  
 27252 header defined in **pax Extended Header** (on page 2913).

27253 For example, the sequence "%(charset)s" is the string value of the name of the character  
 27254 set in the extended header.

27255 The result of the keyword conversion argument shall be the value from the applicable  
 27256 header field or extended header, without any trailing NULs.

27257 All keyword values used as conversion arguments shall be translated from the UTF-8  
 27258 encoding to the character set appropriate for the local file system, user database, and so on,  
 27259 as applicable.

27260 7. An additional conversion specifier character, **T**, shall be used to specify time formats. The **T**  
 27261 conversion specifier character can be preceded by the sequence (*keyword=subformat*), where  
 27262 *subformat* is a date format as defined by *date* operands. The default *keyword* shall be **mtime**  
 27263 and the default subformat shall be:

27264 %b %e %H:%M %Y

27265 8. An additional conversion specifier character, **M**, shall be used to specify the file mode string  
 27266 as defined in *ls* Standard Output. If (*keyword*) is omitted, the **mode** keyword shall be used.  
 27267 For example, %.1M writes the single character corresponding to the <entry type> field of the  
 27268 *ls -l* command.

27269 9. An additional conversion specifier character, **D**, shall be used to specify the device for block  
 27270 or special files, if applicable, in an implementation-defined format. If not applicable, and  
 27271 (*keyword*) is specified, then this conversion shall be equivalent to %(*keyword*)u. If not  
 27272 applicable, and (*keyword*) is omitted, then this conversion shall be equivalent to <space>.

27273 10. An additional conversion specifier character, **F**, shall be used to specify a pathname. The **F**  
 27274 conversion character can be preceded by a sequence of comma-separated keywords:

27275 (*keyword*[,*keyword*] . . . )

27276 The values for all the keywords that are non-null shall be concatenated together, each  
 27277 separated by a '/'. The default shall be **(path)** if the keyword **path** is defined; otherwise,  
 27278 the default shall be **(prefix,name)**.

27279 11. An additional conversion specifier character, **L**, shall be used to specify a symbolic line  
 27280 expansion. If the current file is a symbolic link, then **%L** shall expand to:

27281 "**%s -> %s**", *<value of keyword>*, *<contents of link>*

27282 Otherwise, the **%L** conversion specification shall be the equivalent of **%F**.

### 27283 OPERANDS

27284 The following operands shall be supported:

27285 *directory* The destination directory pathname for **copy** mode.

27286 *file* A pathname of a file to be copied or archived.

27287 *pattern* A pattern matching one or more pathnames of archive members. A pattern must  
 27288 be given in the name-generating notation of the pattern matching notation in  
 27289 Section 2.13 (on page 2264), including the filename expansion rules in Section  
 27290 2.13.3 (on page 2265). The default, if no *pattern* is specified, is to select all members  
 27291 in the archive.

### 27292 STDIN

27293 In **write** mode, the standard input shall be used only if no *file* operands are specified. It shall be a  
 27294 text file containing a list of pathnames, one per line, without leading or trailing **<blank>s**.

27295 In **list** and **read** modes, if **-f** is not specified, the standard input shall be an archive file.

27296 Otherwise, the standard input shall not be used.

### 27297 INPUT FILES

27298 The input file named by the *archive* option-argument, or standard input when the archive is read  
 27299 from there, shall be a file formatted according to one of the specifications in the EXTENDED  
 27300 DESCRIPTION section or some other implementation-defined format.

27301 The file **/dev/tty** shall be used to write prompts and read responses.

### 27302 ENVIRONMENT VARIABLES

27303 The following environment variables shall affect the execution of *pax*:

27304 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 27305 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
 27306 Internationalization Variables for the precedence of internationalization variables  
 27307 used to determine the values of locale categories.)

27308 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 27309 internationalization variables.

#### 27310 *LC\_COLLATE*

27311 Determine the locale for the behavior of ranges, equivalence classes and multi-  
 27312 character collating elements used in the pattern matching expressions for the  
 27313 *pattern* operand, the basic regular expression for the **-s** option, and the extended  
 27314 regular expression defined for the **yesexpr** locale keyword in the *LC\_MESSAGES*  
 27315 category.

27316 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 27317 characters (for example, single-byte as opposed to multi-byte characters in  
 27318 arguments and input files), the behavior of character classes used in the extended  
 27319 regular expression defined for the **yesexpr** locale keyword in the *LC\_MESSAGES*

- 27320 category, and pattern matching.
- 27321 **LC\_MESSAGES**
- 27322 Determine the locale for the processing of affirmative responses that should be  
27323 used to affect the format and contents of diagnostic messages written to standard  
27324 error.
- 27325 **LC\_TIME** Determine the format and contents of date and time strings when the `-v` option is  
27326 specified.
- 27327 XSI **NLSPATH** Determine the location of message catalogs for the processing of **LC\_MESSAGES**.
- 27328 **TMPDIR** Determine the pathname that provides part of the default global extended header  
27329 record file, as described for the `-o globexhdr=` keyword as described in the  
27330 OPTIONS section.
- 27331 **TZ** Determine the timezone used to calculate date and time strings when the `-v` option  
27332 is specified. If **TZ** is unset or null, an unspecified default timezone shall be used.
- 27333 **ASYNCHRONOUS EVENTS**
- 27334 Default.
- 27335 **STDOUT**
- 27336 In **write** mode, if `-f` is not specified, the standard output shall be the archive formatted  
27337 according to one of the specifications in the EXTENDED DESCRIPTION section, or some other  
27338 implementation-defined format (see `-x format`).
- 27339 In **list** mode, when the `-olistopt=format` has been specified, the selected archive members shall  
27340 be written to standard output using the format described under **List Mode Format**  
27341 **Specifications** (on page 2908). In **list** mode without the `-olistopt=format` option, the table of  
27342 contents of the selected archive members shall be written to standard output using the following  
27343 format:
- 27344 `"%s\n", <path name>`
- 27345 If the `-v` option is specified in **list** mode, the table of contents of the selected archive members  
27346 shall be written to standard output using the following formats.
- 27347 For pathnames representing hard links to previous members of the archive:
- 27348 `"%sΔ=Δ%s\n", <ls -l listing>, <linkname>`
- 27349 For all other pathnames:
- 27350 `"%s\n", <ls -l listing>`
- 27351 where `<ls -l listing>` shall be the format specified by the `ls` utility with the `-l` option. When  
27352 writing pathnames in this format, it is unspecified what is written for fields for which the  
27353 underlying archive format does not have the correct information, although the correct number of  
27354 `<blank>`-separated fields shall be written.
- 27355 In **list** mode, standard output shall not be buffered more than a line at a time.
- 27356 **STDERR**
- 27357 If `-v` is specified in **read**, **write**, or **copy** modes, `pax` shall write the pathnames it processes to the  
27358 standard error output using the following format:
- 27359 `"%s\n", <pathname>`
- 27360 These pathnames shall be written as soon as processing is begun on the file or archive member,  
27361 and shall be flushed to standard error. The trailing `<newline>`, which shall not be buffered, is  
27362 written when the file has been read or written.

27363 If the **-s** option is specified, and the replacement string has a trailing 'p', substitutions shall be  
 27364 written to standard error in the following format:

27365 "%sΔ>>Δ%s\n", <original pathname>, <new pathname>

27366 In all operating modes of *pax*, optional messages of unspecified format concerning the input  
 27367 archive format and volume number, the number of files, blocks, volumes, and media parts as  
 27368 well as other diagnostic messages may be written to standard error.

27369 In all formats, for both standard output and standard error, it is unspecified how non-printable  
 27370 characters in pathnames or link names are written.

27371 When *pax* is in **read** mode or **list** mode, using the **-xpax** archive format, and a filename, link  
 27372 name, owner name, or any other field in an extended header record cannot be translated from  
 27373 the *pax* UTF-8 codeset format to the codeset and current locale of the implementation, *pax* shall  
 27374 write a diagnostic message to standard error, shall process the file as described for the **-o**  
 27375 **invalid=option**, and then shall process the next file in the archive.

#### 27376 OUTPUT FILES

27377 In **read** mode, the extracted output files shall be of the archived file type. In **copy** mode, the  
 27378 copied output files shall be the type of the file being copied. In either mode, existing files in the  
 27379 destination hierarchy shall be overwritten only when all permission (**-p**), modification time (**-u**),  
 27380 and invalid-value (**-oinvalid=**) tests allow it.

27381 In **write** mode, the output file named by the **-f** option-argument shall be a file formatted  
 27382 according to one of the specifications in the EXTENDED DESCRIPTION section, or some other  
 27383 implementation-defined format.

#### 27384 EXTENDED DESCRIPTION

##### 27385 **pax Interchange Format**

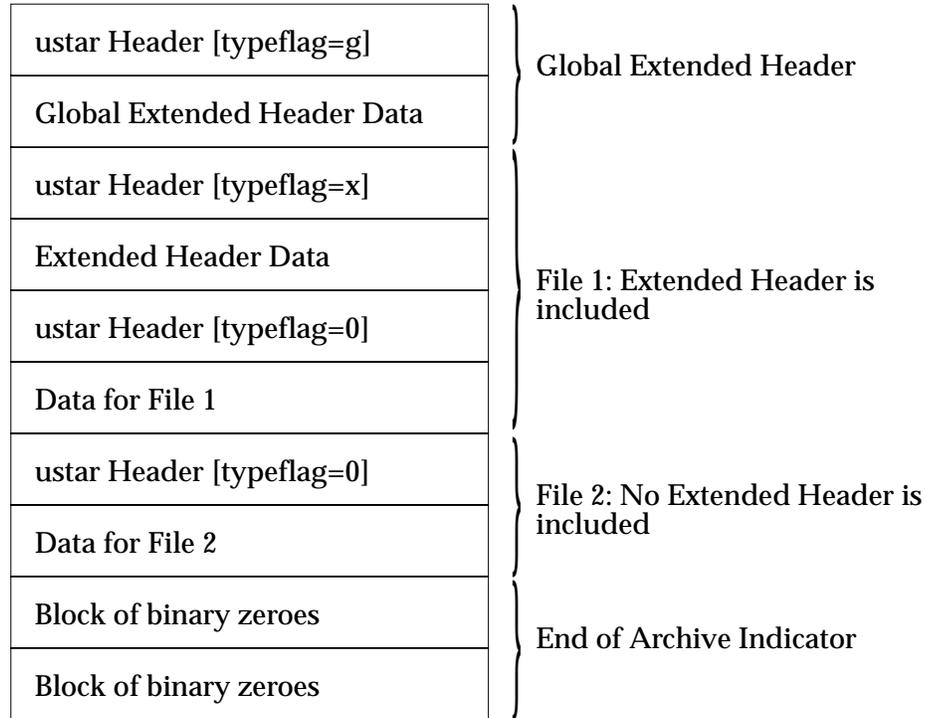
27386 A *pax* archive tape or file produced in the **-xpax** format shall contain a series of blocks. The  
 27387 physical layout of the archive shall be identical to the **ustar** format described in **ustar**  
 27388 **Interchange Format** (on page 2916). Each file archived shall be represented by the following  
 27389 sequence:

- 27390 • An optional header block with extended header records. This header block is of the form  
 27391 described in **pax Header Block** (on page 2912), with a *typeflag* value of **x** or **g**. The extended  
 27392 header records, described in **pax Extended Header** (on page 2913), shall be included as the  
 27393 data for this header block.
- 27394 • A header block that describes the file. Any fields in the preceding optional extended header  
 27395 shall override the associated fields in this header block for this file.
- 27396 • Zero or more blocks that contain the contents of the file.

27397 At the end of the archive file there shall be two 512-byte blocks filled with binary zeroes,  
 27398 interpreted as an end-of-archive indicator.

27399 A schematic of an example archive with global extended header records and two actual files is  
 27400 shown in Figure 4-1 (on page 2912). In the example, the second file in the archive has no  
 27401 extended header preceding it, presumably because it has no need for extended attributes.

27402



27403

Figure 4-1 pax Format Archive Example

27404

**pax Header Block**

27405  
27406

The *pax* header block shall be identical to the **ustar** header block described in **ustar Interchange Format** (on page 2916), except that two additional *typeflag* values are defined:

27407  
27408  
27409

x Represents extended header records for the following file in the archive (which shall have its own **ustar** header block). The format of these extended header records shall be as described in **pax Extended Header** (on page 2913).

27410  
27411  
27412  
27413  
27414  
27415

g Represents global extended header records for the following files in the archive. The format of these extended header records shall be as described in **pax Extended Header** (on page 2913). Each value shall affect all subsequent files that do not override that value in their own extended header record and until another global extended header record is reached that provides another value for the same field. The *typeflag g* global headers should not be used with interchange media that could suffer partial data loss in transporting the archive.

27416  
27417  
27418  
27419  
27420  
27421

For both of these types, the *size* field shall be the size of the extended header records in octets. The other fields in the header block are not meaningful to this version of the *pax* utility. However, if this archive is read by a *pax* utility conforming to a previous version of IEEE Std 1003.1-200x, the header block fields are used to create a regular file that contains the extended header records as data. Therefore, header block field values should be selected to provide reasonable file access to this regular file.

27422  
27423  
27424

A further difference from the **ustar** header block is that data blocks for files of *typeflag 1* (the digit one) (hard link) may be included, which means that the *size* field may be greater than zero. Archives created by **pax -o linkdata** shall include these data blocks with the hard links.

27425 **pax Extended Header**

27426 A *pax* extended header contains values that are inappropriate for the **ustar** header block because  
 27427 of limitations in that format: fields requiring a character encoding other than that described in  
 27428 the ISO/IEC 646:1991 standard, fields representing file attributes not described in the **ustar**  
 27429 header, and fields whose format or length do not fit the requirements of the **ustar** header. The  
 27430 values in an extended header add attributes to the following file (or files; see the description of  
 27431 the *typelflag g* header block) or override values in the following header block(s), as indicated in  
 27432 the following list of keywords.

27433 An extended header shall consist of one or more records, each constructed as follows:

27434 "%d %s=%s\n", <length>, <keyword>, <value>

27435 The extended header records shall be encoded according to the ISO/IEC 10646-1:2000 standard  
 27436 (UTF-8). The <length> field, <blank>, equals sign, and <newline> shown shall be limited to the  
 27437 portable character set, as encoded in UTF-8. The <keyword> and <value> fields can be any UTF-8  
 27438 characters. The <length> field shall be the decimal length of the extended header record in octets,  
 27439 including the trailing <newline>.

27440 The <keyword> field shall be one of the entries from the following list or a keyword provided as  
 27441 an implementation extension. Keywords consisting entirely of lowercase letters, digits, and  
 27442 periods are reserved for future standardization. A keyword shall not include an equals sign. (In  
 27443 the following list, the notations “file(s)” or “block(s)” is used to acknowledge that a keyword  
 27444 affects the following single file after a *typelflag x* extended header, but possibly multiple files after  
 27445 *typelflag g*. Any requirements in the list for *pax* to include a record when in **write** or **copy** mode  
 27446 shall apply only when such a record has not already been provided through the use of the **-o**  
 27447 option. When used in **copy** mode, *pax* shall behave as if an archive had been created with  
 27448 applicable extended header records and then extracted.)

27449 **atime** The file access time for the following file(s), equivalent to the value of the *st\_atime*  
 27450 member of the **stat** structure for a file, as described by the *stat()* function. The  
 27451 access time shall be restored if the process has the appropriate privilege required  
 27452 to do so. The format of the <value> shall be as described in **pax Extended Header**  
 27453 **File Times** (on page 2916).

27454 **charset** The name of the character set used to encode the data in the following file(s). The  
 27455 entries in the following table are defined to refer to known standards; additional  
 27456 names may be agreed on between the originator and recipient.

27457  
27458  
27459  
27460  
27461  
27462  
27463  
27464  
27465  
27466  
27467  
27468  
27469  
27470  
27471  
27472  
27473  
27474  
27475

| <value>                 | Formal Standard               |
|-------------------------|-------------------------------|
| ISO-IRΔ646Δ1990         | ISO/IEC 646: 1990             |
| ISO-IRΔ8859Δ1Δ1998      | ISO/IEC 8859-1: 1998          |
| ISO-IRΔ8859Δ2Δ1999      | ISO/IEC 8859-2: 1999          |
| ISO-IRΔ8859Δ3Δ1999      | ISO/IEC 8859-3: 1999          |
| ISO-IRΔ8859Δ4Δ1998      | ISO/IEC 8859-4: 1998          |
| ISO-IRΔ8859Δ5Δ1999      | ISO/IEC 8859-5: 1999          |
| ISO-IRΔ8859Δ6Δ1999      | ISO/IEC 8859-6: 1999          |
| ISO-IRΔ8859Δ7Δ1987      | ISO/IEC 8859-7: 1987          |
| ISO-IRΔ8859Δ8Δ1999      | ISO/IEC 8859-8: 1999          |
| ISO-IRΔ8859Δ9Δ1999      | ISO/IEC 8859-9: 1999          |
| ISO-IRΔ8859Δ10Δ1998     | ISO/IEC 8859-10: 1998         |
| ISO-IRΔ8859Δ13Δ1998     | ISO/IEC 8859-13: 1998         |
| ISO-IRΔ8859Δ14Δ1998     | ISO/IEC 8859-14: 1998         |
| ISO-IRΔ8859Δ15Δ1999     | ISO/IEC 8859-15: 1999         |
| ISO-IRΔ10646Δ2000       | ISO/IEC 10646: 2000           |
| ISO-IRΔ10646Δ2000ΔUTF-8 | ISO/IEC 10646, UTF-8 encoding |
| BINARY                  | None.                         |

27476  
27477  
27478

The encoding is included in an extended header for information only; when *pax* is used as described in IEEE Std 1003.1-200x, it shall not translate the file data into any other encoding. The **BINARY** entry indicates unencoded binary data.

27479  
27480

When used in **write** or **copy** mode, it is implementation-defined whether *pax* includes a **charset** extended header record for a file.

27481 **comment**  
27482

A series of characters used as a comment. All characters in the <value> field shall be ignored by *pax*.

27483 **ctime**  
27484  
27485  
27486  
27487

The file creation time for the following file(s), equivalent to the value of the *st\_ctime* member of the **stat** structure for a file, as described by the *stat()* function. The creation time shall be restored if the process has the appropriate privilege required to do so. The format of the <value> shall be as described in **pax Extended Header File Times** (on page 2916).

27488 **gid**  
27489  
27490  
27491  
27492

The group ID of the group that owns the file, expressed as a decimal number using digits from the ISO/IEC 646: 1991 standard. This record shall override the *gid* field in the following header block(s). When used in **write** or **copy** mode, *pax* shall include a *gid* extended header record for each file whose group ID is greater than 2 097 151 (octal 7 777 777).

27493 **gname**  
27494  
27495  
27496  
27497  
27498  
27499  
27500  
27501  
27502

The group of the file(s), formatted as a group name in the group database. This record shall override the *gid* and *gname* fields in the following header block(s), and any *gid* extended header record. When used in **read**, **copy**, or **list** mode, *pax* shall translate the name from the UTF-8 encoding in the header record to the character set appropriate for the group database on the receiving system. If any of the UTF-8 characters cannot be translated, and if the **-oinvalid=UTF-8** option is not specified, the results are implementation-defined. When used in **write** or **copy** mode, *pax* shall include a **gname** extended header record for each file whose group name cannot be represented entirely with the letters and digits of the portable character set.

27503 **linkpath**  
27504  
27505

The pathname of a link being created to another file, of any type, previously archived. This record shall override the *linkname* field in the following **ustar** header block(s). The following **ustar** header block shall determine the type of link created.

- 27506 If *typeflag* of the following header block is 1, it shall be a hard link. If *typeflag* is 2, it  
 27507 shall be a symbolic link and the **linkpath** value shall be the contents of the  
 27508 symbolic link. The *pax* utility shall translate the name of the link (contents of the  
 27509 symbolic link) from the UTF-8 encoding to the character set appropriate for the  
 27510 local file system. When used in **write** or **copy** mode, *pax* shall include a **linkpath**  
 27511 extended header record for each link whose pathname cannot be represented  
 27512 entirely with the members of the portable character set other than NUL.
- 27513 **mtime** The file modification time of the following file(s), equivalent to the value of the  
 27514 *st\_mtime* member of the **stat** structure for a file, as described in the *stat()* function.  
 27515 This record shall override the *mtime* field in the following header block(s). The  
 27516 modification time shall be restored if the process has the appropriate privilege  
 27517 required to do so. The format of the *<value>* shall be as described in **pax Extended**  
 27518 **Header File Times** (on page 2916).
- 27519 **path** The pathname of the following file(s). This record shall override the *name* and  
 27520 *prefix* fields in the following header block(s). The *pax* utility shall translate the  
 27521 pathname of the file from the UTF-8 encoding to the character set appropriate for  
 27522 the local file system.
- 27523 When used in **write** or **copy** mode, *pax* shall include a *path* extended header record  
 27524 for each file whose pathname cannot be represented entirely with the members of  
 27525 the portable character set other than NUL.
- 27526 **realtime.any** The keywords prefixed by “realtime.” are reserved for future standardization.
- 27527 **security.any** The keywords prefixed by “security.” are reserved for future standardization.
- 27528 **size** The size of the file in octets, expressed as a decimal number using digits from the  
 27529 ISO/IEC 646:1991 standard. This record shall override the *size* field in the  
 27530 following header block(s). When used in **write** or **copy** mode, *pax* shall include a  
 27531 *size* extended header record for each file with a size value greater than 8 589 934 591  
 27532 (octal 7 777 777 777).
- 27533 **uid** The user ID of the file owner, expressed as a decimal number using digits from the  
 27534 ISO/IEC 646:1991 standard. This record shall override the *uid* field in the  
 27535 following header block(s). When used in **write** or **copy** mode, *pax* shall include a  
 27536 *uid* extended header record for each file whose owner ID is greater than 2 097 151  
 27537 (octal 7 777 777).
- 27538 **uname** The owner of the following file(s), formatted as a user name in the user database.  
 27539 This record shall override the *uid* and *uname* fields in the following header block(s),  
 27540 and any *uid* extended header record. When used in **read**, **copy**, or **list** mode, *pax*  
 27541 shall translate the name from the UTF-8 encoding in the header record to the  
 27542 character set appropriate for the user database on the receiving system. If any of  
 27543 the UTF-8 characters cannot be translated, and if the **-oinvalid=** UTF-8 option is  
 27544 not specified, the results are implementation-defined. When used in **write** or **copy**  
 27545 mode, *pax* shall include a **uname** extended header record for each file whose user  
 27546 name cannot be represented entirely with the letters and digits of the portable  
 27547 character set.
- 27548 If the *<value>* field is zero length, it shall delete any header block field, previously entered  
 27549 extended header value, or global extended header value of the same name.
- 27550 If a keyword in an extended header record (or in a **-o** option-argument) overrides or deletes a  
 27551 corresponding field in the **ustar** header block, *pax* shall ignore the contents of that header block  
 27552 field.

27553 Unlike the **ustar** header block fields, NULs shall not delimit *<value>*s; all characters within the  
 27554 *<value>* field shall be considered data for the field. None of the length limitations of the **ustar**  
 27555 header block fields in Table 4-13 (on page 2917) shall apply to the extended header records.

#### 27556 **pax Extended Header Keyword Precedence**

27557 This section describes the precedence in which the various header records and fields and  
 27558 command line options are selected to apply to a file in the archive. When *pax* is used in **read** or  
 27559 **list** modes, it shall determine a file attribute in the following sequence:

- 27560 1. If **-odelete=keyword-prefix** is used, the affected attributes shall be determined from step 7.,  
 27561 if applicable, or ignored otherwise.
- 27562 2. If **-okeyword:=** is used, the affected attributes shall be ignored.
- 27563 3. If **-okeyword:=value** is used, the affected attribute shall be assigned the value.
- 27564 4. If there is a *typeflag* **x** extended header record, the affected attribute shall be assigned the  
 27565 *<value>*. When extended header records conflict, the last one given in the header shall take  
 27566 precedence.
- 27567 5. If **-okeyword=value** is used, the affected attribute shall be assigned the value.
- 27568 6. If there is a *typeflag* **g** global extended header record, the affected attribute shall be  
 27569 assigned the *<value>*. When global extended header records conflict, the last one given in  
 27570 the global header shall take precedence.
- 27571 7. Otherwise, the attribute shall be determined from the **ustar** header block.

#### 27572 **pax Extended Header File Times**

27573 The *pax* utility shall write an **mtime** record for each file in **write** or **copy** modes if the file's  
 27574 modification time cannot be represented exactly in the **ustar** header logical record described in  
 27575 **ustar Interchange Format**. This can occur if the time is out of **ustar** range, or if the file system of  
 27576 the underlying implementation supports non-integer time granularities and the time is not an  
 27577 integer. All of these time records shall be formatted as a decimal representation of the time in  
 27578 seconds since the Epoch. If a period ( ' . ' ) decimal point character is present, the digits to the  
 27579 right of the point shall represent the units of a subsecond timing granularity, where the first digit  
 27580 is tenths of a second and each subsequent digit is a tenth of the previous digit. In **read** or **copy**  
 27581 mode, the *pax* utility shall truncate the time of a file to the greatest value that is not greater than  
 27582 the input header file time. In **write** or **copy** mode, the *pax* utility shall output a time exactly if it  
 27583 can be represented exactly as a decimal number, and otherwise shall generate only enough digits  
 27584 so that the same time shall be recovered if the file is extracted on a system whose underlying  
 27585 implementation supports the same time granularity.

#### 27586 **ustar Interchange Format**

27587 A **ustar** archive tape or file shall contain a series of logical records. Each logical record shall be a  
 27588 fixed-size logical record of 512 octets (see below). Although this format may be thought of as  
 27589 being stored on 9-track industry-standard 12.7mm (0.5in) magnetic tape, other types of  
 27590 transportable media are not excluded. Each file archived shall be represented by a header logical  
 27591 record that describes the file, followed by zero or more logical records that give the contents of  
 27592 the file. At the end of the archive file there shall be two 512-octet logical records filled with  
 27593 binary zeros, interpreted as an end-of-archive indicator.

27594 The logical records may be grouped for physical I/O operations, as described under the  
 27595 **-bblocksize** and **-x ustar** options. Each group of logical records may be written with a single  
 27596 operation equivalent to the *write()* function. On magnetic tape, the result of this write shall be a

27597 single tape physical block. The last physical block shall always be the full size, so logical records  
27598 after the two zero logical records may contain undefined data.

27599 The header logical record shall be structured as shown in the following table. All lengths and  
27600 offsets are in decimal.

27601

**Table 4-13** ustar Header Block

27602

| Field Name      | Octet Offset | Length (in Octets) |
|-----------------|--------------|--------------------|
| <i>name</i>     | 0            | 100                |
| <i>mode</i>     | 100          | 8                  |
| <i>uid</i>      | 108          | 8                  |
| <i>gid</i>      | 116          | 8                  |
| <i>size</i>     | 124          | 12                 |
| <i>mtime</i>    | 136          | 12                 |
| <i>chksum</i>   | 148          | 8                  |
| <i>typeflag</i> | 156          | 1                  |
| <i>linkname</i> | 157          | 100                |
| <i>magic</i>    | 257          | 6                  |
| <i>version</i>  | 263          | 2                  |
| <i>uname</i>    | 265          | 32                 |
| <i>gname</i>    | 297          | 32                 |
| <i>devmajor</i> | 329          | 8                  |
| <i>devminor</i> | 337          | 8                  |
| <i>prefix</i>   | 345          | 155                |

27603

27604

27605

27606

27607

27608

27609

27610

27611

27612

27613

27614

27615

27616

27617

27618

27619 All characters in the header logical record shall be represented in the coded character set of the  
27620 ISO/IEC 646:1991 standard. For maximum portability between implementations, names should  
27621 be selected from characters represented by the portable filename character set as octets with the  
27622 most significant bit zero. If an implementation supports the use of characters outside of slash  
27623 and the portable filename character set in names for files, users, and groups, one or more  
27624 implementation-defined encodings of these characters shall be provided for interchange  
27625 purposes.

27626 However, the *pax* utility shall never create filenames on the local system that cannot be accessed  
27627 via the procedures described in IEEE Std 1003.1-200x. If a filename is found on the medium that  
27628 would create an invalid filename, it is implementation-defined whether the data from the file is  
27629 stored on the file hierarchy and under what name it is stored. The *pax* utility may choose to  
27630 ignore these files as long as it produces an error indicating that the file is being ignored.

27631 Each field within the header logical record is contiguous; that is, there is no padding used. Each  
27632 character on the archive medium shall be stored contiguously.

27633 The fields *magic*, *uname*, and *gname* are character strings each terminated by a NUL character.  
27634 The fields *name*, *linkname*, and *prefix* are NUL-terminated character strings except when all  
27635 characters in the array contain non-NUL characters including the last character. The *version* field  
27636 is two octets containing the characters "00" (zero-zero). The *typeflag* contains a single character.  
27637 All other fields are leading zero-filled octal numbers using digits from the ISO/IEC 646:1991  
27638 standard IRV. Each numeric field is terminated by one or more <space> or NUL characters.

27639 The *name* and the *prefix* fields shall produce the pathname of the file. A new pathname shall be  
27640 formed, if *prefix* is not an empty string (its first character is not NUL), by concatenating *prefix* (up  
27641 to the first NUL character), a slash character, and *name*; otherwise, *name* is used alone. In either  
27642 case, *name* is terminated at the first NUL character. If *prefix* begins with a NUL character, it shall  
27643 be ignored. In this manner, pathnames of at most 256 characters can be supported. If a pathname

27644 does not fit in the space provided, *pax* shall notify the user of the error, and shall not store any  
 27645 part of the file—header or data—on the medium.

27646 The *linkname* field, described below, shall not use the *prefix* to produce a pathname. As such, a  
 27647 *linkname* is limited to 100 characters. If the name does not fit in the space provided, *pax* shall  
 27648 notify the user of the error, and shall not attempt to store the link on the medium.

27649 The *mode* field provides 12 bits encoded in the ISO/IEC 646:1991 standard octal digit  
 27650 representation. The encoded bits shall represent the following values:

27651 **Table 4-14** ustar *mode* Field

| Bit Value | IEEE Std 1003.1-200x Bit | Description                                     |
|-----------|--------------------------|-------------------------------------------------|
| 04 000    | S_ISUID                  | Set UID on execution.                           |
| 02 000    | S_ISGID                  | Set GID on execution.                           |
| 01 000    | <reserved>               | Reserved for future standardization.            |
| 00 400    | S_IRUSR                  | Read permission for file owner class.           |
| 00 200    | S_IWUSR                  | Write permission for file owner class.          |
| 00 100    | S_IXUSR                  | Execute/search permission for file owner class. |
| 00 040    | S_IRGRP                  | Read permission for file group class.           |
| 00 020    | S_IWGRP                  | Write permission for file group class.          |
| 00 010    | S_IXGRP                  | Execute/search permission for file group class. |
| 00 004    | S_IROTH                  | Read permission for file other class.           |
| 00 002    | S_IWOTH                  | Write permission for file other class.          |
| 00 001    | S_IXOTH                  | Execute/search permission for file other class. |

27665 When appropriate privilege is required to set one of these mode bits, and the user restoring the  
 27666 files from the archive does not have the appropriate privilege, the mode bits for which the user  
 27667 does not have appropriate privilege shall be ignored. Some of the mode bits in the archive  
 27668 format are not mentioned elsewhere in this volume of IEEE Std 1003.1-200x. If the  
 27669 implementation does not support those bits, they may be ignored.

27670 The *uid* and *gid* fields are the user and group ID of the owner and group of the file, respectively.

27671 The *size* field is the size of the file in octets. If the *typeflag* field is set to specify a file to be of type |  
 27672 1 (a link) or 2 (a symbolic link), the *size* field shall be specified as zero. If the *typeflag* field is set to |  
 27673 specify a file of type 5 (directory), the *size* field shall be interpreted as described under the  
 27674 definition of that record type. No data logical records are stored for types 1, 2, or 5. If the *typeflag*  
 27675 field is set to 3 (character special file), 4 (block special file), or 6 (FIFO), the meaning of the *size*  
 27676 field is unspecified by this volume of IEEE Std 1003.1-200x, and no data logical records shall be  
 27677 stored on the medium. Additionally, for type 6, the *size* field shall be ignored when reading. If  
 27678 the *typeflag* field is set to any other value, the number of logical records written following the  
 27679 header shall be  $(size+511)/512$ , ignoring any fraction in the result of the division.

27680 The *mtime* field shall be the modification time of the file at the time it was archived. It is the  
 27681 ISO/IEC 646:1991 standard representation of the octal value of the modification time obtained  
 27682 from the *stat()* function.

27683 The *chksum* field shall be the ISO/IEC 646:1991 standard IRV representation of the octal value of  
 27684 the simple sum of all octets in the header logical record. Each octet in the header shall be treated  
 27685 as an unsigned value. These values shall be added to an unsigned integer, initialized to zero, the  
 27686 precision of which is not less than 17 bits. When calculating the checksum, the *chksum* field is  
 27687 treated as if it were all spaces.

27688 The *typeflag* field specifies the type of file archived. If a particular implementation does not  
 27689 recognize the type, or the user does not have appropriate privilege to create that type, the file

|       |       |                                                                                                                      |
|-------|-------|----------------------------------------------------------------------------------------------------------------------|
| 27690 |       | shall be extracted as if it were a regular file if the file type is defined to have a meaning for the                |
| 27691 |       | <i>size</i> field that could cause data logical records to be written on the medium (see the previous                |
| 27692 |       | description for <i>size</i> ). If conversion to a regular file occurs, the <i>pax</i> utility shall produce an error |
| 27693 |       | indicating that the conversion took place. All of the <i>typeflag</i> fields shall be coded in the                   |
| 27694 |       | ISO/IEC 646:1991 standard IRV:                                                                                       |
| 27695 | 0     | Represents a regular file. For backward compatibility, a <i>typeflag</i> value of binary zero                        |
| 27696 |       | (' \0 ') should be recognized as meaning a regular file when extracting files from the                               |
| 27697 |       | archive. Archives written with this version of the archive file format create regular files                          |
| 27698 |       | with a <i>typeflag</i> value of the ISO/IEC 646:1991 standard IRV ' 0 '.                                             |
| 27699 | 1     | Represents a file linked to another file, of any type, previously archived. Such files are                           |
| 27700 |       | identified by each file having the same device and file serial number. The linked-to                                 |
| 27701 |       | name is specified in the <i>linkname</i> field with a NUL-character terminator if it is less than                    |
| 27702 |       | 100 octets in length.                                                                                                |
| 27703 | 2     | Represents a symbolic link. The contents of the symbolic link shall be stored in the                                 |
| 27704 |       | <i>linkname</i> field.                                                                                               |
| 27705 | 3 , 4 | Represent character special files and block special files respectively. In this case the                             |
| 27706 |       | <i>devmajor</i> and <i>devminor</i> fields shall contain information defining the device, the format                 |
| 27707 |       | of which is unspecified by this volume of IEEE Std 1003.1-200x. Implementations may                                  |
| 27708 |       | map the device specifications to their own local specification or may ignore the entry.                              |
| 27709 | 5     | Specifies a directory or subdirectory. On systems where disk allocation is performed on                              |
| 27710 |       | a directory basis, the <i>size</i> field shall contain the maximum number of octets (which may                       |
| 27711 |       | be rounded to the nearest disk block allocation unit) that the directory may hold. A <i>size</i>                     |
| 27712 |       | field of zero indicates no such limiting. Systems that do not support limiting in this                               |
| 27713 |       | manner should ignore the <i>size</i> field.                                                                          |
| 27714 | 6     | Specifies a FIFO special file. Note that the archiving of a FIFO file archives the existence                         |
| 27715 |       | of this file and not its contents.                                                                                   |
| 27716 | 7     | Reserved to represent a file to which an implementation has associated some high-                                    |
| 27717 |       | performance attribute. Implementations without such extensions should treat this file                                |
| 27718 |       | as a regular file (type 0).                                                                                          |
| 27719 | A-Z   | The letters 'A' to 'Z', inclusive, are reserved for custom implementations. All other                                |
| 27720 |       | values are reserved for future versions of IEEE Std 1003.1-200x.                                                     |
| 27721 |       | Attempts to archive a socket using <b>ustar</b> interchange format shall produce a diagnostic message.               |
| 27722 |       | Handling of other file types is implementation-defined.                                                              |
| 27723 |       | The <i>magic</i> field is the specification that this archive was output in this archive format. If this field       |
| 27724 |       | contains <b>ustar</b> (the five characters from the ISO/IEC 646:1991 standard IRV shown followed by                  |
| 27725 |       | NUL), the <i>uname</i> and <i>gname</i> fields shall contain the ISO/IEC 646:1991 standard IRV                       |
| 27726 |       | representation of the owner and group of the file, respectively (truncated to fit, if necessary).                    |
| 27727 |       | When the file is restored by a privileged, protection-preserving version of the utility, the user                    |
| 27728 |       | and group databases shall be scanned for these names. If found, the user and group IDs                               |
| 27729 |       | contained within these files shall be used rather than the values contained within the <i>uid</i> and <i>gid</i>     |
| 27730 |       | fields.                                                                                                              |

27731 **cpio Interchange Format**

27732 The octet-oriented *cpio* archive format shall be a series of entries, each comprising a header that  
 27733 describes the file, the name of the file, and then the contents of the file.

27734 An archive may be recorded as a series of fixed-size blocks of octets. This blocking shall be used  
 27735 only to make physical I/O more efficient. The last group of blocks shall be always at the full  
 27736 size.

27737 For the octet-oriented *cpio* archive format, the individual entry information shall be in the order  
 27738 indicated and described by the following table; see also the <*cpio.h*> header.

27739 **Table 4-15** Octet-Oriented *cpio* Archive Entry

| Header Field Name       | Length (in Octets) | Interpreted as  |
|-------------------------|--------------------|-----------------|
| 27740 <i>c_magic</i>    | 6                  | Octal number    |
| 27741 <i>c_dev</i>      | 6                  | Octal number    |
| 27742 <i>c_ino</i>      | 6                  | Octal number    |
| 27743 <i>c_mode</i>     | 6                  | Octal number    |
| 27744 <i>c_uid</i>      | 6                  | Octal number    |
| 27745 <i>c_gid</i>      | 6                  | Octal number    |
| 27746 <i>c_nlink</i>    | 6                  | Octal number    |
| 27747 <i>c_rdev</i>     | 6                  | Octal number    |
| 27748 <i>c_mtime</i>    | 11                 | Octal number    |
| 27749 <i>c_namesize</i> | 6                  | Octal number    |
| 27750 <i>c_filesize</i> | 11                 | Octal number    |
| Filename Field Name     | Length             | Interpreted as  |
| 27751 <i>c_name</i>     | <i>c_namesize</i>  | Pathname string |
| File Data Field Name    | Length             | Interpreted as  |
| 27752 <i>c_filedata</i> | <i>c_filesize</i>  | Data            |

27753 **cpio Header**

27754 For each file in the archive, a header as defined previously shall be written. The information in  
 27755 the header fields is written as streams of the ISO/IEC 646: 1991 standard characters interpreted  
 27756 as octal numbers. The octal numbers shall be extended to the necessary length by appending the  
 27757 ISO/IEC 646: 1991 standard IRV zeros at the most-significant-digit end of the number; the result  
 27758 is written to the most-significant digit of the stream of octets first. The fields shall be interpreted  
 27759 as follows:

- 27760 *c\_magic* Identify the archive as being a transportable archive by containing the identifying  
 27761 value "070707".
- 27762 *c\_dev, c\_ino* Contains values that uniquely identify the file within the archive (that is, no files  
 27763 contain the same pair of *c\_dev* and *c\_ino* values unless they are links to the same  
 27764 file). The values shall be determined in an unspecified manner.
- 27765 *c\_mode* Contains the file type and access permissions as defined in the following table.

27769

Table 4-16 Values for *cpio c\_mode* Field

27770

| File Permissions Name | Value    | Indicates              |
|-----------------------|----------|------------------------|
| C_IRUSR               | 000 400  | Read by owner          |
| C_IWUSR               | 000 200  | Write by owner         |
| C_IXUSR               | 000 100  | Execute by owner       |
| C_IRGRP               | 000 040  | Read by group          |
| C_IWGRP               | 000 020  | Write by group         |
| C_IXGRP               | 000 010  | Execute by group       |
| C_IROTH               | 000 004  | Read by others         |
| C_IWOTH               | 000 002  | Write by others        |
| C_IXOTH               | 000 001  | Execute by others      |
| C_ISUID               | 004 000  | Set <i>uid</i>         |
| C_ISGID               | 002 000  | Set <i>gid</i>         |
| C_ISVTX               | 001 000  | Reserved               |
| File Type Name        | Value    | Indicates              |
| C_ISDIR               | 040 000  | Directory              |
| C_ISFIFO              | 010 000  | FIFO                   |
| C_ISREG               | 0100 000 | Regular file           |
| C_ISLNK               | 0120 000 | Symbolic link          |
| C_ISBLK               | 060 000  | Block special file     |
| C_ISCHR               | 020 000  | Character special file |
| C_ISSOCK              | 0140 000 | Socket                 |
| C_ISCTG               | 0110 000 | Reserved               |

27783

27784

27785

27786

27787

27788

27789

27790

27791

27792

27793

27794

27795

27796

Directories, FIFOs, symbolic links, and regular files shall be supported on a system conforming to this volume of IEEE Std 1003.1-200x; additional values defined previously are reserved for compatibility with existing systems. Additional file types may be supported; however, such files should not be written to archives intended to be transported to other systems.

27797

*c\_uid*

Contains the user ID of the owner.

27798

*c\_gid*

Contains the group ID of the group.

27799

*c\_nlink*

Contains the number of links referencing the file at the time the archive was created.

27800

27801

*c\_rdev*

Contains implementation-defined information for character or block special files.

27802

*c\_mtime*

Contains the latest time of modification of the file at the time the archive was created.

27803

27804

*c\_namesize*

Contains the length of the pathname, including the terminating NUL character.

27805

*c\_filesize*

Contains the length of the file in octets. This shall be the length of the data section following the header structure.

27806

- 27807 **cpio Filename**
- 27808 The *c\_name* field shall contain the pathname of the file. The length of this field in octets is the  
27809 value of *c\_namesize*.
- 27810 If a filename is found on the medium that would create an invalid pathname, it is  
27811 implementation-defined whether the data from the file is stored on the file hierarchy and under  
27812 what name it is stored.
- 27813 All characters shall be represented in the ISO/IEC 646:1991 standard IRV. For maximum  
27814 portability between implementations, names should be selected from characters represented by  
27815 the portable filename character set as octets with the most significant bit zero. If an  
27816 implementation supports the use of characters outside the portable filename character set in  
27817 names for files, users, and groups, one or more implementation-defined encodings of these  
27818 characters shall be provided for interchange purposes. However, the *pax* utility shall never  
27819 create filenames on the local system that cannot be accessed via the procedures described  
27820 previously in this volume of IEEE Std 1003.1-200x. If a filename is found on the medium that  
27821 would create an invalid filename, it is implementation-defined whether the data from the file is  
27822 stored on the local file system and under what name it is stored. The *pax* utility may choose to  
27823 ignore these files as long as it produces an error indicating that the file is being ignored.
- 27824 **cpio File Data**
- 27825 Following *c\_name*, there shall be *c\_filesize* octets of data. Interpretation of such data occurs in a  
27826 manner dependent on the file. If *c\_filesize* is zero, no data shall be contained in *c\_filedata*.
- 27827 When restoring from an archive:
- 27828 • If the user does not have the appropriate privilege to create a file of the specified type, *pax*  
27829 shall ignore the entry and write an error message to standard error.
  - 27830 • Only regular files have data to be restored. Presuming a regular file meets any selection  
27831 criteria that might be imposed on the format-reading utility by the user, such data shall be  
27832 restored.
  - 27833 • If a user does not have appropriate privilege to set a particular mode flag, the flag shall be  
27834 ignored. Some of the mode flags in the archive format are not mentioned elsewhere in this  
27835 volume of IEEE Std 1003.1-200x. If the implementation does not support those flags, they  
27836 may be ignored.
- 27837 **cpio Special Entries**
- 27838 FIFO special files, directories, and the trailer shall be recorded with *c\_filesize* equal to zero. For  
27839 other special files, *c\_filesize* is unspecified by this volume of IEEE Std 1003.1-200x. The header for  
27840 the next file entry in the archive shall be written directly after the last octet of the file entry  
27841 preceding it. A header denoting the filename **TRAILER!!!** shall indicate the end of the archive; |  
27842 the contents of octets in the last block of the archive following such a header are undefined. |
- 27843 **EXIT STATUS**
- 27844 The following exit values shall be returned:
- 27845 0 All files were processed successfully.
  - 27846 >0 An error occurred.

27847 **CONSEQUENCES OF ERRORS**

27848 If *pax* cannot create a file or a link when reading an archive or cannot find a file when writing an  
 27849 archive, or cannot preserve the user ID, group ID, or file mode when the **-p** option is specified, a  
 27850 diagnostic message shall be written to standard error and a non-zero exit status shall be  
 27851 returned, but processing shall continue. In the case where *pax* cannot create a link to a file, *pax*  
 27852 shall not, by default, create a second copy of the file.

27853 If the extraction of a file from an archive is prematurely terminated by a signal or error, *pax* may  
 27854 have only partially extracted the file or (if the **-n** option was not specified) may have extracted a  
 27855 file of the same name as that specified by the user, but which is not the file the user wanted.  
 27856 Additionally, the file modes of extracted directories may have additional bits from the S\_IRWXU  
 27857 mask set as well as incorrect modification and access times.

27858 **APPLICATION USAGE**

27859 The **-p** (privileges) option was invented to reconcile differences between historical *tar* and *cpio*  
 27860 implementations. In particular, the two utilities use **-m** in diametrically opposed ways. The **-p**  
 27861 option also provides a consistent means of extending the ways in which future file attributes can  
 27862 be addressed, such as for enhanced security systems or high-performance files. Although it may  
 27863 seem complex, there are really two modes that are most commonly used:

27864 **-p e** “Preserve everything”. This would be used by the historical superuser, someone with  
 27865 all the appropriate privileges, to preserve all aspects of the files as they are recorded in  
 27866 the archive. The **e** flag is the sum of **o** and **p**, and other implementation-defined  
 27867 attributes.

27868 **-p p** “Preserve” the file mode bits. This would be used by the user with regular privileges  
 27869 who wished to preserve aspects of the file other than the ownership. The file times are  
 27870 preserved by default, but two other flags are offered to disable these and use the time  
 27871 of extraction.

27872 The one pathname per line format of standard input precludes pathnames containing  
 27873 <newline>s. Although such pathnames violate the portable filename guidelines, they may exist  
 27874 and their presence may inhibit usage of *pax* within shell scripts. This problem is inherited from  
 27875 historical archive programs. The problem can be avoided by listing filename arguments on the  
 27876 command line instead of on standard input.

27877 It is almost certain that appropriate privileges are required for *pax* to accomplish parts of this  
 27878 volume of IEEE Std 1003.1-200x. Specifically, creating files of type block special or character  
 27879 special, restoring file access times unless the files are owned by the user (the **-t** option), or  
 27880 preserving file owner, group, and mode (the **-p** option) all probably require appropriate  
 27881 privileges.

27882 In **read** mode, implementations are permitted to overwrite files when the archive has multiple  
 27883 members with the same name. This may fail if permissions on the first version of the file do not  
 27884 permit it to be overwritten.

27885 The **cpio** and **ustar** formats can only support files up to 8589934592 bytes ( $8 * 2^{30}$ ) in size. |

27886 **EXAMPLES**

27887 The following command:

```
27888 pax -w -f /dev/rmt/1m .
```

27889 copies the contents of the current directory to tape drive 1, medium density (assuming historical  
 27890 System V device naming procedures. The historical BSD device name would be **/dev/rmt9**).

27891 The following commands:

```

27892      mkdir newdir
27893      pax -rw olddir newdir
27894      copy the olddir directory hierarchy to newdir.
27895      pax -r -s ',^//*usr//*,,' -f a.pax
27896      reads the archive a.pax, with all files rooted in /usr in the archive extracted relative to the current
27897      directory.
27898      Using the option:
27899      -o listopt="%M %(atime)T %(size)D %(name)s"
27900      overrides the default output description in Standard Output and instead writes:
27901      -rw-rw--- Jan 12 15:53 1492 /usr/foo/bar
27902      Using the options:
27903      -o listopt='%L\t%(size)D\n%.7' \
27904      -o listopt='(name)s\n%(ctime)T\n%T'
27905      overrides the default output description in Standard Output and instead writes:
27906      /usr/foo/bar -> /tmp 1492
27907      /usr/fo
27908      Jan 12 1991
27909      Jan 31 15:53
27910 RATIONALE
27911      The pax utility was new, commissioned for the ISO POSIX-2:1993 standard. It represents a
27912      peaceful compromise between advocates of the historical tar and cpio utilities.
27913      A fundamental difference between cpio and tar was in the way directories were treated. The cpio
27914      utility did not treat directories differently from other files, and to select a directory and its
27915      contents required that each file in the hierarchy be explicitly specified. For tar, a directory
27916      matched every file in the file hierarchy it rooted.
27917      The pax utility offers both interfaces; by default, directories map into the file hierarchy they root.
27918      The -d option causes pax to skip any file not explicitly referenced, as cpio historically did. The tar
27919      -style behavior was chosen as the default because it was believed that this was the more
27920      common usage and because tar is the more commonly available interface, as it was historically
27921      provided on both System V and BSD implementations.
27922      The data interchange format specification in this volume of IEEE Std 1003.1-200x requires that
27923      processes with "appropriate privileges" shall always restore the ownership and permissions of
27924      extracted files exactly as archived. If viewed from the historic equivalence between superuser
27925      and "appropriate privileges", there are two problems with this requirement. First, users running
27926      as superusers may unknowingly set dangerous permissions on extracted files. Second, it is
27927      needlessly limiting, in that superusers cannot extract files and own them as superuser unless the
27928      archive was created by the superuser. (It should be noted that restoration of ownerships and
27929      permissions for the superuser, by default, is historical practice in cpio, but not in tar.) In order to
27930      avoid these two problems, the pax specification has an additional "privilege" mechanism, the -p
27931      option. Only a pax invocation with the privileges needed, and which has the -p option set using
27932      the e specification character, has the "appropriate privilege" to restore full ownership and
27933      permission information.
27934      Note also that this volume of IEEE Std 1003.1-200x requires that the file ownership and access
27935      permissions shall be set, on extraction, in the same fashion as the creat() function when provided

```

27936 the mode stored in the archive. This means that the file creation mask of the user is applied to  
27937 the file permissions.

27938 Users should note that directories may be created by *pax* while extracting files with permissions  
27939 that are different from those that existed at the time the archive was created. When extracting  
27940 sensitive information into a directory hierarchy that no longer exists, users are encouraged to set  
27941 their file creation mask appropriately to protect these files during extraction.

27942 The table of contents output is written to standard output to facilitate pipeline processing.

27943 An early proposal had hard links displaying for all pathnames. This was removed because it  
27944 complicates the output of the case where `-v` is not specified and does not match historical *cpio*  
27945 usage. The hard-link information is available in the `-v` display.

27946 The description of the `-l` option allows implementations to make hard links to symbolic links. |  
27947 IEEE Std 1003.1-200x does not specify any way to create a hard link to a symbolic link, but many |  
27948 implementations provide this capability as an extension. If there are hard links to symbolic links |  
27949 when an archive is created, the implementation is required to archive the hard link in the archive |  
27950 (unless `-H` or `-L` is specified). When in **read** mode and in **copy** mode, implementations |  
27951 supporting hard links to symbolic links should use them when appropriate. |

27952 The archive formats inherited from the POSIX.1-1990 standard have certain restrictions that |  
27953 have been brought along from historical usage. For example, there are restrictions on the length |  
27954 of pathnames stored in the archive. When *pax* is used in **copy**( `-rw`) mode (copying directory |  
27955 hierarchies), the ability to use extensions from the `-xpax` format overcomes these restrictions. |

27956 The default *blocksize* value of 5 120 bytes for *cpio* was selected because it is one of the standard |  
27957 block-size values for *cpio*, set when the `-B` option is specified. (The other default block-size value |  
27958 for *cpio* is 512 bytes, and this was considered to be too small.) The default block value of 10 240 |  
27959 bytes for *tar* was selected because that is the standard block-size value for BSD *tar*. The |  
27960 maximum block size of 32 256 bytes ( $2^{15}$ –512 bytes) is the largest multiple of 512 bytes that fits |  
27961 into a signed 16-bit tape controller transfer register. There are known limitations in some |  
27962 historical systems that would prevent larger blocks from being accepted. Historical values were |  
27963 chosen to improve compatibility with historical scripts using *dd* or similar utilities to manipulate |  
27964 archives. Also, default block sizes for any file type other than character special file has been |  
27965 deleted from this volume of IEEE Std 1003.1-200x as unimportant and not likely to affect the |  
27966 structure of the resulting archive.

27967 Implementations are permitted to modify the block-size value based on the archive format or |  
27968 the device to which the archive is being written. This is to provide implementations with the |  
27969 opportunity to take advantage of special types of devices, and it should not be used without a |  
27970 great deal of consideration as it almost certainly decreases archive portability.

27971 The intended use of the `-n` option was to permit extraction of one or more files from the archive |  
27972 without processing the entire archive. This was viewed by the standard developers as offering |  
27973 significant performance advantages over historical implementations. The `-n` option in early |  
27974 proposals had three effects; the first was to cause special characters in patterns to not be treated |  
27975 specially. The second was to cause only the first file that matched a pattern to be extracted. The |  
27976 third was to cause *pax* to write a diagnostic message to standard error when no file was found |  
27977 matching a specified pattern. Only the second behavior is retained by this volume of |  
27978 IEEE Std 1003.1-200x, for many reasons. First, it is in general not acceptable for a single option to |  
27979 have multiple effects. Second, the ability to make pattern matching characters act as normal |  
27980 characters is useful for parts of *pax* other than file extraction. Third, a finer degree of control over |  
27981 the special characters is useful because users may wish to normalize only a single special |  
27982 character in a single filename. Fourth, given a more general escape mechanism, the previous |  
27983 behavior of the `-n` option can be easily obtained using the `-s` option or a *sed* script. Finally,

- 27984 writing a diagnostic message when a pattern specified by the user is unmatched by any file is  
27985 useful behavior in all cases.
- 27986 In this version, the `-n` was removed from the `copy` mode synopsis of *pax*; it is inapplicable  
27987 because there are no pattern operands specified in this mode.
- 27988 There is another method than *pax* for copying subtrees in IEEE Std 1003.1-200x described as part  
27989 of the *cp* utility. Both methods are historical practice: *cp* provides a simpler, more intuitive  
27990 interface, while *pax* offers a finer granularity of control. Each provides additional functionality to  
27991 the other; in particular, *pax* maintains the hard-link structure of the hierarchy while *cp* does not.  
27992 It is the intention of the standard developers that the results be similar (using appropriate option  
27993 combinations in both utilities). The results are not required to be identical; there seemed  
27994 insufficient gain to applications to balance the difficulty of implementations having to guarantee  
27995 that the results would be exactly identical.
- 27996 A single archive may span more than one file. It is suggested that implementations provide  
27997 informative messages to the user on standard error whenever the archive file is changed.
- 27998 The `-d` option (do not create intermediate directories not listed in the archive) found in early  
27999 proposals was originally provided as a complement to the historic `-d` option of *cpio*. It has been  
28000 deleted.
- 28001 The `-s` option in early proposals specified a subset of the substitution command from the *ed*  
28002 utility. As there was no reason for only a subset to be supported, the `-s` option is now  
28003 compatible with the current *ed* specification. Since the delimiter can be any non-null character,  
28004 the following usage with single spaces is valid:
- 28005 `pax -s " foo bar " ...`
- 28006 The `-t` description is worded so as to note that this may cause the access time update caused by  
28007 some other activity (which occurs while the file is being read) to be overwritten.
- 28008 The default behavior of *pax* with regard to file modification times is the same as historical  
28009 implementations of *tar*. It is not the historical behavior of *cpio*.
- 28010 Because the `-i` option uses `/dev/tty`, utilities without a controlling terminal are not able to use  
28011 this option.
- 28012 The `-y` option, found in early proposals, has been deleted because a line containing a single  
28013 period for the `-i` option has equivalent functionality. The special lines for the `-i` option (a single  
28014 period and the empty line) are historical practice in *cpio*.
- 28015 In early drafts, an `-echarmap` option was included to increase portability of files between systems  
28016 using different coded character sets. This option was omitted because it was apparent that  
28017 consensus could not be formed for it. In this version, the use of UTF-8 should be an adequate  
28018 substitute.
- 28019 The `-k` option was added to address international concerns about the dangers involved in the  
28020 character set transformations of `-e` (if the target character set were different from the source, the  
28021 filenames might be transformed into names matching existing files) and also was made more  
28022 general to protect files transferred between file systems with different `{NAME_MAX}` values  
28023 (truncating a filename on a smaller system might also inadvertently overwrite existing files). As  
28024 stated, it prevents any overwriting, even if the target file is older than the source. This version  
28025 adds more granularity of options to solve this problem by introducing the `-oinvalid=` option—  
28026 specifically the UTF-8 action. (Note that an existing file that is named with a UTF-8 encoding is  
28027 still subject to overwriting in this case. The `-k` option closes that loophole.)
- 28028 Some of the file characteristics referenced in this volume of IEEE Std 1003.1-200x might not be  
28029 supported by some archive formats. For example, neither the *tar* nor *cpio* formats contain the file

28030 access time. For this reason, the **e** specification character has been provided, intended to cause all  
 28031 file characteristics specified in the archive to be retained.

28032 It is required that extracted directories, by default, have their access and modification times and  
 28033 permissions set to the values specified in the archive. This has obvious problems in that the  
 28034 directories are almost certainly modified after being extracted and that directory permissions  
 28035 may not permit file creation. One possible solution is to create directories with the mode  
 28036 specified in the archive, as modified by the *umask* of the user, with sufficient permissions to  
 28037 allow file creation. After all files have been extracted, *pax* would then reset the access and  
 28038 modification times and permissions as necessary.

28039 The list-mode formatting description borrows heavily from the one defined by the *printf* utility.  
 28040 However, since there is no separate operand list to get conversion arguments, the format was  
 28041 extended to allow specifying the name of the conversion argument as part of the conversion  
 28042 specification.

28043 The **T** conversion specifier allows time fields to be displayed in any of the date formats. Unlike  
 28044 the *ls* utility, *pax* does not adjust the format when the date is less than six months in the past.  
 28045 This makes parsing the output more predictable.

28046 The **D** conversion specifier handles the ability to display the major/minor or file size, as with *ls*,  
 28047 by using `%-8(size)D`.

28048 The **L** conversion specifier handles the *ls* display for symbolic links.

28049 Conversion specifiers were added to generate existing known types used for *ls*.

### 28050 **pax Interchange Format**

28051 The new POSIX data interchange format was developed primarily to satisfy international  
 28052 concerns that the **ustar** and *cpio* formats did not provide for file, user, and group names encoded  
 28053 in characters outside a subset of the ISO/IEC 646:1991 standard. The standard developers  
 28054 realized that this new POSIX data interchange format should be very extensible because there  
 28055 were other requirements they foresaw in the near future:

- 28056 • Support international character encodings and locale information
- 28057 • Support security information (ACLs, and so on)
- 28058 • Support future file types, such as realtime or contiguous files
- 28059 • Include data areas for implementation use
- 28060 • Support systems with words larger than 32 bits and timers with subsecond granularity

28061 The following were not goals for this format because these are better handled by separate  
 28062 utilities or are inappropriate for a portable format:

- 28063 • Encryption
- 28064 • Compression
- 28065 • Data translation between locales and codesets
- 28066 • *inode* storage

28067 The format chosen to support the goals is an extension of the **ustar** format. Of the two formats  
 28068 previously available, only the **ustar** format was selected for extensions because:

- 28069 • It was easier to extend in an upward-compatible way. It offered version flags and header  
 28070 block type fields with room for future standardization. The *cpio* format, while possessing a  
 28071 more flexible file naming methodology, could not be extended without breaking some

28072 theoretical implementation or using a dummy filename that could be a legitimate filename.

28073 • Industry experience since the original “tar wars” fought in developing the ISO POSIX-1  
28074 standard has clearly been in favor of the **ustar** format, which is generally the default output  
28075 format selected for *pax* implementations on new systems.

28076 The new format was designed with one additional goal in mind: reasonable behavior when an  
28077 older *tar* or *pax* utility happened to read an archive. Since the POSIX.1-1990 standard mandated  
28078 that a “format-reading utility” had to treat unrecognized *typeflag* values as regular files, this  
28079 allowed the format to include all the extended information in a pseudo-regular file that preceded  
28080 each real file. An option is given that allows the archive creator to set up reasonable names for  
28081 these files on the older systems. Also, the normative text suggests that reasonable file access  
28082 values be used for this **ustar** header block. Making these header files inaccessible for convenient  
28083 reading and deleting would not be reasonable. File permissions of 600 or 700 are suggested.

28084 The **ustar** *typeflag* field was used to accommodate the additional functionality of the new format  
28085 rather than magic or version because the POSIX.1-1990 standard (and, by reference, the previous  
28086 version of *pax*), mandated the behavior of the format-reading utility when it encountered an  
28087 unknown *typeflag*, but was silent about the other two fields.

28088 Early proposals of the first revision to IEEE Std 1003.1-200x contained a proposed archive format  
28089 that was based on compatibility with the standard for tape files (ISO 1001, similar to the format  
28090 used historically on many mainframes and minicomputers). This format was overly complex  
28091 and required considerable overhead in volume and header records. Furthermore, the standard  
28092 developers felt that it would not be acceptable to the community of POSIX developers, so it was  
28093 later changed to be a format more closely related to historical practice on POSIX systems.

28094 The prefix and name split of pathnames in **ustar** was replaced by the single path extended  
28095 header record for simplicity.

28096 The concept of a global extended header (*typeflagg*) was controversial. If this were applied to an  
28097 archive being recorded on magnetic tape, a few unreadable blocks at the beginning of the tape  
28098 could be a serious problem; a utility attempting to extract as many files as possible from a  
28099 damaged archive could lose a large percentage of file header information in this case. However,  
28100 if the archive were on a reliable medium, such as a CD-ROM, the global extended header offers  
28101 considerable potential size reductions by eliminating redundant information. Thus, the text  
28102 warns against using the global method for unreliable media and provides a method for  
28103 implanting global information in the extended header for each file, rather than in the *typeflag g*  
28104 records.

28105 No facility for data translation or filtering on a per-file basis is included because the standard  
28106 developers could not invent an interface that would allow this in an efficient manner. If a filter,  
28107 such as encryption or compression, is to be applied to all the files, it is more efficient to apply the  
28108 filter to the entire archive as a single file. The standard developers considered interfaces that  
28109 would invoke a shell script for each file going into or out of the archive, but the system overhead  
28110 in this approach was considered to be too high.

28111 One such approach would be to have **filter=** records that give a pathname for an executable.  
28112 When the program is invoked, the file and archive would be open for standard input/output  
28113 and all the header fields would be available as environment variables or command-line  
28114 arguments. The standard developers did discuss such schemes, but they were omitted from  
28115 IEEE Std 1003.1-200x due to concerns about excessive overhead. Also, the program itself would  
28116 need to be in the archive if it were to be used portably.

28117 There is currently no portable means of identifying the character set(s) used for a file in the file  
28118 system. Therefore, *pax* has not been given a mechanism to generate charset records  
28119 automatically. The only portable means of doing this is for the user to write the archive using the

28120        –**charset=string** command line option. This assumes that all of the files in the archive use the  
 28121        same encoding. The “implementation-defined” text is included to allow for a system that can  
 28122        identify the encodings used for each of its files.

28123        The table of standards that accompanies the charset record description is acknowledged to be  
 28124        very limited. Only a limited number of character set standards is reasonable for maximal  
 28125        interchange. Any character set is, of course, possible by prior agreement. It was suggested that  
 28126        EBCDIC be listed, but it was omitted because it is not defined by a formal standard. Formal  
 28127        standards, and then only those with reasonably large followings, can be included here, simply as  
 28128        a matter of practicality. The <value>s represent names of officially registered character sets in the  
 28129        format required by the ISO 2375:1985 standard.

28130        The normal comma or <blank>-separated list rules are not followed in the case of keyword  
 28131        options to allow ease of argument parsing for *getopts*.

28132        Further information on character encodings is in **pax Archive Character Set Encoding/Decoding**  
 28133        (on page 2931).

28134        The standard developers have reserved keyword name space for vendor extensions. It is  
 28135        suggested that the format to be used is:

28136        *VENDOR.keyword*

28137        where *VENDOR* is the name of the vendor or organization in all uppercase letters. It is further  
 28138        suggested that the keyword following the period be named differently than any of the standard  
 28139        keywords so that it could be used for future standardization, if appropriate, by omitting the  
 28140        *VENDOR* prefix.

28141        The <length> field in the extended header record was included to make it simpler to step  
 28142        through the records, even if a record contains an unknown format (to a particular *pax*) with  
 28143        complex interactions of special characters. It also provides a minor integrity checkpoint within  
 28144        the records to aid a program attempting to recover files from a damaged archive.

28145        There are no extended header versions of the *devmajor* and *devminor* fields because the  
 28146        unspecified format **ustar** header field should be sufficient. If they are not, vendor-specific  
 28147        extended keywords (such as *VENDOR.devmajor*) should be used.

28148        Device and *i*-number labeling of files was not adopted from *cpio*; files are interchanged strictly  
 28149        on a symbolic name basis, as in **ustar**.

28150        Just as with the **ustar** format descriptions, the new format makes no special arrangements for  
 28151        multi-volume archives. Each of the *pax* archive types is assumed to be inside a single POSIX file  
 28152        and splitting that file over multiple volumes (diskettes, tape cartridges, and so on), processing  
 28153        their labels, and mounting each in the proper sequence are considered to be implementation  
 28154        details that cannot be described portably.

28155        The *pax* format is intended for interchange, not only for backup on a single (family of) systems. It  
 28156        is not as densely packed as might be possible for backup:

28157        

- It contains information as coded characters that could be coded in binary.
- It identifies extended records with name fields that could be omitted in favor of a fixed-field layout.
- It translates names into a portable character set and identifies locale-related information, both of which are probably unnecessary for backup.

28162        The requirements on restoring from an archive are slightly different from the historical wording,  
 28163        allowing for non-monolithic privilege to bring forward as much as possible. In particular,  
 28164        attributes such as “high performance file” might be broadly but not universally granted while

28165 set-user-ID or *chown()* might be much more restricted. There is no implication in  
28166 IEEE Std 1003.1-200x that the security information be honored after it is restored to the file  
28167 hierarchy, in spite of what might be improperly inferred by the silence on that topic. That is a  
28168 topic for another standard.

28169 Links are recorded in the fashion described here because a link can be to any file type. It is  
28170 desirable in general to be able to restore part of an archive selectively and restore all of those  
28171 files completely. If the data is not associated with each link, it is not possible to do this.  
28172 However, the data associated with a file can be large, and when selective restoration is not  
28173 needed, this can be a significant burden. The archive is structured so that files that have no  
28174 associated data can always be restored by the name of any link name of any link, and the user  
28175 may choose whether data is recorded with each instance of a file that contains data. The format  
28176 permits mixing of both types of links in a single archive; this can be done for special needs, and  
28177 *pax* is expected to interpret such archives on input properly, despite the fact that there is no *pax*  
28178 option that would force this mixed case on output. (When **-o linkdata** is used, the output must  
28179 contain the duplicate data, but the implementation is free to include it or omit it when **-o**  
28180 **linkdata** is not used.)

28181 The time values are included as extended header records for those implementations needing  
28182 more than the eleven octal digits allowed by the **ustar** format. Portable file timestamps cannot be  
28183 negative. If *pax* encounters a file with a negative timestamp in **copy** or **write** mode, it can reject  
28184 the file, substitute a non-negative timestamp, or generate a non-portable timestamp with a  
28185 leading ' - '. Even though some implementations can support finer file-time granularities than  
28186 seconds, the normative text requires support only for seconds since the Epoch because the  
28187 ISO POSIX-1 standard states them that way. The **ustar** format includes only *mtime*; the new  
28188 format adds *atime* and *ctime* for symmetry. The *atime* access time restored to the file system will  
28189 be affected by the **-p a** and **-p e** options. The *ctime* creation time (actually *inode* modification  
28190 time) is described with “appropriate privilege” so that it can be ignored when writing to the file  
28191 system. POSIX does not provide a portable means to change file creation time. Nothing is  
28192 intended to prevent a non-portable implementation of *pax* from restoring the value.

28193 The *gid*, *size*, and *uid* extended header records were included to allow expansion beyond the  
28194 sizes specified in the regular *tar* header. New file system architectures are emerging that will  
28195 exhaust the 12-digit size field. There are probably not many systems requiring more than 8 digits  
28196 for user and group IDs, but the extended header values were included for completeness,  
28197 allowing overrides for all of the decimal values in the *tar* header.

28198 The standard developers intended to describe the effective results of *pax* with regard to file  
28199 ownerships and permissions; implementations are not restricted in timing or sequencing the  
28200 restoration of such, provided the results are as specified.

28201 Much of the text describing the extended headers refers to use in “**write** or **copy** modes”. The  
28202 **copy** mode references are due to the normative text: “The effect of the copy shall be as if the  
28203 copied files were written to an archive file and then subsequently extracted ...”. There is  
28204 certainly no way to test whether *pax* is actually generating the extended headers in **copy** mode,  
28205 but the effects must be as if it had.

**28206 pax Archive Character Set Encoding/Decoding**

28207 There is a need to exchange archives of files between systems of different native codesets.  
28208 Filenames, group names, and user names must be preserved to the fullest extent possible when  
28209 an archive is read on the receiving platform. Translation of the contents of files is not within the  
28210 scope of the *pax* utility.

28211 There will also be the need to represent characters that are not available on the receiving  
28212 platform. These unsupported characters cannot be automatically folded to the local set of  
28213 characters due to the chance of collisions. This could result in overwriting previous extracted  
28214 files from the archive or pre-existing files on the system.

28215 For these reasons, the codeset used to represent characters within the extended header records of  
28216 the *pax* archive must be sufficiently rich to handle all commonly used character sets. The fields  
28217 requiring translation include, at a minimum, filenames, user names, group names, and link  
28218 pathnames. Implementations may wish to have localized extended keywords that use non-  
28219 portable characters.

28220 The standard developers considered the following options:

- 28221 • The archive creator specifies the well-defined name of the source codeset. The receiver must  
28222 then recognize the codeset name and perform the appropriate translations to the destination  
28223 codeset.
- 28224 • The archive creator includes within the archive the character mapping table for the source  
28225 codeset used to encode extended header records. The receiver must then read the character  
28226 mapping table and perform the appropriate translations to the destination codeset.
- 28227 • The archive creator translates the extended header records in the source codeset into a  
28228 canonical form. The receiver must then perform the appropriate translations to the  
28229 destination codeset.

28230 The approach that incorporates the name of the source codeset poses the problem of codeset  
28231 name registration, and makes the archive useless to *pax* archive decoders that do not recognize  
28232 that codeset.

28233 Because parts of an archive may be corrupted, the standard developers felt that including the  
28234 character map of the source codeset was too fragile. The loss of this one key component could  
28235 result in making the entire archive useless. (The difference between this and the global extended  
28236 header decision was that the latter has a workaround—duplicating extended header records on  
28237 unreliable media—but this would be too burdensome for large character set maps.)

28238 Both of the above approaches also put an undue burden on the *pax* archive receiver to handle the  
28239 cross-product of all source and destination codesets.

28240 To simplify the translation from the source codeset to the canonical form and from the canonical  
28241 form to the destination codeset, the standard developers decided that the internal representation  
28242 should be a stateless encoding. A stateless encoding is one where each codepoint has the same  
28243 meaning, without regard to the decoder being in a specific state. An example of a stateful  
28244 encoding would be the Japanese Shift-JIS; an example of a stateless encoding would be the  
28245 ISO/IEC 646:1991 standard (equivalent to 7-bit ASCII).

28246 For these reasons, the standard developers decided to adopt a canonical format for the  
28247 representation of file information strings. The obvious, well-endorsed candidate is the  
28248 ISO/IEC 10646-1:2000 standard (based in part on Unicode), which can be used to represent the  
28249 characters of virtually all standardized character sets. The standard developers initially agreed  
28250 upon using UCS2 (16-bit Unicode) as the internal representation. This repertoire of characters  
28251 provides a sufficiently rich set to represent all commonly-used codesets.

28252 However, the standard developers found that the 16-bit Unicode representation had some  
 28253 problems. It forced the issue of standardizing byte ordering. The 2-byte length of each character  
 28254 made the extended header records twice as long for the case of strings coded entirely from  
 28255 historical 7-bit ASCII. For these reasons, the standard developers chose the UTF-8 defined in the  
 28256 ISO/IEC 10646-1:2000 standard. This multi-byte representation encodes UCS2 or UCS4  
 28257 characters reliably and deterministically, eliminating the need for a canonical byte ordering. In  
 28258 addition, NUL octets and other characters possibly confusing to POSIX file systems do not  
 28259 appear, except to represent themselves. It was realized that certain national codesets take up  
 28260 more space after the encoding, due to their placement within the UCS range; it was felt that the  
 28261 usefulness of the encoding of the names outweighs the disadvantage of size increase for file,  
 28262 user, and group names.

28263 The encoding of UTF-8 is as follows:

| 28264 | UCS4 Hex Encoding | UTF-8 Binary Encoding                                 |
|-------|-------------------|-------------------------------------------------------|
| 28265 | 00000000-0000007F | 0xxxxxxx                                              |
| 28266 | 00000080-000007FF | 110xxxxx 10xxxxxx                                     |
| 28267 | 00000800-0000FFFF | 1110xxxx 10xxxxxx 10xxxxxx                            |
| 28268 | 00010000-001FFFFF | 11110xxx 10xxxxxx 10xxxxxx 10xxxxxx                   |
| 28269 | 00200000-03FFFFFF | 111110xx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx          |
| 28270 | 04000000-7FFFFFFF | 1111110x 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx |

28271 where each 'x' represents a bit value from the character being translated.

#### 28272 **ustar Interchange Format**

28273 The description of the **ustar** format reflects numerous enhancements over pre-1988 versions of  
 28274 the historical *tar* utility. The goal of these changes was not only to provide the functional  
 28275 enhancements desired, but also to retain compatibility between new and old versions. This  
 28276 compatibility has been retained. Archives written using the old archive format are compatible  
 28277 with the new format.

28278 Implementors should be aware that the previous file format did not include a mechanism to  
 28279 archive directory type files. For this reason, the convention of using a filename ending with slash  
 28280 was adopted to specify a directory on the archive.

28281 The total size of the *name* and *prefix* fields have been set to meet the minimum requirements for  
 28282 {PATH\_MAX}. If a pathname will fit within the *name* field, it is recommended that the pathname  
 28283 be stored there without the use of the *prefix* field. Although the name field is known to be too  
 28284 small to contain {PATH\_MAX} characters, the value was not changed in this version of the  
 28285 archive file format to retain backward compatibility, and instead the prefix was introduced.  
 28286 Also, because of the earlier version of the format, there is no way to remove the restriction on the  
 28287 *linkname* field being limited in size to just that of the *name* field.

28288 The *size* field is required to be meaningful in all implementation extensions, although it could be  
 28289 zero. This is required so that the data blocks can always be properly counted.

28290 It is suggested that if device special files need to be represented that cannot be represented in the  
 28291 standard format that one of the extension types (A-Z) be used, and that the additional  
 28292 information for the special file be represented as data and be reflected in the *size* field.

28293 Attempting to restore a special file type, where it is converted to ordinary data and conflicts  
 28294 with an existing filename, need not be specially detected by the utility. If run as an ordinary user,  
 28295 *pax* should not be able to overwrite the entries in, for example, */dev* in any case (whether the file  
 28296 is converted to another type or not). If run as a privileged user, it should be able to do so, and it  
 28297 would be considered a bug if it did not. The same is true of ordinary data files and similarly

28298 named special files; it is impossible to anticipate the needs of the user (who could really intend  
 28299 to overwrite the file), so the behavior should be predictable (and thus regular) and rely on the  
 28300 protection system as required.

28301 The value 7 in the *typeflag* field is intended to define how contiguous files can be stored in a  
 28302 **ustar** archive. IEEE Std 1003.1-200x does not require the contiguous file extension, but does  
 28303 define a standard way of archiving such files so that all conforming systems can interpret these  
 28304 file types in a meaningful and consistent manner. On a system that does not support extended  
 28305 file types, the *pax* utility should do the best it can with the file and go on to the next.

28306 The file protection modes are those conventionally used by the *ls* utility. This is extended  
 28307 beyond the usage in the ISO POSIX-2 standard to support the “shared text” or “sticky” bit. It is  
 28308 intended that the conformance document should not document anything beyond the existence  
 28309 of and support of such a mode. Further extensions are expected to these bits, particularly with  
 28310 overloading the set-user-ID and set-group-ID flags.

### 28311 **cpio Interchange Format**

28312 The reference to appropriate privilege in the *cpio* format refers to an error on standard output;  
 28313 the **ustar** format does not make comparable statements.

28314 The model for this format was the historical System V *cpio-c* data interchange format. This  
 28315 model documents the portable version of the *cpio* format and not the binary version. It has the  
 28316 flexibility to transfer data of any type described within IEEE Std 1003.1-200x, yet is extensible to  
 28317 transfer data types specific to extensions beyond IEEE Std 1003.1-200x (for example, contiguous  
 28318 files). Because it describes existing practice, there is no question of maintaining upward  
 28319 compatibility.

### 28320 **cpio Header**

28321 There has been some concern that the size of the *c\_ino* field of the header is too small to handle  
 28322 those systems that have very large *inode* numbers. However, the *c\_ino* field in the header is used  
 28323 strictly as a hard-link resolution mechanism for archives. It is not necessarily the same value as  
 28324 the *inode* number of the file in the location from which that file is extracted.

28325 The name *c\_magic* is based on historical usage.

### 28326 **cpio Filename**

28327 For most historical implementations of the *cpio* utility, {PATH\_MAX} octets can be used to  
 28328 describe the pathname without the addition of any other header fields (the NUL character  
 28329 would be included in this count). {PATH\_MAX} is the minimum value for pathname size,  
 28330 documented as 256 bytes. However, an implementation may use *c\_namesize* to determine the  
 28331 exact length of the pathname. With the current description of the **<cpio.h>** header, this  
 28332 pathname size can be as large as a number that is described in six octal digits.

28333 Two values are documented under the *c\_mode* field values to provide for extensibility for known  
 28334 file types:

28335 **0110 000** Reserved for contiguous files. The implementation may treat the rest of the  
 28336 information for this archive like a regular file. If this file type is undefined, the  
 28337 implementation may create the file as a regular file.

28338 This provides for extensibility of the *cpio* format while allowing for the ability to read old  
 28339 archives. Files of an unknown type may be read as “regular files” on some implementations. On  
 28340 a system that does not support extended file types, the *pax* utility should do the best it can with  
 28341 the file and go on to the next.

28342 **FUTURE DIRECTIONS**

28343 None.

28344 **SEE ALSO**28345 *cp*, *ed*, *getopts*, *printf*, the Base Definitions volume of IEEE Std 1003.1-200x, <**cpio.h**>, the System  
28346 Interfaces volume of IEEE Std 1003.1-200x, *chown()*, *creat()*, *mkdir()*, *stat()*, *write()*28347 **CHANGE HISTORY**

28348 First released in Issue 4.

28349 **Issue 5**28350 A note is added to the APPLICATION USAGE indicating that the *cpio* and *tar* formats can only  
28351 support files up to 8 gigabytes in size.28352 **Issue 6**28353 The *pax* utility is aligned with the IEEE P1003.2b draft standard:

- 28354 • Support has been added for symbolic links in the options and interchange formats.
- 28355 • A new format has been devised, based on extensions to *ustar*.
- 28356 • References to the “extended” *tar* and *cpio* formats derived from the POSIX.1-1990 standard
- 28357 have been changed to remove the “extended” adjective because this could cause confusion
- 28358 with the extended *tar* header added in this revision. (All references to *tar* are actually to
- 28359 **ustar**).

28360 IEEE PASC Interpretation 1003.2 #168 is applied clarifying that *mkdir()* and *mkfifo()* calls can  
28361 ignore an [EEXIST] error when extracting an archive.28362 The *TZ* entry is added to the ENVIRONMENT VARIABLES section.28363 IEEE PASC Interpretation 1003.2 #180 is applied, clarifying how extracted files are created when  
28364 in **read** mode.28365 IEEE PASC Interpretation 1003.2 #181 is applied, clarifying the description of the **-t** option. |

28366 IEEE PASC Interpretation 1003.2 #195 is applied. |

28367 IEEE PASC Interpretation 1003.2 #206 is applied, clarifying the handling of links for the **-H**, **-L**, |  
28368 and **-l** options. |

## 28369 NAME

28370 pr — print files

## 28371 SYNOPSIS

```
28372 pr [+page][-column][-adFmrt][-e[char][gap]][-h header][-i[char][gap]]
28373 xSI [-l lines][-n[char][width]][-o offset][-s[char]][-w width][-fp]
28374 [file...]
```

## 28375 DESCRIPTION

28376 The *pr* utility is a printing and pagination filter. If multiple input files are specified, each shall be  
 28377 read, formatted, and written to standard output. By default, the input shall be separated into 66-  
 28378 line pages, each with:

- 28379 • A 5-line header that includes the page number, date, time, and the pathname of the file
- 28380 • A 5-line trailer consisting of blank lines

28381 If standard output is associated with a terminal, diagnostic messages shall be deferred until the  
 28382 *pr* utility has completed processing.

28383 When options specifying multi-column output are specified, output text columns shall be of  
 28384 equal width; input lines that do not fit into a text column shall be truncated. By default, text  
 28385 columns shall be separated with at least one <blank>.

## 28386 OPTIONS

28387 The *pr* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2,  
 28388 Utility Syntax Guidelines, except that: the *page* option has a '+' delimiter; *page* and *column* can  
 28389 be multi-digit numbers; some of the option-arguments are optional; and some of the option-  
 28390 arguments cannot be specified as separate arguments from the preceding option letter. In  
 28391 particular, the *-s* option does not allow the option letter to be separated from its argument, and  
 28392 the options *-e*, *-i*, and *-n* require that both arguments, if present, not be separated from the  
 28393 option letter.

28394 The following options shall be supported. In the following option descriptions, *column*, *lines*,  
 28395 *offset*, *page*, and *width* are positive decimal integers; *gap* is a non-negative decimal integer.

- 28396 *+page* Begin output at page number *page* of the formatted input.
- 28397 *-column* Produce multi-column output that is arranged in *column* columns (the default shall  
 28398 be 1) and is written down each column in the order in which the text is received  
 28399 from the input file. This option should not be used with *-m*. The options *-e* and *-i*  
 28400 shall be assumed for multiple text-column output. Whether or not text columns  
 28401 are produced with identical vertical lengths is unspecified, but a text column shall  
 28402 never exceed the length of the page (see the *-l* option). When used with *-t*, use the  
 28403 minimum number of lines to write the output.
- 28404 *-a* Modify the effect of the *-column* option so that the columns are filled across the  
 28405 page in a round-robin order (for example, when *column* is 2, the first input line  
 28406 heads column 1, the second heads column 2, the third is the second line in column  
 28407 1, and so on).
- 28408 *-d* Produce output that is double-spaced; append an extra <newline> following every  
 28409 <newline> found in the input.
- 28410 *-e*[char][gap]  
 28411 Expand each input <tab> to the next greater column position specified by the  
 28412 formula  $n*gap+1$ , where *n* is an integer > 0. If *gap* is zero or is omitted, it shall  
 28413 default to 8. All <tab>s in the input shall be expanded into the appropriate number  
 28414 of <space>s. If any non-digit character, *char*, is specified, it shall be used as the

|           |                        |                                                                                                                                                                                                                                                                                                                                                                                                 |
|-----------|------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 28415     |                        | input <tab>.                                                                                                                                                                                                                                                                                                                                                                                    |
| 28416 XSI | <b>-f</b>              | Use a <form-feed> for new pages, instead of the default behavior that uses a sequence of <newline>s. Pause before beginning the first page if the standard output is associated with a terminal.                                                                                                                                                                                                |
| 28417     |                        |                                                                                                                                                                                                                                                                                                                                                                                                 |
| 28418     |                        |                                                                                                                                                                                                                                                                                                                                                                                                 |
| 28419     | <b>-F</b>              | Use a <form-feed> for new pages, instead of the default behavior that uses a sequence of <newline>s.                                                                                                                                                                                                                                                                                            |
| 28420     |                        |                                                                                                                                                                                                                                                                                                                                                                                                 |
| 28421     | <b>-h header</b>       | Use the string <i>header</i> to replace the contents of the <i>file</i> operand in the page header.                                                                                                                                                                                                                                                                                             |
| 28422     | <b>-i[char][gap]</b>   |                                                                                                                                                                                                                                                                                                                                                                                                 |
| 28423     |                        | In output, replace multiple <space>s with <tab>s wherever two or more adjacent <space>s reach column positions <i>gap</i> +1, 2* <i>gap</i> +1, 3* <i>gap</i> +1, and so on. If <i>gap</i> is zero or is omitted, default tab settings at every eighth column position shall be assumed. If any non-digit character, <i>char</i> , is specified, it shall be used as the output <tab>.          |
| 28424     |                        |                                                                                                                                                                                                                                                                                                                                                                                                 |
| 28425     |                        |                                                                                                                                                                                                                                                                                                                                                                                                 |
| 28426     |                        |                                                                                                                                                                                                                                                                                                                                                                                                 |
| 28427     |                        |                                                                                                                                                                                                                                                                                                                                                                                                 |
| 28428     | <b>-l lines</b>        | Override the 66-line default and reset the page length to <i>lines</i> . If <i>lines</i> is not greater than the sum of both the header and trailer depths (in lines), the <i>pr</i> utility shall suppress both the header and trailer, as if the <b>-t</b> option were in effect.                                                                                                             |
| 28429     |                        |                                                                                                                                                                                                                                                                                                                                                                                                 |
| 28430     |                        |                                                                                                                                                                                                                                                                                                                                                                                                 |
| 28431     | <b>-m</b>              | Merge files. Standard output shall be formatted so the <i>pr</i> utility writes one line from each file specified by a <i>file</i> operand, side by side into text columns of equal fixed widths, in terms of the number of column positions. Implementations shall support merging of at least nine <i>file</i> operands.                                                                      |
| 28432     |                        |                                                                                                                                                                                                                                                                                                                                                                                                 |
| 28433     |                        |                                                                                                                                                                                                                                                                                                                                                                                                 |
| 28434     |                        |                                                                                                                                                                                                                                                                                                                                                                                                 |
| 28435     | <b>-n[char][width]</b> |                                                                                                                                                                                                                                                                                                                                                                                                 |
| 28436     |                        | Provide <i>width</i> -digit line numbering (default for <i>width</i> shall be 5). The number shall occupy the first <i>width</i> column positions of each text column of default output or each line of <b>-m</b> output. If <i>char</i> (any non-digit character) is given, it shall be appended to the line number to separate it from whatever follows (default for <i>char</i> is a <tab>). |
| 28437     |                        |                                                                                                                                                                                                                                                                                                                                                                                                 |
| 28438     |                        |                                                                                                                                                                                                                                                                                                                                                                                                 |
| 28439     |                        |                                                                                                                                                                                                                                                                                                                                                                                                 |
| 28440     |                        |                                                                                                                                                                                                                                                                                                                                                                                                 |
| 28441     | <b>-o offset</b>       | Each line of output shall be preceded by offset <space>s. If the <b>-o</b> option is not specified, the default offset shall be zero. The space taken is in addition to the output line width (see the <b>-w</b> option below).                                                                                                                                                                 |
| 28442     |                        |                                                                                                                                                                                                                                                                                                                                                                                                 |
| 28443     |                        |                                                                                                                                                                                                                                                                                                                                                                                                 |
| 28444     | <b>-p</b>              | Pause before beginning each page if the standard output is directed to a terminal ( <i>pr</i> shall write an <alert> to standard error and wait for a <carriage-return> to be read on <b>/dev/tty</b> ).                                                                                                                                                                                        |
| 28445     |                        |                                                                                                                                                                                                                                                                                                                                                                                                 |
| 28446     |                        |                                                                                                                                                                                                                                                                                                                                                                                                 |
| 28447     | <b>-r</b>              | Write no diagnostic reports on failure to open files.                                                                                                                                                                                                                                                                                                                                           |
| 28448     | <b>-s[char]</b>        | Separate text columns by the single character <i>char</i> instead of by the appropriate number of <space>s (default for <i>char</i> shall be <tab>).                                                                                                                                                                                                                                            |
| 28449     |                        |                                                                                                                                                                                                                                                                                                                                                                                                 |
| 28450     | <b>-t</b>              | Write neither the five-line identifying header nor the five-line trailer usually supplied for each page. Quit writing after the last line of each file without spacing to the end of the page.                                                                                                                                                                                                  |
| 28451     |                        |                                                                                                                                                                                                                                                                                                                                                                                                 |
| 28452     |                        |                                                                                                                                                                                                                                                                                                                                                                                                 |
| 28453     | <b>-w width</b>        | Set the width of the line to <i>width</i> column positions for multiple text-column output only. If the <b>-w</b> option is not specified and the <b>-s</b> option is not specified, the default width shall be 72. If the <b>-w</b> option is not specified and the <b>-s</b> option is specified, the default width shall be 512.                                                             |
| 28454     |                        |                                                                                                                                                                                                                                                                                                                                                                                                 |
| 28455     |                        |                                                                                                                                                                                                                                                                                                                                                                                                 |
| 28456     |                        |                                                                                                                                                                                                                                                                                                                                                                                                 |
| 28457     |                        | For single column output, input lines shall not be truncated.                                                                                                                                                                                                                                                                                                                                   |

28458 **OPERANDS**

28459 The following operand shall be supported:

28460 *file* A pathname of a file to be written. If no *file* operands are specified, or if a *file*  
 28461 operand is '-', the standard input shall be used.

28462 **STDIN**

28463 The standard input shall be used only if no *file* operands are specified, or if a *file* operand is '-'.  
 28464 See the INPUT FILES section.

28465 **INPUT FILES**

28466 The input files shall be text files.

28467 The file `/dev/tty` shall be used to read responses required by the `-p` option. |

28468 **ENVIRONMENT VARIABLES**

28469 The following environment variables shall affect the execution of *pr*:

28470 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 28471 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
 28472 Internationalization Variables for the precedence of internationalization variables  
 28473 used to determine the values of locale categories.)

28474 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 28475 internationalization variables.

28476 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 28477 characters (for example, single-byte as opposed to multi-byte characters in  
 28478 arguments and input files) and which characters are defined as printable (character  
 28479 class **print**). Non-printable characters are still written to standard output, but are  
 28480 not counted for the purpose for column-width and line-length calculations.

28481 *LC\_MESSAGES*

28482 Determine the locale that should be used to affect the format and contents of  
 28483 diagnostic messages written to standard error.

28484 *LC\_TIME* Determine the format of the date and time for use in writing header lines.

28485 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

28486 *TZ* Determine the timezone used to calculate date and time strings written in header  
 28487 lines. If *TZ* is unset or null, an unspecified default timezone shall be used.

28488 **ASYNCHRONOUS EVENTS**

28489 If *pr* receives an interrupt while writing to a terminal, it shall flush all accumulated error  
 28490 messages to the screen before terminating.

28491 **STDOUT**

28492 The *pr* utility output shall be a paginated version of the original file (or files). This pagination  
 28493 shall be accomplished using either `<form-feed>`s or a sequence of `<newline>`s, as controlled by  
 28494 XSI the `-F` or `-f` option. Page headers shall be generated unless the `-t` option is specified. The page  
 28495 headers shall be of the form:

28496 "`\n\n%s %s Page %d\n\n\n`", `<output of date>`, `<file>`, `<page number>`

28497 In the POSIX locale, the `<output of date>` field, representing the date and time of last modification  
 28498 of the input file (or the current date and time if the input file is standard input), shall be  
 28499 equivalent to the output of the following command as it would appear if executed at the given  
 28500 time:

28501 date "+%b %e %H:%M %Y"

28502 without the trailing <newline>, if the page being written is from standard input. If the page  
 28503 being written is not from standard input, in the POSIX locale, the same format shall be used, but  
 28504 the time used shall be the modification time of the file corresponding to *file* instead of the current  
 28505 time. When the *LC\_TIME* locale category is not set to the POSIX locale, a different format and  
 28506 order of presentation of this field may be used.

28507 If the standard input is used instead of a *file* operand, the <*file*> field shall be replaced by a null  
 28508 string.

28509 If the **-h** option is specified, the <*file*> field shall be replaced by the *header* argument.

#### 28510 **STDERR**

28511 The standard error shall be used for diagnostic messages and for alerting the terminal when **-p** |  
 28512 is specified.

#### 28513 **OUTPUT FILES**

28514 None.

#### 28515 **EXTENDED DESCRIPTION**

28516 None.

#### 28517 **EXIT STATUS**

28518 The following exit values shall be returned:

28519 0 Successful completion.

28520 >0 An error occurred.

#### 28521 **CONSEQUENCES OF ERRORS**

28522 Default.

#### 28523 **APPLICATION USAGE**

28524 None.

#### 28525 **EXAMPLES**

28526 1. Print a numbered list of all files in the current directory:

28527 `ls -a | pr -n -h "Files in $(pwd)."`

28528 2. Print **file1** and **file2** as a double-spaced, three-column listing headed by “file list”:

28529 `pr -3d -h "file list" file1 file2`

28530 3. Write **file1** on **file2**, expanding tabs to columns 10, 19, 28, ...:

28531 `pr -e9 -t <file1 >file2`

#### 28532 **RATIONALE**

28533 This utility is one of those that does not follow the Utility Syntax Guidelines because of its  
 28534 historical origins. The standard developers could have added new options that obeyed the  
 28535 guidelines (and marked the old options *obsolescent*) or devised an entirely new utility; there are  
 28536 examples of both actions in this volume of IEEE Std 1003.1-200x. Because of its widespread use  
 28537 by historical applications, the standard developers decided to exempt this version of *pr* from  
 28538 many of the guidelines.

28539 Implementations are required to accept option-arguments to the **-h**, **-l**, **-o**, and **-w** options  
 28540 whether presented as part of the same argument or as a separate argument to *pr*, as suggested by  
 28541 the Utility Syntax Guidelines. The **-n** and **-s** options, however, are specified as in historical  
 28542 practice because they are frequently specified without their optional arguments. If a <blank>

- 28543 were allowed before the option-argument in these cases, a *file* operand could mistakenly be  
28544 interpreted as an option-argument in historical applications.
- 28545 The text about the minimum number of lines in multi-column output was included to ensure  
28546 that a best effort is made in balancing the length of the columns. There are known historical  
28547 implementations in which, for example, 60-line files are listed by *pr -2* as one column of 56 lines  
28548 and a second of 4. Although this is not a problem when a full page with headers and trailers is  
28549 produced, it would be relatively useless when used with *-t*.
- 28550 Historical implementations of the *pr* utility have differed in the action taken for the *-f* option.  
28551 BSD uses it as described here for the *-F* option; System V uses it to change trailing *<newline>s*  
28552 on each page to a *<form-feed>* and, if standard output is a TTY device, sends an *<alert>* to  
28553 standard error and reads a line from */dev/tty* before the first page. There were strong arguments  
28554 from both sides of this issue concerning historical practice and as a result the *-F* option was  
28555 added. XSI-conformant systems support the System V historical actions for the *-f* option.
- 28556 The *<output of date>* field in the *-I* format is specified only for the POSIX locale. As noted, the  
28557 format can be different in other locales. No mechanism for defining this is present in this volume  
28558 of IEEE Std 1003.1-200x, as the appropriate vehicle is a message catalog; that is, the format  
28559 should be specified as a “message”.
- 28560 **FUTURE DIRECTIONS**
- 28561 None.
- 28562 **SEE ALSO**
- 28563 *expand, lp*
- 28564 **CHANGE HISTORY**
- 28565 First released in Issue 2.
- 28566 **Issue 6**
- 28567 The following new requirements on POSIX implementations derive from alignment with the  
28568 Single UNIX Specification:
- 28569
  - The *-p* option is added.
- 28570 The normative text is reworded to avoid use of the term “must” for application requirements.

28571 **NAME**

28572           printf — write formatted output

28573 **SYNOPSIS**

28574           printf *format*[*argument...*]

28575 **DESCRIPTION**

28576           The *printf* utility shall write formatted operands to the standard output. The *argument* operands  
28577           shall be formatted under control of the *format* operand.

28578 **OPTIONS**

28579           None.

28580 **OPERANDS**

28581           The following operands shall be supported:

28582           *format*        A string describing the format to use to write the remaining operands. See the  
28583           EXTENDED DESCRIPTION section.

28584           *argument*    The strings to be written to standard output, under the control of *format*. See the  
28585           EXTENDED DESCRIPTION section.

28586 **STDIN**

28587           Not used.

28588 **INPUT FILES**

28589           None.

28590 **ENVIRONMENT VARIABLES**

28591           The following environment variables shall affect the execution of *printf*:

28592           *LANG*        Provide a default value for the internationalization variables that are unset or null.  
28593                            (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
28594                            Internationalization Variables for the precedence of internationalization variables  
28595                            used to determine the values of locale categories.)

28596           *LC\_ALL*     If set to a non-empty string value, override the values of all the other  
28597           internationalization variables.

28598           *LC\_CTYPE*   Determine the locale for the interpretation of sequences of bytes of text data as  
28599           characters (for example, single-byte as opposed to multi-byte characters in  
28600           arguments).

28601           *LC\_MESSAGES*

28602                            Determine the locale that should be used to affect the format and contents of  
28603           diagnostic messages written to standard error.

28604           *LC\_NUMERIC*

28605                            Determine the locale for numeric formatting. It shall affect the format of numbers  
28606           written using the e, E, f, g, and G conversion specifier characters (if supported).

28607 XSI           *NLSPATH*   Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

28608 **ASYNCHRONOUS EVENTS**

28609           Default.

28610 **STDOUT**

28611           See the EXTENDED DESCRIPTION section.

28612 **STDERR**

28613 The standard error shall be used only for diagnostic messages.

28614 **OUTPUT FILES**

28615 None.

28616 **EXTENDED DESCRIPTION**

28617 The *format* operand shall be used as the *format* string described in the Base Definitions volume of  
28618 IEEE Std 1003.1-200x, Chapter 5, File Format Notation with the following exceptions:

- 28619 1. A <space> in the format string, in any context other than a flag of a conversion  
28620 specification, shall be treated as an ordinary character that is copied to the output.
- 28621 2. A 'Δ' character in the format string shall be treated as a 'Δ' character, not as a <space>.
- 28622 3. In addition to the escape sequences shown in the Base Definitions volume of  
28623 IEEE Std 1003.1-200x, Chapter 5, File Format Notation ('\\', '\a', '\b', '\f', '\n',  
28624 '\r', '\t', '\v'), "\ddd", where *ddd* is a one, two, or three-digit octal number, shall be  
28625 written as a byte with the numeric value specified by the octal number.
- 28626 4. The implementation shall not precede or follow output from the *d* or *u* conversion  
28627 specifiers with <blank>s not specified by the *format* operand.
- 28628 5. The implementation shall not precede output from the *o* conversion specifier with zeros  
28629 not specified by the *format* operand.
- 28630 6. The *e*, *E*, *f*, *g*, and *G* conversion specifiers need not be supported.
- 28631 7. An additional conversion specifier character, *b*, shall be supported as follows. The  
28632 argument shall be taken to be a string that may contain backslash-escape sequences. The  
28633 following backslash-escape sequences shall be supported:
  - 28634 — The escape sequences listed in the Base Definitions volume of IEEE Std 1003.1-200x,  
28635 Chapter 5, File Format Notation ('\\', '\a', '\b', '\f', '\n', '\r', '\t', '\v'),  
28636 which shall be converted to the characters they represent
  - 28637 — "\0ddd", where *ddd* is a zero, one, two, or three-digit octal number that shall be  
28638 converted to a byte with the numeric value specified by the octal number
  - 28639 — '\c', which shall not be written and shall cause *printf* to ignore any remaining  
28640 characters in the string operand containing it, any remaining string operands, and any  
28641 additional characters in the *format* operand
- 28642 The interpretation of a backslash followed by any other sequence of characters is  
28643 unspecified.
- 28644 Bytes from the converted string shall be written until the end of the string or the number of  
28645 bytes indicated by the precision specification is reached. If the precision is omitted, it shall  
28646 be taken to be infinite, so all bytes up to the end of the converted string shall be written.
- 28647 8. For each conversion specification that consumes an argument, the next argument operand  
28648 shall be evaluated and converted to the appropriate type for the conversion as specified  
28649 below.
- 28650 9. The *format* operand shall be reused as often as necessary to satisfy the argument operands.  
28651 Any extra *c* or *s* conversion specifiers shall be evaluated as if a null string argument were  
28652 supplied; other extra conversion specifications shall be evaluated as if a zero argument  
28653 were supplied. If the *format* operand contains no conversion specifications and *argument*  
28654 operands are present, the results are unspecified.

28655 10. If a character sequence in the *format* operand begins with a '%' character, but does not  
28656 form a valid conversion specification, the behavior is unspecified.

28657 The *argument* operands shall be treated as strings if the corresponding conversion specifier is b,  
28658 c, or s; otherwise, it shall be evaluated as a C constant, as described by the ISO C standard, with  
28659 the following extensions:

- 28660 • A leading plus or minus sign shall be allowed.
- 28661 • If the leading character is a single-quote or double-quote, the value shall be the numeric  
28662 value in the underlying codeset of the character following the single-quote or double-quote.

28663 If an argument operand cannot be completely converted into an internal value appropriate to  
28664 the corresponding conversion specification, a diagnostic message shall be written to standard  
28665 error and the utility shall not exit with a zero exit status, but shall continue processing any  
28666 remaining operands and shall write the value accumulated at the time the error was detected to  
28667 standard output.

28668 It is not considered an error if an argument operand is not completely used for a c or s  
28669 conversion or if a string operand's first or second character is used to get the numeric value of a  
28670 character.

#### 28671 EXIT STATUS

28672 The following exit values shall be returned:

28673 0 Successful completion.

28674 >0 An error occurred.

#### 28675 CONSEQUENCES OF ERRORS

28676 Default.

#### 28677 APPLICATION USAGE

28678 The floating-point formatting conversion specifications of *printf()* are not required because all  
28679 arithmetic in the shell is integer arithmetic. The *awk* utility performs floating-point calculations  
28680 and provides its own **printf** function. The *bc* utility can perform arbitrary-precision floating-  
28681 point arithmetic, but does not provide extensive formatting capabilities. (This *printf* utility  
28682 cannot really be used to format *bc* output; it does not support arbitrary precision.)  
28683 Implementations are encouraged to support the floating-point conversions as an extension.

28684 Note that this *printf* utility, like the *printf()* function defined in the System Interfaces volume of  
28685 IEEE Std 1003.1-200x on which it is based, makes no special provision for dealing with multi-  
28686 byte characters when using the %c conversion specification or when a precision is specified in a  
28687 %b or %s conversion specification. Applications should be extremely cautious using either of  
28688 these features when there are multi-byte characters in the character set.

28689 No provision is made in this volume of IEEE Std 1003.1-200x which allows field widths and  
28690 precisions to be specified as '\*' since the '\*' can be replaced directly in the *format* operand  
28691 using shell variable substitution. Implementations can also provide this feature as an extension  
28692 if they so choose.

28693 Hexadecimal character constants as defined in the ISO C standard are not recognized in the  
28694 *format* operand because there is no consistent way to detect the end of the constant. Octal  
28695 character constants are limited to, at most, three octal digits, but hexadecimal character  
28696 constants are only terminated by a non-hex-digit character. In the ISO C standard, the "##"  
28697 concatenation operator can be used to terminate a constant and follow it with a hexadecimal  
28698 character to be written. In the shell, concatenation occurs before the *printf* utility has a chance to  
28699 parse the end of the hexadecimal constant.

28700 The %b conversion specification is not part of the ISO C standard; it has been added here as a  
 28701 portable way to process backslash escapes expanded in string operands as provided by the *echo*  
 28702 utility. See also the APPLICATION USAGE section of *echo* (on page 2534) for ways to use *printf*  
 28703 as a replacement for all of the traditional versions of the *echo* utility.

28704 If an argument cannot be parsed correctly for the corresponding conversion specification, the  
 28705 *printf* utility is required to report an error. Thus, overflow and extraneous characters at the end  
 28706 of an argument being used for a numeric conversion shall be reported as errors.

#### 28707 EXAMPLES

28708 To alert the user and then print and read a series of prompts:

```
28709 printf "\aPlease fill in the following: \nName: "
28710 read name
28711 printf "Phone number: "
28712 read phone
```

28713 To read out a list of right and wrong answers from a file, calculate the percentage correctly, and  
 28714 print them out. The numbers are right-justified and separated by a single <tab>. The percentage  
 28715 is written to one decimal place of accuracy:

```
28716 while read right wrong ; do
28717     percent=$(echo "scale=1;($right*100)/($right+$wrong)" | bc)
28718     printf "%2d right\t%2d wrong\t(%%)\n" \
28719         $right $wrong $percent
28720 done < database_file
```

28721 The command:

```
28722 printf "%5d%4d\n" 1 21 321 4321 54321
```

28723 produces:

```
28724     1  21
28725     3214321
28726 54321   0
```

28727 Note that the *format* operand is used three times to print all of the given strings and that a '0'  
 28728 was supplied by *printf* to satisfy the last %4d conversion specification.

28729 The *printf* utility is required to notify the user when conversion errors are detected while  
 28730 producing numeric output; thus, the following results would be expected on an implementation  
 28731 with 32-bit twos-complement integers when %d is specified as the *format* operand:

| Argument    | Standard Output | Diagnostic Output                         |
|-------------|-----------------|-------------------------------------------|
| 5a          | 5               | printf: "5a" not completely converted     |
| 9999999999  | 2147483647      | printf: "9999999999" arithmetic overflow  |
| -9999999999 | -2147483648     | printf: "-9999999999" arithmetic overflow |
| ABC         | 0               | printf: "ABC" expected numeric value      |

28738 The diagnostic message format is not specified, but these examples convey the type of  
 28739 information that should be reported. Note that the value shown on standard output is what  
 28740 would be expected as the return value from the *strtol()* function as defined in the System  
 28741 Interfaces volume of IEEE Std 1003.1-200x. A similar correspondence exists between %u and  
 28742 *strtoul()* and %e, %f, and %g (if the implementation supports floating-point conversions) and  
 28743 *strtod()*.

- 28744 In a locale using the ISO/IEC 646:1991 standard as the underlying codeset, the command:
- 28745 `printf "%d\n" 3 +3 -3 \'3 \"+3 "'-3"`
- 28746 produces:
- 28747 3 Numeric value of constant 3
- 28748 3 Numeric value of constant 3
- 28749 -3 Numeric value of constant -3
- 28750 51 Numeric value of the character '3' in the ISO/IEC 646:1991 standard codeset
- 28751 43 Numeric value of the character '+' in the ISO/IEC 646:1991 standard codeset
- 28752 45 Numeric value of the character '-' in the ISO/IEC 646:1991 standard codeset
- 28753 Note that in a locale with multi-byte characters, the value of a character is intended to be the
- 28754 value of the equivalent of the `wchar_t` representation of the character as described in the System
- 28755 Interfaces volume of IEEE Std 1003.1-200x.
- 28756 **RATIONALE**
- 28757 The *printf* utility was added to provide functionality that has historically been provided by *echo*.
- 28758 However, due to irreconcilable differences in the various versions of *echo* extant, the version has
- 28759 few special features, leaving those to this new *printf* utility, which is based on one in the Ninth
- 28760 Edition system.
- 28761 The EXTENDED DESCRIPTION section almost exactly matches the *printf()* function in the
- 28762 ISO C standard, although it is described in terms of the file format notation in the Base
- 28763 Definitions volume of IEEE Std 1003.1-200x, Chapter 5, File Format Notation.
- 28764 **FUTURE DIRECTIONS**
- 28765 None.
- 28766 **SEE ALSO**
- 28767 *awk*, *bc*, *echo*, the System Interfaces volume of IEEE Std 1003.1-200x, *printf()*
- 28768 **CHANGE HISTORY**
- 28769 First released in Issue 4.

## 28770 NAME

28771 prs — print an SCCS file (**DEVELOPMENT**)

## 28772 SYNOPSIS

28773 xSI prs [-a][-d *dataspec*][-r[*SID*]] *file...*28774 xSI prs [-e|-l] -c *cutoff* [-d *dataspec*] *file...*28775 xSI prs [-e|-l] -r[*SID*][-d *dataspec*]*file...*

28776

## 28777 DESCRIPTION

28778 The *prs* utility shall write to standard output parts or all of an SCCS file in a user-supplied  
28779 format.

## 28780 OPTIONS

28781 The *prs* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section  
28782 12.2, Utility Syntax Guidelines, except that the *-r* option has an optional option-argument. This  
28783 optional option-argument cannot be presented as a separate argument. The following options  
28784 shall be supported:28785 *-d dataspec* Specify the output data specification. The *dataspec* shall be a string consisting of  
28786 SCCS file *data keywords* (see **Data Keywords** (on page 2946)) interspersed with  
28787 optional user-supplied text.28788 *-r[SID]* Specify the SCCS identification string (SID) of a delta for which information is  
28789 desired. If no *SID* option-argument is specified, the SID of the most recently  
28790 created delta shall be assumed.28791 *-e* Request information for all deltas created earlier than and including the delta  
28792 designated via the *-r* option or the date-time given by the *-c* option.28793 *-l* Request information for all deltas created later than and including the delta  
28794 designated via the *-r* option or the date-time given by the *-c* option.28795 *-c cutoff* Indicate the *cutoff* date-time, in the form:28796 *YY[MM[DD[HH[MM[SS]]]]]*28797 For the *YY* component, values in the range [69,99] shall refer to years 1969 to 1999  
28798 inclusive, and values in the range [00,68] shall refer to years 2000 to 2068 inclusive.28799 **Note:** It is expected that in a future version of IEEE Std 1003.1-200x the default  
28800 century inferred from a 2-digit year will change. (This would apply to all  
28801 commands accepting a 2-digit year as input.)28802 No changes (deltas) to the SCCS file that were created after the specified *cutoff*  
28803 date-time shall be included in the output. Units omitted from the date-time default  
28804 to their maximum possible values; for example, *-c 7502* is equivalent to  
28805 *-c 750228235959*.28806 *-a* Request writing of information for both removed, that is, *delta type=R* (see *rm del*  
28807 (on page 3027)) and existing, that is, *delta type=D*, deltas. If the *-a* option is not  
28808 specified, information for existing deltas only shall be provided.

## 28809 OPERANDS

28810 The following operand shall be supported:

28811 *file* A pathname of an existing SCCS file or a directory. If *file* is a directory, the *prs*  
28812 utility shall behave as though each file in the directory were specified as a named  
28813 file, except that non-SCCS files (last component of the pathname does not begin

28814 with s.) and unreadable files shall be silently ignored.

28815 If exactly one *file* operand appears, and it is '-', the standard input shall be read;  
28816 each line of the standard input shall be taken to be the name of an SCCS file to be  
28817 processed. Non-SCCS files and unreadable files shall be silently ignored.

28818 **STDIN**

28819 The standard input shall be a text file used only when the *file* operand is specified as '-'. Each  
28820 line of the text file shall be interpreted as an SCCS pathname.

28821 **INPUT FILES**

28822 Any SCCS files displayed are files of an unspecified format.

28823 **ENVIRONMENT VARIABLES**

28824 The following environment variables shall affect the execution of *prs*:

28825 *LANG* Provide a default value for the internationalization variables that are unset or null.  
28826 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
28827 Internationalization Variables for the precedence of internationalization variables  
28828 used to determine the values of locale categories.)

28829 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
28830 internationalization variables.

28831 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
28832 characters (for example, single-byte as opposed to multi-byte characters in  
28833 arguments and input files).

28834 *LC\_MESSAGES*

28835 Determine the locale that should be used to affect the format and contents of  
28836 diagnostic messages written to standard error.

28837 *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

28838 **ASYNCHRONOUS EVENTS**

28839 Default.

28840 **STDOUT**

28841 The standard output shall be a text file whose format is dependent on the data keywords  
28842 specified with the **-d** option.

28843 **Data Keywords**

28844 Data keywords specify which parts of an SCCS file shall be retrieved and output. All parts of an  
28845 SCCS file have an associated data keyword. A data keyword may appear in a *dataspec* multiple  
28846 times.

28847 The information written by *prs* shall consist of:

28848 1. The user-supplied text

28849 2. Appropriate values (extracted from the SCCS file) substituted for the recognized data  
28850 keywords in the order of appearance in the *dataspec*

28851 The format of a data keyword value shall either be simple ('S'), in which keyword substitution  
28852 is direct, or multi-line ('M').

28853 User-supplied text shall be any text other than recognized data keywords. A <tab> shall be  
28854 specified by '\t' and <newline> by '\n'. When the **-r** option is not specified, the default  
28855 *dataspec* shall be:

28856 :PN: :\n\n

28857 and the following *dataspec* shall be used for each selected delta:

28858 :Dt: \t:DL: \nMRs: \n:MR: COMMENTS: \n:C:

28859

28860

28861

28862

28863

28864

28865

28866

28867

28868

28869

28870

28871

28872

28873

28874

28875

28876

28877

28878

28879

28880

28881

28882

28883

28884

28885

28886

28887

28888

28889

28890

28891

28892

28893

28894

28895

28896

28897

28898

28899

28900

28901

28902

| SCCS File Data Keywords |                                                         |              |                |        |
|-------------------------|---------------------------------------------------------|--------------|----------------|--------|
| Keyword                 | Data Item                                               | File Section | Value          | Format |
| :Dt:                    | Delta information                                       | Delta Table  | See below*     | S      |
| :DL:                    | Delta line statistics                                   | "            | :Li:/:Ld:/:Lu: | S      |
| :Li:                    | Lines inserted by Delta                                 | "            | nnnnn***       | S      |
| :Ld:                    | Lines deleted by Delta                                  | "            | nnnnn***       | S      |
| :Lu:                    | Lines unchanged by Delta                                | "            | nnnnn***       | S      |
| :DT:                    | Delta type                                              | "            | D or R         | S      |
| :I:                     | SCCS ID string (SID)                                    | "            | See below**    | S      |
| :R:                     | Release number                                          | "            | nnnn           | S      |
| :L:                     | Level number                                            | "            | nnnn           | S      |
| :B:                     | Branch number                                           | "            | nnnn           | S      |
| :S:                     | Sequence number                                         | "            | nnnn           | S      |
| :D:                     | Date delta created                                      | "            | :Dy:/:Dm:/:Dd: | S      |
| :Dy:                    | Year delta created                                      | "            | nn             | S      |
| :Dm:                    | Month delta created                                     | "            | nn             | S      |
| :Dd:                    | Day delta created                                       | "            | nn             | S      |
| :T:                     | Time delta created                                      | "            | :Th:::Tm:::Ts: | S      |
| :Th:                    | Hour delta created                                      | "            | nn             | S      |
| :Tm:                    | Minutes delta created                                   | "            | nn             | S      |
| :Ts:                    | Seconds delta created                                   | "            | nn             | S      |
| :P:                     | Programmer who created Delta                            | "            | logname        | S      |
| :DS:                    | Delta sequence number                                   | "            | nnnn           | S      |
| :DP:                    | Predecessor Delta sequence number                       | "            | nnnn           | S      |
| :DI:                    | Sequence number of deltas included, excluded or ignored | "            | :Dn:/:Dx:/:Dg: | S      |
| :Dn:                    | Deltas included (sequence #)                            | "            | :DS: :DS: ...  | S      |
| :Dx:                    | Deltas excluded (sequence #)                            | "            | :DS: :DS: ...  | S      |
| :Dg:                    | Deltas ignored (sequence #)                             | "            | :DS: :DS: ...  | S      |
| :MR:                    | MR numbers for delta                                    | "            | text           | M      |
| :C:                     | Comments for delta                                      | "            | text           | M      |
| :UN:                    | User names                                              | User Names   | text           | M      |
| :FL:                    | Flag list                                               | Flags        | text           | M      |
| :Y:                     | Module type flag                                        | "            | text           | S      |
| :MF:                    | MR validation flag                                      | "            | yes or no      | S      |
| :MP:                    | MR validation program name                              | "            | text           | S      |
| :KF:                    | Keyword error, warning flag                             | "            | yes or no      | S      |
| :KV:                    | Keyword validation string                               | "            | text           | S      |
| :BF:                    | Branch flag                                             | "            | yes or no      | S      |
| :J:                     | Joint edit flag                                         | "            | yes or no      | S      |
| :LK:                    | Locked releases                                         | "            | :R: ...        | S      |
| :Q:                     | User-defined keyword                                    | "            | text           | S      |

28903

28904

28905

28906

28907

28908

28909

28910

28911

28912

28913

28914

28915

28916

28917

28918

| SCCS File Data Keywords |                              |              |                   |        |
|-------------------------|------------------------------|--------------|-------------------|--------|
| Keyword                 | Data Item                    | File Section | Value             | Format |
| :M:                     | Module name                  | "            | <i>text</i>       | S      |
| :FB:                    | Floor boundary               | "            | :R:               | S      |
| :CB:                    | Ceiling boundary             | "            | :R:               | S      |
| :Ds:                    | Default SID                  | "            | :I:               | S      |
| :ND:                    | Null delta flag              | "            | yes or no         | S      |
| :FD:                    | File descriptive text        | Comments     | <i>text</i>       | M      |
| :BD:                    | Body                         | Body         | <i>text</i>       | M      |
| :GB:                    | Gotten body                  | "            | <i>text</i>       | M      |
| :W:                     | A form of <i>what</i> string | N/A          | :Z::M:\t:I:       | S      |
| :A:                     | A form of <i>what</i> string | N/A          | :Z::Y: :M: :I::Z: | S      |
| :Z:                     | <i>what</i> string delimiter | N/A          | @( # )            | S      |
| :F:                     | SCCS filename                | N/A          | <i>text</i>       | S      |
| :PN:                    | SCCS file pathname           | N/A          | <i>text</i>       | S      |

28919

\* :Dt=:DT: :I: :D: :T: :P: :DS: :DP:

28920

\*\* :R::L::B::S: if the delta is a branch delta (:BF:= =yes)

28921

:R::L: if the delta is not a branch delta (:BF:= =no)

28922

\*\*\* The line statistics are capped at 99 999. For example, if 100 000 lines were unchanged in a

28923

certain revision, :Lu: shall produce the value 99 999.

28924 **STDERR**

28925

The standard error shall be used only for diagnostic messages.

28926 **OUTPUT FILES**

28927

None.

28928 **EXTENDED DESCRIPTION**

28929

None.

28930 **EXIT STATUS**

28931

The following exit values shall be returned:

28932

0 Successful completion.

28933

>0 An error occurred.

28934 **CONSEQUENCES OF ERRORS**

28935

Default.

28936 **APPLICATION USAGE**

28937

None.

28938 **EXAMPLES**

28939

1. The following example:

28940

```
prs -d "User Names for :F: are:\n:UN:" s.file
```

28941

might write to standard output:

28942

```
User Names for s.file are:
```

28943

```
xyz
```

28944

```
131
```

28945

```
abc
```

- 28946           2. The following example:
- 28947            `prs -d "Delta for pgm :M:: :I: - :D: By :P:" -r s.file`
- 28948            might write to standard output:
- 28949            `Delta for pgm main.c: 3.7 - 77/12/01 By cas`
- 28950           3. As a special case:
- 28951            `prs s.file`
- 28952            might write to standard output:
- 28953            `s.file:`
- 28954            `<blank line>`
- 28955            `D 1.1 77/12/01 00:00:00 cas 1 000000/00000/00000`
- 28956            `MRs:`
- 28957            `b178-12345`
- 28958            `b179-54321`
- 28959            `COMMENTS:`
- 28960            `this is the comment line for s.file initial delta`
- 28961            `<blank line>`
- 28962            for each delta table entry of the **D** type. The only option allowed to be used with this
- 28963            special case is the **-a** option.
- 28964 **RATIONALE**
- 28965            None.
- 28966 **FUTURE DIRECTIONS**
- 28967            None.
- 28968 **SEE ALSO**
- 28969            `admin, delta, get, what`
- 28970 **CHANGE HISTORY**
- 28971            First released in Issue 2.
- 28972 **Issue 5**
- 28973            The phrase “in which keyword substitution is followed by a <newline>” is deleted from the end
- 28974            of the second paragraph of **Data Keywords** (on page 2946).
- 28975            The interpretation of the **YY** component of the **-c cutoff** argument is noted.
- 28976 **Issue 6**
- 28977            The normative text is reworded to emphasize the term “shall” for implementation requirements.
- 28978            The Open Group Base Resolution bwg2001-007 is applied, updating the table in **STDOUT** with a
- 28979            note that line statistics are capped at 99 999 for the **:Li:**, **:Ld:**, **:Lu:**, and **:DL:** keywords.
- 28980            The Open Group Interpretation **PIN4C.00009** is applied.

## 28981 NAME

28982 ps — report process status

## 28983 SYNOPSIS

28984 UP XSI ps [-aA][--defl][--G *grouplist*][--o *format*]...[--p *proclist*][--t *termlist*]28985 [--U *userlist*][--g *grouplist*][--n *namelist*][--u *userlist*]

28986

## 28987 DESCRIPTION

28988 The *ps* utility shall write information about processes, subject to having the appropriate  
28989 privileges to obtain information about those processes.28990 By default, *ps* shall select all processes with the same effective user ID as the current user and the  
28991 same controlling terminal as the invoker.

## 28992 OPTIONS

28993 The *ps* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2,  
28994 Utility Syntax Guidelines.

28995 The following options shall be supported:

28996 **-a** Write information for all processes associated with terminals. Implementations  
28997 may omit session leaders from this list.28998 **-A** Write information for all processes.28999 XSI **-d** Write information for all processes, except session leaders.29000 XSI **-e** Write information for all processes. (Equivalent to **-A**.)29001 XSI **-f** Generate a **full** listing. (See the STDOUT section for the contents of a **full** listing.)29002 XSI **-g *grouplist*** Write information for processes whose session leaders are given in *grouplist*. The  
29003 application shall ensure that the *grouplist* is a single argument in the form of a  
29004 <blank> or comma-separated list.29005 **-G *grouplist*** Write information for processes whose real group ID numbers are given in  
29006 *grouplist*. The application shall ensure that the *grouplist* is a single argument in the  
29007 form of a <blank> or comma-separated list.29008 XSI **-l** Generate a **long** listing. (See STDOUT for the contents of a **long** listing.)29009 XSI **-n *namelist*** Specify the name of an alternative system *namelist* file in place of the default. The  
29010 name of the default file and the format of a *namelist* file are unspecified.29011 **-o *format*** Write information according to the format specification given in *format*. This is  
29012 fully described in the STDOUT section. Multiple **-o** options can be specified; the  
29013 format specification shall be interpreted as the <space>-separated concatenation of  
29014 all the *format* option-arguments.29015 **-p *proclist*** Write information for processes whose process ID numbers are given in *proclist*.  
29016 The application shall ensure that the *proclist* is a single argument in the form of a  
29017 <blank> or comma-separated list.29018 **-t *termlist*** Write information for processes associated with terminals given in *termlist*. The  
29019 application shall ensure that the *termlist* is a single argument in the form of a  
29020 <blank> or comma-separated list. Terminal identifiers shall be given in an  
29021 XSI implementation-defined format. On XSI-conformant systems, they shall be given  
29022 in one of two forms: the device's filename (for example, **tty04**) or, if the device's  
29023 filename starts with **tty**, just the identifier following the characters **tty** (for

- 29024                   example, "04").
- 29025 XSI    **-u *userlist***   Write information for processes whose user ID numbers or login names are given in *userlist*. The application shall ensure that the *userlist* is a single argument in the form of a <blank> or comma-separated list. In the listing, the numerical user ID shall be written unless the **-f** option is used, in which case the login name shall be written.
- 29026
- 29027
- 29028
- 29029
- 29030    **-U *userlist***   Write information for processes whose real user ID numbers or login names are given in *userlist*. The application shall ensure that the *userlist* is a single argument in the form of a <blank> or comma-separated list.
- 29031
- 29032
- 29033           With the exception of **-o *format***, all of the options shown are used to select processes. If any are specified, the default list shall be ignored and *ps* shall select the processes represented by the inclusive OR of all the selection-criteria options.
- 29034
- 29035
- 29036 **OPERANDS**
- 29037           None.
- 29038 **STDIN**
- 29039           Not used.
- 29040 **INPUT FILES**
- 29041           None.
- 29042 **ENVIRONMENT VARIABLES**
- 29043           The following environment variables shall affect the execution of *ps*:
- 29044    **COLUMNS**   Override the system-selected horizontal display line size, used to determine the number of text columns to display. See the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 8, Environment Variables for valid values and results when it is unset or null.
- 29045
- 29046
- 29047
- 29048    **LANG**       Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)
- 29049
- 29050
- 29051
- 29052    **LC\_ALL**      If set to a non-empty string value, override the values of all the other internationalization variables.
- 29053
- 29054    **LC\_CTYPE**   Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).
- 29055
- 29056
- 29057    **LC\_MESSAGES**
- 29058           Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error and informative messages written to standard output.
- 29059
- 29060
- 29061    **LC\_TIME**     Determine the format and contents of the date and time strings displayed.
- 29062 XSI    **NLSPATH**     Determine the location of message catalogs for the processing of **LC\_MESSAGES**.
- 29063    **TZ**         Determine the timezone used to calculate date and time strings displayed. If **TZ** is unset or null, an unspecified default timezone shall be used.
- 29064

29065 **ASYNCHRONOUS EVENTS**

29066 Default.

29067 **STDOUT**29068 When the **-o** option is not specified, the standard output format is unspecified.

29069 XSI On XSI-conformant systems, the output format shall be as follows. The column headings and  
 29070 descriptions of the columns in a *ps* listing are given below. The precise meanings of these fields  
 29071 are implementation-defined. The letters 'f' and 'l' (below) indicate the option (**full** or **long**)  
 29072 that shall cause the corresponding heading to appear; **all** means that the heading always  
 29073 appears. Note that these two options determine only what information is provided for a process;  
 29074 they do not determine which processes are listed.

|       |              |       |                                                                                                            |
|-------|--------------|-------|------------------------------------------------------------------------------------------------------------|
| 29075 | <b>F</b>     | (l)   | Flags (octal and additive) associated with the process.                                                    |
| 29076 | <b>S</b>     | (l)   | The state of the process.                                                                                  |
| 29077 | <b>UID</b>   | (f,l) | The user ID number of the process owner; the login name is printed<br>29078 under the <b>-f</b> option.    |
| 29079 | <b>PID</b>   | (all) | The process ID of the process; it is possible to kill a process if this<br>29080 datum is known.           |
| 29081 | <b>PPID</b>  | (f,l) | The process ID of the parent process.                                                                      |
| 29082 | <b>C</b>     | (f,l) | Processor utilization for scheduling.                                                                      |
| 29083 | <b>PRI</b>   | (l)   | The priority of the process; higher numbers mean lower priority.                                           |
| 29084 | <b>NI</b>    | (l)   | Nice value; used in priority computation.                                                                  |
| 29085 | <b>ADDR</b>  | (l)   | The address of the process.                                                                                |
| 29086 | <b>SZ</b>    | (l)   | The size in blocks of the core image of the process.                                                       |
| 29087 | <b>WCHAN</b> | (l)   | The event for which the process is waiting or sleeping; if blank, the<br>29088 process is running.         |
| 29089 | <b>STIME</b> | (f)   | Starting time of the process.                                                                              |
| 29090 | <b>TTY</b>   | (all) | The controlling terminal for the process.                                                                  |
| 29091 | <b>TIME</b>  | (all) | The cumulative execution time for the process.                                                             |
| 29092 | <b>CMD</b>   | (all) | The command name; the full command name and its arguments are<br>29093 written under the <b>-f</b> option. |

29094 A process that has exited and has a parent, but has not yet been waited for by the parent, shall be  
 29095 marked **defunct**.

29096 Under the option **-f**, *ps* tries to determine the command name and arguments given when the  
 29097 process was created by examining memory or the swap area. Failing this, the command name, as  
 29098 it would appear without the option **-f**, is written in square brackets.

29099 The **-o** option allows the output format to be specified under user control.

29100 The application shall ensure that the format specification is a list of names presented as a single  
 29101 argument, <blank> or comma-separated. Each variable has a default header. The default header  
 29102 can be overridden by appending an equals sign and the new text of the header. The rest of the  
 29103 characters in the argument shall be used as the header text. The fields specified shall be written  
 29104 in the order specified on the command line, and should be arranged in columns in the output.  
 29105 The field widths shall be selected by the system to be at least as wide as the header text (default  
 29106 or overridden value). If the header text is null, such as **-o user=**, the field width shall be at least  
 29107 as wide as the default header text. If all header text fields are null, no header line shall be  
 29108 written.

29109 The following names are recognized in the POSIX locale:

|       |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|-------|---------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 29110 | <b>ruser</b>  | The real user ID of the process. This shall be the textual user ID, if it can be obtained and the field width permits, or a decimal representation otherwise.                                                                                                                                                                                                                                                                                                                                                                                            |
| 29111 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 29112 | <b>user</b>   | The effective user ID of the process. This shall be the textual user ID, if it can be obtained and the field width permits, or a decimal representation otherwise.                                                                                                                                                                                                                                                                                                                                                                                       |
| 29113 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 29114 | <b>rgroup</b> | The real group ID of the process. This shall be the textual group ID, if it can be obtained and the field width permits, or a decimal representation otherwise.                                                                                                                                                                                                                                                                                                                                                                                          |
| 29115 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 29116 | <b>group</b>  | The effective group ID of the process. This shall be the textual group ID, if it can be obtained and the field width permits, or a decimal representation otherwise.                                                                                                                                                                                                                                                                                                                                                                                     |
| 29117 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 29118 | <b>pid</b>    | The decimal value of the process ID.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 29119 | <b>ppid</b>   | The decimal value of the parent process ID.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 29120 | <b>pgid</b>   | The decimal value of the process group ID.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 29121 | <b>pcpu</b>   | The ratio of CPU time used recently to CPU time available in the same period, expressed as a percentage. The meaning of “recently” in this context is unspecified. The CPU time available is determined in an unspecified manner.                                                                                                                                                                                                                                                                                                                        |
| 29122 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 29123 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 29124 | <b>vsz</b>    | The size of the process in (virtual) memory in 1 024 byte units as a decimal integer.                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 29125 | <b>nice</b>   | The decimal value of the nice value of the process; see <i>nice</i> (on page 2863).                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 29126 | <b>etime</b>  | In the POSIX locale, the elapsed time since the process was started, in the form:                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 29127 |               | [ [ <i>dd</i> -] <i>hh</i> : ] <i>mm</i> : <i>ss</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 29128 |               | where <i>dd</i> shall represent the number of days, <i>hh</i> the number of hours, <i>mm</i> the number of minutes, and <i>ss</i> the number of seconds. The <i>dd</i> field shall be a decimal integer. The <i>hh</i> , <i>mm</i> , and <i>ss</i> fields shall be two-digit decimal integers padded on the left with zeros.                                                                                                                                                                                                                             |
| 29129 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 29130 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 29131 | <b>time</b>   | In the POSIX locale, the cumulative CPU time of the process in the form:                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 29132 |               | [ [ <i>dd</i> -] <i>hh</i> : <i>mm</i> : ] <i>ss</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 29133 |               | The <i>dd</i> , <i>hh</i> , <i>mm</i> , and <i>ss</i> fields shall be as described in the <b>etime</b> specifier.                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 29134 | <b>tty</b>    | The name of the controlling terminal of the process (if any) in the same format used by the <i>who</i> utility.                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 29135 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 29136 | <b>comm</b>   | The name of the command being executed ( <i>argv</i> [0] value) as a string.                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 29137 | <b>args</b>   | The command with all its arguments as a string. The implementation may truncate this value to the field width; it is implementation-defined whether any further truncation occurs. It is unspecified whether the string represented is a version of the argument list as it was passed to the command when it started, or is a version of the arguments as they may have been modified by the application. Applications cannot depend on being able to modify their argument list and having that modification be reflected in the output of <i>ps</i> . |
| 29138 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 29139 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 29140 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 29141 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 29142 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 29143 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 29144 |               | Any field need not be meaningful in all implementations. In such a case a hyphen (‘-’) should be output in place of the field value.                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 29145 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 29146 |               | Only <b>comm</b> and <b>args</b> shall be allowed to contain <blank>s; all others shall not. Any implementation-defined variables shall be specified in the system documentation along with the default header and indicating if the field may contain <blank>s.                                                                                                                                                                                                                                                                                         |
| 29147 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 29148 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 29149 |               | The following table specifies the default header to be used in the POSIX locale corresponding to each format specifier.                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 29150 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |

29151

Table 4-17 Variable Names and Default Headers in *ps*

29152

| Format Specifier | Default Header | Format Specifier | Default Header |
|------------------|----------------|------------------|----------------|
| <b>args</b>      | <b>COMMAND</b> | <b>ppid</b>      | <b>PPID</b>    |
| <b>comm</b>      | <b>COMMAND</b> | <b>rgroup</b>    | <b>RGROUP</b>  |
| <b>etime</b>     | <b>ELAPSED</b> | <b>ruser</b>     | <b>RUSER</b>   |
| <b>group</b>     | <b>GROUP</b>   | <b>time</b>      | <b>TIME</b>    |
| <b>nice</b>      | <b>NI</b>      | <b>tty</b>       | <b>TT</b>      |
| <b>pcpu</b>      | <b>%CPU</b>    | <b>user</b>      | <b>USER</b>    |
| <b>pgid</b>      | <b>PGID</b>    | <b>vsz</b>       | <b>VSZ</b>     |
| <b>pid</b>       | <b>PID</b>     |                  |                |

29153

29154

29155

29156

29157

29158

29159

29160

**29161 STDERR**

29162 The standard error shall be used only for diagnostic messages. |

**29163 OUTPUT FILES**

29164 None.

**29165 EXTENDED DESCRIPTION**

29166 None.

**29167 EXIT STATUS**

29168 The following exit values shall be returned:

29169 0 Successful completion.

29170 >0 An error occurred.

**29171 CONSEQUENCES OF ERRORS**

29172 Default.

**29173 APPLICATION USAGE**

29174 Things can change while *ps* is running; the snapshot it gives is only true for an instant, and might  
29175 not be accurate by the time it is displayed.

29176 The **args** format specifier is allowed to produce a truncated version of the command arguments.  
29177 In some implementations, this information is no longer available when the *ps* utility is executed.

29178 If the field width is too narrow to display a textual ID, the system may use a numeric version.  
29179 Normally, the system would be expected to choose large enough field widths, but if a large  
29180 number of fields were selected to write, it might squeeze fields to their minimum sizes to fit on  
29181 one line. One way to ensure adequate width for the textual IDs is to override the default header  
29182 for a field to make it larger than most or all user or group names.

29183 There is no special quoting mechanism for header text. The header text is the rest of the  
29184 argument. If multiple header changes are needed, multiple **-o** options can be used, such as:

29185 `ps -o "user=User Name" -o pid=Process\ ID`

29186 On some implementations, especially multi-level secure systems, *ps* may be severely restricted |  
29187 and produce information only about child processes owned by the user.

**29188 EXAMPLES**

29189 The command:

29190 `ps -o user,pid,ppid=MOM -o args`

29191 writes at least the following in the POSIX locale:

29192 USER PID MOM COMMAND

29193 helene 34 12 ps -o uid,pid,ppid=MOM -o args

29194 The contents of the **COMMAND** field need not be the same in all implementations, due to  
29195 possible truncation.

29196 **RATIONALE**

29197 There is very little commonality between BSD and System V implementations of *ps*. Many  
29198 options conflict or have subtly different usages. The standard developers attempted to select a  
29199 set of options for the base standard that were useful on a wide range of systems and selected  
29200 options that either can be implemented on both BSD and System V-based systems without  
29201 breaking the current implementations or where the options are sufficiently similar that any  
29202 changes would not be unduly problematic for users or implementors.

29203 It is recognized that on some implementations, especially multi-level secure systems, *ps* may be  
29204 nearly useless. The default output has therefore been chosen such that it does not break  
29205 historical implementations and also is likely to provide at least some useful information on most  
29206 systems.

29207 The major change is the addition of the format specification capability. The motivation for this  
29208 invention is to provide a mechanism for users to access a wider range of system information, if  
29209 the system permits it, in a portable manner. The fields chosen to appear in this volume of  
29210 IEEE Std 1003.1-200x were arrived at after considering what concepts were likely to be both  
29211 reasonably useful to the “average” user and had a reasonable chance of being implemented on a  
29212 wide range of systems. Again it is recognized that not all systems are able to provide all the  
29213 information and, conversely, some may wish to provide more. It is hoped that the approach  
29214 adopted will be sufficiently flexible and extensible to accommodate most systems.  
29215 Implementations may be expected to introduce new format specifiers.

29216 The default output should consist of a short listing containing the process ID, terminal name,  
29217 cumulative execution time, and command name of each process.

29218 The preference of the standard developers would have been to make the format specification an  
29219 operand of the *ps* command. Unfortunately, BSD usage precluded this.

29220 At one time a format was included to display the environment array of the process. This was  
29221 deleted because there is no portable way to display it.

29222 The **-A** option is equivalent to the BSD **-g** and the SVID **-e**. Because the two systems differed, a  
29223 mnemonic compromise was selected.

29224 The **-a** option is described with some optional behavior because the SVID omits session leaders,  
29225 but BSD does not.

29226 In an early proposal, format specifiers appeared for priority and start time. The former was not  
29227 defined adequately in this volume of IEEE Std 1003.1-200x and was removed in deference to the  
29228 defined nice value; the latter because elapsed time was considered to be more useful.

29229 In a new BSD version of *ps*, a **-O** option can be used to write all of the default information,  
29230 followed by additional format specifiers. This was not adopted because the default output is  
29231 implementation-defined. Nevertheless, this is a useful option that should be reserved for that  
29232 purpose. In the **-o** option for the POSIX Shell and Utilities *ps*, the format is the concatenation of  
29233 each **-o**. Therefore, the user can have an alias or function that defines the beginning of their  
29234 desired format and add more fields to the end of the output in certain cases where that would be  
29235 useful.

29236 The format of the terminal name is unspecified, but the descriptions of *ps*, *talk*, *who*, and *write*  
29237 require that they all use the same format.

29238 The **pcpu** field indicates that the CPU time available is determined in an unspecified manner.  
29239 This is because it is difficult to express an algorithm that is useful across all possible machine

29240 architectures. Historical counterparts to this value have attempted to show percentage of use in  
29241 the recent past, such as the preceding minute. Frequently, these values for all processes did not  
29242 add up to 100%. Implementations are encouraged to provide data in this field to users that will  
29243 help them identify processes currently affecting the performance of the system.

29244 **FUTURE DIRECTIONS**

29245 None.

29246 **SEE ALSO**

29247 *kill, nice, renice*

29248 **CHANGE HISTORY**

29249 First released in Issue 2.

29250 **Issue 6**

29251 This utility is now marked as part of the User Portability Utilities option.

29252 The normative text is reworded to avoid use of the term “must” for application requirements.

29253 The *TZ* entry is added to the ENVIRONMENT VARIABLES section.

29254 **NAME**

29255           pwd — return working directory name

29256 **SYNOPSIS**

29257           pwd [-L | -P ]

29258 **DESCRIPTION**29259           The *pwd* utility shall write to standard output an absolute pathname of the current working  
29260           directory, which does not contain the filenames dot or dot-dot.29261 **OPTIONS**29262           The *pwd* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section  
29263           12.2, Utility Syntax Guidelines.

29264           The following options shall be supported by the implementation:

29265           **-L**           If the *PWD* environment variable contains an absolute pathname of the current  
29266           directory that does not contain the filenames dot or dot-dot, *pwd* shall write this  
29267           pathname to standard output. Otherwise, the **-L** option shall behave as the **-P**  
29268           option.29269           **-P**           The absolute pathname written shall not contain filenames that, in the context of  
29270           the pathname, refer to files of type symbolic link.29271           If both **-L** and **-P** are specified, the last one shall apply. If neither **-L** nor **-P** is specified, the *pwd*  
29272           utility shall behave as if **-L** had been specified.29273 **OPERANDS**

29274           None.

29275 **STDIN**

29276           Not used.

29277 **INPUT FILES**

29278           None.

29279 **ENVIRONMENT VARIABLES**29280           The following environment variables shall affect the execution of *pwd*:29281           **LANG**           Provide a default value for the internationalization variables that are unset or null.  
29282           (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
29283           Internationalization Variables for the precedence of internationalization variables  
29284           used to determine the values of locale categories.)29285           **LC\_ALL**          If set to a non-empty string value, override the values of all the other  
29286           internationalization variables.29287           **LC\_MESSAGES**29288           Determine the locale that should be used to affect the format and contents of  
29289           diagnostic messages written to standard error.29290 **XSI**           **NLSPATH**       Determine the location of message catalogs for the processing of *LC\_MESSAGES*.29291           **PWD**           If the **-P** option is in effect, this variable shall be set to an absolute pathname of the  
29292           current working directory that does not contain any components that specify  
29293           symbolic links, does not contain any components that are dot, and does not  
29294           contain any components that are dot-dot. If an application sets or unsets the value  
29295           of *PWD*, the behavior of *pwd* is unspecified.

29296 **ASYNCHRONOUS EVENTS**

29297 Default.

29298 **STDOUT**29299 The *pwd* utility output is an absolute pathname of the current working directory:

29300 "%s\n", &lt;directory pathname&gt;

29301 **STDERR**

29302 The standard error shall be used only for diagnostic messages.

29303 **OUTPUT FILES**

29304 None.

29305 **EXTENDED DESCRIPTION**

29306 None.

29307 **EXIT STATUS**

29308 The following exit values shall be returned:

29309 0 Successful completion.

29310 &gt;0 An error occurred.

29311 **CONSEQUENCES OF ERRORS**29312 If an error is detected, output shall not be written to standard output, a diagnostic message shall  
29313 be written to standard error, and the exit status is not zero.29314 **APPLICATION USAGE**

29315 None.

29316 **EXAMPLES**

29317 None.

29318 **RATIONALE**29319 Some implementations have historically provided *pwd* as a shell special built-in command.29320 In most utilities, if an error occurs, partial output may be written to standard output. This does  
29321 not happen in historical implementations of *pwd*. Because *pwd* is frequently used in historical  
29322 shell scripts without checking the exit status, it is important that the historical behavior is  
29323 required here; therefore, the CONSEQUENCES OF ERRORS section specifically disallows any  
29324 partial output being written to standard output.29325 **FUTURE DIRECTIONS**

29326 None.

29327 **SEE ALSO**29328 *cd*, the System Interfaces volume of IEEE Std 1003.1-200x, *getcwd()*29329 **CHANGE HISTORY**

29330 First released in Issue 2.

29331 **Issue 6**29332 The **-P** and **-L** options are added to describe actions relating to symbolic links as specified in the  
29333 IEEE P1003.2b draft standard.

29334 **NAME**

29335 qalter — alter batch job

29336 **SYNOPSIS**

```
29337 BE qalter [-a date_time][-A account_string][-c interval][-e path_name]
29338 [-h hold_list][-j join_list][-k keep_list][-l resource_list]
29339 [-m mail_options][-M mail_list][-N name][-o path_name]
29340 [-p priority][-r y|n][-S path_name_list][-u user_list]
29341 job_identifier ...
29342
```

29343 **DESCRIPTION**

29344 The attributes of a batch job are altered by a request to the batch server that manages the batch  
 29345 job. The *qalter* utility is a user-accessible batch client that requests the alteration of the attributes  
 29346 of one or more batch jobs.

29347 The *qalter* utility shall alter the attributes of those batch jobs, and only those batch jobs, for which  
 29348 a batch *job\_identifier* is presented to the utility.

29349 The *qalter* utility shall alter the attributes of batch jobs in the order in which the batch  
 29350 *job\_identifiers* are presented to the utility.

29351 If the *qalter* utility fails to process a batch *job\_identifier* successfully, the utility shall proceed to  
 29352 process the remaining batch *job\_identifiers*, if any.

29353 For each batch *job\_identifier* for which the *qalter* utility succeeds, each attribute of the identified  
 29354 batch job shall be altered as indicated by all the options presented to the utility.

29355 For each identified batch job for which the *qalter* utility fails, the utility shall not alter any  
 29356 attribute of the batch job.

29357 For each batch job that the *qalter* utility processes, the utility shall not modify any attribute other  
 29358 than those required by the options and option-arguments presented to the utility.

29359 The *qalter* utility shall alter batch jobs by sending a *Modify Job Request* to the batch server that  
 29360 manages each batch job. At the time the *qalter* utility exits, it shall have modified the batch job  
 29361 corresponding to each successfully processed batch *job\_identifier*. An attempt to alter the  
 29362 attributes of a batch job in the RUNNING state is implementation-defined.

29363 **OPTIONS**

29364 The *qalter* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section  
 29365 12.2, Utility Syntax Guidelines.

29366 The following options shall be supported by the implementation:

29367 **-a *date\_time*** Redefine the time at which the batch job becomes eligible for execution.

29368 The *date\_time* argument shall be in the same form and represent the same time as  
 29369 for the *touch* utility. The time so represented shall be set into the *Execution\_Time*  
 29370 attribute of the batch job. If the time specified is earlier than the current time, the  
 29371 **-a** option shall have no effect.

29372 **-A *account\_string***

29373 Redefine the account to which the resource consumption of the batch job should be  
 29374 charged.

29375 The syntax of the *account\_string* option-argument is unspecified.

29376 The *qalter* utility shall set the *Account\_Name* attribute of the batch job to the value  
 29377 of the *account\_string* option-argument.

- 29378        **-c interval**   Redefine whether the batch job should be checkpointed, and if so, how often.
- 29379                    The *qalter* utility shall accept a value for the interval option-argument that is one of  
29380                    the following:
- 29381                    n            No checkpointing is to be performed on the batch batch job  
29382                    (NO\_CHECKPOINT).
- 29383                    s            Checkpointing is to be performed only when the batch server is shut  
29384                    down (CHECKPOINT\_AT\_SHUTDOWN).
- 29385                    c            Automatic periodic checkpointing is to be performed at the  
29386                    *Minimum\_Cpu\_Interval* attribute of the batch queue, in units of CPU  
29387                    minutes (CHECKPOINT\_AT\_MIN\_CPU\_INTERVAL).
- 29388                    c=*minutes*   Automatic periodic checkpointing is to be performed every *minutes*  
29389                    of CPU time, or every *Minimum\_Cpu\_Interval* minutes, whichever is  
29390                    greater. The *minutes* argument shall conform to the syntax for  
29391                    unsigned integers and shall be greater than zero.
- 29392                    An implementation may define other checkpoint intervals. The conformance  
29393                    document for an implementation shall describe any alternative checkpoint  
29394                    intervals, how they are specified, their internal behavior, and how they affect the  
29395                    behavior of the utility.
- 29396                    The *qalter* utility shall set the *Checkpoint* attribute of the batch job to the value of the  
29397                    *interval* option-argument.
- 29398        **-e path\_name** Redefine the path to be used for the standard error stream of the batch job.
- 29399                    The *qalter* utility shall accept a *path\_name* option-argument that conforms to the  
29400                    syntax of the *path\_name* element defined in the System Interfaces volume of  
29401                    IEEE Std 1003.1-200x, which can be preceded by a host name element of the form  
29402                    *hostname:*.
- 29403                    If the *path\_name* option-argument constitutes an absolute pathname, the *qalter*  
29404                    utility shall set the *Error\_Path* attribute of the batch job to the value of the  
29405                    *path\_name* option-argument, including the host name element, if present.
- 29406                    If the *path\_name* option-argument constitutes a relative pathname and no host  
29407                    name element is specified, the *qalter* utility shall set the *Error\_Path* attribute of the  
29408                    batch job to the value of the absolute pathname derived by expanding the  
29409                    *path\_name* option-argument relative to the current directory of the process that  
29410                    executes the *qalter* utility.
- 29411                    If the *path\_name* option-argument constitutes a relative pathname and a host name  
29412                    element is specified, the *qalter* utility shall set the *Error\_Path* attribute of the batch  
29413                    job to the value of the option-argument without expansion.
- 29414                    If the *path\_name* option-argument does not include a host name element, the *qalter*  
29415                    utility shall prefix the pathname in the *Error\_Path* attribute with *hostname:*, where  
29416                    *hostname* is the name of the host upon which the *qalter* utility is being executed.
- 29417        **-h hold\_list** Redefine the types of holds, if any, on the batch job. The *qalter -h* option shall  
29418                    accept a value for the *hold\_list* option-argument that is a string of alphanumeric  
29419                    characters in the portable character set.
- 29420                    The *qalter* utility shall accept a value for the *hold\_list* option-argument that is a  
29421                    string of one or more of the characters 'u', 's', or 'o', or the single character  
29422                    'n'. For each unique character in the *hold\_list* option-argument, the *qalter* utility

- 29423 shall add a value to the *Hold\_Types* attribute of the batch job as follows, each  
29424 representing a different hold type:
- 29425 u USER
  - 29426 s SYSTEM
  - 29427 o OPERATOR
- 29428 If any of these characters are duplicated in the *hold\_list* option-argument, the  
29429 duplicates shall be ignored. An existing *Hold\_Types* attribute can be cleared by the  
29430 hold type:
- 29431 n NO\_HOLD
- 29432 The *qalter* utility shall consider it an error if any hold type other than 'n' is  
29433 combined with hold type 'n'. Strictly conforming applications shall not repeat  
29434 any of the characters 'u', 's', 'o', or 'n' within the *hold\_list* option-argument.  
29435 The *qalter* utility shall permit the repetition of characters, but shall not assign  
29436 additional meaning to the repeated characters. An implementation may define  
29437 other hold types. The conformance document for an implementation shall describe  
29438 any additional hold types, how they are specified, their internal behavior, and how  
29439 they affect the behavior of the utility.
- 29440 **-j *join\_list*** Redefine which streams of the batch job are to be merged. The *qalter* **-j** option shall  
29441 accept a value for the *join\_list* option-argument that is a string of alphanumeric  
29442 characters in the portable character set.
- 29443 The *qalter* utility shall accept a *join\_list* option-argument that consists of one or  
29444 more of the characters 'e' and 'o', or the single character 'n'.
- 29445 All of the other batch job output streams specified shall be merged into the output  
29446 stream represented by the character listed first in the *join\_list* option-argument.
- 29447 For each unique character in the *join\_list* option-argument, the *qalter* utility shall  
29448 add a value to the *Join\_Path* attribute of the batch job as follows, each representing  
29449 a different batch job stream to join:
- 29450 e The standard error of the batch batch job (JOIN\_STD\_ERROR).
  - 29451 o The standard output of the batch batch job (JOIN\_STD\_OUTPUT).
- 29452 An existing *Join\_Path* attribute can be cleared by the join type:
- 29453 n NO\_JOIN
- 29454 If 'n' is specified, then no files are joined. The *qalter* utility shall consider it an  
29455 error if any join type other than 'n' is combined with join type 'n'.
- 29456 Strictly conforming applications shall not repeat any of the characters 'e', 'o', or  
29457 'n' within the *join\_list* option-argument. The *qalter* utility shall permit the  
29458 repetition of characters, but shall not assign additional meaning to the repeated  
29459 characters.
- 29460 An implementation may define other join types. The conformance document for an  
29461 implementation shall describe any additional batch job streams, how they are  
29462 specified, their internal behavior, and how they affect the behavior of the utility.
- 29463 **-k *keep\_list*** Redefine which output of the batch job to retain on the execution host.
- 29464 The *qalter* **-k** option shall accept a value for the *keep\_list* option-argument that is a  
29465 string of alphanumeric characters in the portable character set.

- 29466 The *qalter* utility shall accept a *keep\_list* option-argument that consists of one or  
 29467 more of the characters 'e' and 'o' or the single character 'n'.
- 29468 For each unique character in the *keep\_list* option-argument, the *qalter* utility shall  
 29469 add a value to the *Keep\_Files* attribute of the batch job as follows, each representing  
 29470 a different batch job stream to keep:
- 29471 e The standard error of the batch batch job (KEEP\_STD\_ERROR).
  - 29472 o The standard output of the batch batch job (KEEP\_STD\_OUTPUT).
- 29473 If both 'e' and 'o' are specified, then both files are retained. An existing  
 29474 *Keep\_Files* attribute can be cleared by the keep type:
- 29475 n NO\_KEEP
- 29476 If 'n' is specified, then no files are retained. The *qalter* utility shall consider it an  
 29477 error if any keep type other than 'n' is combined with keep type 'n'.
- 29478 Strictly conforming applications shall not repeat any of the characters 'e', 'o', or  
 29479 'n' within the *keep\_list* option-argument. The *qalter* utility shall permit the  
 29480 repetition of characters, but shall not assign additional meaning to the repeated  
 29481 characters. An implementation may define other keep types. The conformance  
 29482 document for an implementation shall describe any additional keep types, how  
 29483 they are specified, their internal behavior, and how they affect the behavior of the  
 29484 utility.
- 29485 **-l *resource\_list***  
 29486 Redefine the resources that are allowed or required by the batch job.
- 29487 The *qalter* utility shall accept a *resource\_list* option-argument that conforms to the  
 29488 following syntax:
- 29489 resource=value[ , , resource=value , , . . . ]
- 29490 The *qalter* utility shall set one entry in the value of the *Resource\_List* attribute of the  
 29491 batch job for each resource listed in the *resource\_list* option-argument.
- 29492 Because the list of supported resource names might vary by batch server, the *qalter*  
 29493 utility shall rely on the batch server to validate the resource names and associated  
 29494 values. See Section 3.3.3 (on page 2325) for a means of removing *keyword=value*  
 29495 (and *value@keyword*) pairs and other general rules for list-oriented batch job  
 29496 attributes.
- 29497 **-m *mail\_options***  
 29498 Redefine the points in the execution of the batch job at which the batch server is to  
 29499 send mail about a change in the state of the batch job.
- 29500 The *qalter* **-m** option shall accept a value for the *mail\_options* option-argument that  
 29501 is a string of alphanumeric characters in the portable character set.
- 29502 The *qalter* utility shall accept a value for the *mail\_options* option-argument that is a  
 29503 string of one or more of the characters 'e', 'b', and 'a', or the single character  
 29504 'n'. For each unique character in the *mail\_options* option-argument, the *qalter*  
 29505 utility shall add a value to the *Mail\_Users* attribute of the batch job as follows, each  
 29506 representing a different time during the life of a batch job at which to send mail:
- 29507 e MAIL\_AT\_EXIT
  - 29508 b MAIL\_AT\_BEGINNING

- 29509 a MAIL\_AT\_ABORT
- 29510 If any of these characters are duplicated in the *mail\_options* option-argument, the  
29511 duplicates shall be ignored.
- 29512 An existing *Mail\_Points* attribute can be cleared by the mail type:
- 29513 n NO\_MAIL
- 29514 If 'n' is specified, then mail is not sent. The *qalter* utility shall consider it an error  
29515 if any mail type other than 'n' is combined with mail type 'n'. Strictly  
29516 conforming applications shall not repeat any of the characters 'e', 'b', 'a', or  
29517 'n' within the *mail\_options* option-argument. The *qalter* utility shall permit the  
29518 repetition of characters but shall not assign additional meaning to the repeated  
29519 characters.
- 29520 An implementation may define other mail types. The conformance document for  
29521 an implementation shall describe any additional mail types, how they are  
29522 specified, their internal behavior, and how they affect the behavior of the utility.
- 29523 **-M mail\_list** Redefine the list of users to which the batch server that executes the batch job is to  
29524 send mail, if the batch server sends mail about the batch job.
- 29525 The syntax of the *mail\_list* option-argument is unspecified. If the implementation  
29526 of the *qalter* utility uses a name service to locate users, the utility shall accept the  
29527 syntax used by the name service.
- 29528 If the implementation of the *qalter* utility does not use a name service to locate  
29529 users, the implementation shall accept the following syntax for user names:
- 29530 mail\_address[ , mail\_address , . . . ]
- 29531 The interpretation of *mail\_address* is implementation-defined.
- 29532 The *qalter* utility shall set the *Mail\_Users* attribute of the batch job to the value of  
29533 the *mail\_list* option-argument.
- 29534 **-N name** Redefine the name of the batch job.
- 29535 The *qalter* **-N** option shall accept a value for the *name* option-argument that is a  
29536 string of up to 15 alphanumeric characters in the portable character set where the  
29537 first character is alphabetic.
- 29538 The syntax of the *name* option-argument is unspecified.
- 29539 The *qalter* utility shall set the *Job\_Name* attribute of the batch job to the value of the  
29540 *name* option-argument.
- 29541 **-o path\_name** Redefine the path for the standard output of the batch job.
- 29542 The *qalter* utility shall accept a *path\_name* option-argument that conforms to the  
29543 syntax of the *path\_name* element defined in the System Interfaces volume of  
29544 IEEE Std 1003.1-200x, which can be preceded by a host name element of the form  
29545 *hostname*:
- 29546 If the *path\_name* option-argument constitutes an absolute pathname, the *qalter*  
29547 utility shall set the *Output\_Path* attribute of the batch job to the value of the  
29548 *path\_name* option-argument.
- 29549 If the *path\_name* option-argument constitutes a relative pathname and no host  
29550 name element is specified, the *qalter* utility shall set the *Output\_Path* attribute of the  
29551 batch job to the absolute pathname derived by expanding the *path\_name* option-

29552 argument relative to the current directory of the process that executes the *qalter*  
 29553 utility.

29554 If the *path\_name* option-argument constitutes a relative pathname and a host name  
 29555 element is specified, the *qalter* utility shall set the *Output\_Path* attribute of the batch  
 29556 job to option-argument without any expansion of the pathname.

29557 If the *path\_name* option-argument does not include a host name element, the *qalter*  
 29558 utility shall prefix the pathname in the *Output\_Path* attribute with *hostname:*, where  
 29559 *hostname* is the name of the host upon which the *qalter* utility is being executed.

29560 **-p *priority*** Redefine the priority of the batch job.

29561 The *qalter* utility shall accept a value for the priority option-argument that  
 29562 conforms to the syntax for signed decimal integers, and which is not less than  
 29563 -1 024 and not greater than 1 023.

29564 The *qalter* utility shall set the *Priority* attribute of the batch job to the value of the  
 29565 *priority* option-argument.

29566 **-r *y* | *n*** Redefine whether the batch job is rerunable.

29567 If the value of the option-argument is '*y*', the *qalter* utility shall set the *Rerunable*  
 29568 attribute of the batch job to TRUE.

29569 If the value of the option-argument is '*n*', the *qalter* utility shall set the *Rerunable*  
 29570 attribute of the batch job to FALSE.

29571 The *qalter* utility shall consider it an error if any character other than '*y*' or '*n*' is  
 29572 specified in the option-argument.

29573 **-S *path\_name\_list***

29574 Redefine the shell that interprets the script at the destination system.

29575 The *qalter* utility shall accept a *path\_name\_list* option-argument that conforms to  
 29576 the following syntax:

29577 `pathname[@host][,pathname[@host],...]`

29578 The *qalter* utility shall accept only one pathname that is missing a corresponding  
 29579 host name. The *qalter* utility shall allow only one pathname per named host.

29580 The *qalter* utility shall add a value to the *Shell\_Path\_List* attribute of the batch job  
 29581 for each entry in the *path\_name\_list* option-argument. See Section 3.3.3 (on page  
 29582 2325) for a means of removing *keyword=value* (and *value@keyword*) pairs and other  
 29583 general rules for list-oriented batch job attributes.

29584 **-u *user\_list*** Redefine the user name under which the batch job is to run at the destination  
 29585 system.

29586 The *qalter* utility shall accept a *user\_list* option-argument that conforms to the  
 29587 following syntax:

29588 `username[@host][,username[@host],...]`

29589 The *qalter* utility shall accept only one user name that is missing a corresponding  
 29590 host name. The *qalter* utility shall accept only one user name per named host.

29591 The *qalter* utility shall add a value to the *User\_List* attribute of the batch job for each  
 29592 entry in the *user\_list* option-argument. See Section 3.3.3 (on page 2325) for a means  
 29593 of removing *keyword=value* (and *value@keyword*) pairs and other general rules for  
 29594 list-oriented batch job attributes.

29595 **OPERANDS**

29596 The *qalter* utility shall accept one or more operands that conform to the syntax for a batch  
 29597 *job\_identifier* (see Section 3.3.1 (on page 2324)).

29598 **STDIN**

29599 Not used.

29600 **INPUT FILES**

29601 None.

29602 **ENVIRONMENT VARIABLES**

29603 The following environment variables shall affect the execution of *qalter*:

29604 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 29605 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
 29606 Internationalization Variables for the precedence of internationalization variables  
 29607 used to determine the values of locale categories.)

29608 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 29609 internationalization variables.

29610 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 29611 characters (for example, single-byte as opposed to multi-byte characters in  
 29612 arguments).

29613 *LC\_MESSAGES*

29614 Determine the locale that should be used to affect the format and contents of  
 29615 diagnostic messages written to standard error.

29616 *LOGNAME* Determine the login name of the user.

29617 *TZ* Determine the timezone used to interpret the *date-time* option-argument. If *TZ* is  
 29618 unset or null, an unspecified default timezone shall be used.

29619 **ASYNCHRONOUS EVENTS**

29620 Default.

29621 **STDOUT**

29622 None.

29623 **STDERR**

29624 The standard error shall be used only for diagnostic messages. |

29625 **OUTPUT FILES**

29626 None.

29627 **EXTENDED DESCRIPTION**

29628 None.

29629 **EXIT STATUS**

29630 The following exit values shall be returned:

29631 0 Successful completion.

29632 >0 An error occurred.

29633 **CONSEQUENCES OF ERRORS**

29634 In addition to the default behavior, the *qalter* utility shall not be required to write a diagnostic  
 29635 message to standard error when the error reply received from a batch server indicates that the  
 29636 batch *job\_identifier* does not exist on the server. Whether or not the *qalter* utility attempts to  
 29637 locate the batch job on other batch servers is implementation-defined.

## 29638 APPLICATION USAGE

29639 None.

## 29640 EXAMPLES

29641 None.

## 29642 RATIONALE

29643 The *qalter* utility allows users to change the attributes of a batch job.

29644 As a means of altering a queued job, the *qalter* utility is superior to deleting and requeuing the  
29645 batch job insofar as an altered job retains its place in the queue with some traditional selection  
29646 algorithms. In addition, the *qalter* utility is both shorter and simpler than a sequence of *qdel* and  
29647 *qsub* utilities.

29648 The result of an attempt on the part of a user to alter a batch job in a RUNNING state is  
29649 implementation-defined because a batch job in the RUNNING state will already have opened its  
29650 output files and otherwise performed any actions indicated by the options in effect at the time  
29651 the batch job began execution.

29652 The options processed by the *qalter* utility are identical to those of the *qsub* utility, with a few  
29653 exceptions: **-V**, **-v**, and **-q**. The **-V** and **-v** are inappropriate for the *qalter* utility, since they  
29654 capture potentially transient environment information from the submitting process. The **-q**  
29655 option would specify a new queue, which would largely negate the previously stated advantage  
29656 of using *qalter*; furthermore, the *qmove* utility provides a superior means of moving jobs.

29657 Each of the following paragraphs provides the rationale for a *qalter* option.29658 Additional rationale concerning these options can be found in the rationale for the *qsub* utility.

29659 The **-a** option allows users to alter the date and time at which a batch job becomes eligible to  
29660 run.

29661 The **-A** option allows users to change the account that will be charged for the resources  
29662 consumed by the batch job. Support for the **-A** option is mandatory for conforming  
29663 implementations of *qalter*, even though support of accounting is optional for servers. Whether or  
29664 not to support accounting is left to the implementor of the server, but mandatory support of the  
29665 **-A** option assures users of a consistent interface and allows them to control accounting on  
29666 servers that support accounting.

29667 The **-c** option allows users to alter the checkpointing interval of a batch job. A checkpointing  
29668 system, which is not defined by IEEE Std 1003.1-200x, allows recovery of a batch job at the most  
29669 recent checkpoint in the event of a crash. Checkpointing is typically used for jobs that consume  
29670 expensive computing time or must meet a critical schedule. Users should be allowed to make  
29671 the tradeoff between the overhead of checkpointing and the risk to the timely completion of the  
29672 batch job; therefore, this volume of IEEE Std 1003.1-200x provides the checkpointing interval  
29673 option. Support for checkpointing is optional for servers.

29674 The **-e** option allows users to alter the name and location of the standard error stream written by  
29675 a batch job. However, the path of the standard error stream is meaningless if the value of the  
29676 *Join\_Path* attribute of the batch job is TRUE.

29677 The **-h** option allows users to set the hold type in the *Hold\_Types* attribute of a batch job. The  
29678 *qhold* and *qrls* utilities add or remove hold types to the *Hold\_Types* attribute, respectively. The **-h**  
29679 option has been modified to allow for implementation-defined hold types.

29680 The **-j** option allows users to alter the decision to join (merge) the standard error stream of the  
29681 batch job with the standard output stream of the batch job.

- 29682 The **-l** option allows users to change the resource limits imposed on a batch job.
- 29683 The **-m** option allows users to modify the list of points in the life of a batch job at which the  
29684 designated users will receive mail notification.
- 29685 The **-M** option allows users to alter the list of users who will receive notification about events in  
29686 the life of a batch job.
- 29687 The **-N** option allows users to change the name of a batch job.
- 29688 The **-o** option allows users to alter the name and path to which the standard output stream of  
29689 the batch job will be written.
- 29690 The **-P** option allows users to modify the priority of a batch job. Support for priority is optional  
29691 for batch servers.
- 29692 The **-r** option allows users to alter the rerunability status of a batch job.
- 29693 The **-S** option allows users to change the name and location of the shell image that will be  
29694 invoked to interpret the script of the batch job. This option has been modified to allow a list of  
29695 shell name and locations associated with different host.
- 29696 The **-u** option allows users to change the user identifier under which the batch job will execute.
- 29697 The *job\_identifier* operand syntax is provided so that the user can differentiate between the  
29698 originating and destination (or executing) batch server. These may or may not be the same. The  
29699 *.server\_name* portion identifies the originating batch server, while the *@server* portion identifies  
29700 the destination batch server.
- 29701 Historically, the *qalter* utility has been a component of the Network Queuing System (NQS), the  
29702 existing practice from which this utility has been derived.
- 29703 **FUTURE DIRECTIONS**
- 29704 None.
- 29705 **SEE ALSO**
- 29706 *qdel, qhold, qmove, qrls, qsub, touch*, Chapter 3 (on page 2303)
- 29707 **CHANGE HISTORY**
- 29708 Derived from IEEE Std 1003.2d-1994.
- 29709 **Issue 6**
- 29710 The *TZ* entry is added to the ENVIRONMENT VARIABLES section.
- 29711 IEEE PASC Interpretation 1003.2 #182 is applied, clarifying the description of the **-a** option.

29712 **NAME**

29713 qdel — delete batch jobs

29714 **SYNOPSIS**29715 BE qdel *job\_identifier* ...

29716

29717 **DESCRIPTION**29718 A batch job is deleted by sending a request to the batch server that manages the batch job. A  
29719 batch job that has been deleted is no longer subject to management by batch services.29720 The *qdel* utility is a user-accessible client of batch services that requests the deletion of one or  
29721 more batch jobs.29722 The *qdel* utility shall request a batch server to delete those batch jobs for which a batch  
29723 *job\_identifier* is presented to the utility.29724 The *qdel* utility shall delete batch jobs in the order in which their batch *job\_identifiers* are  
29725 presented to the utility.29726 If the *qdel* utility fails to process any batch *job\_identifier* successfully, the utility shall proceed to  
29727 process the remaining batch *job\_identifiers*, if any.29728 The *qdel* utility shall delete each batch job by sending a *Delete Job Request* to the batch server that  
29729 manages the batch job.29730 The *qdel* utility shall not exit until the batch job corresponding to each successfully processed  
29731 batch *job\_identifier* has been deleted.29732 **OPTIONS**

29733 None.

29734 **OPERANDS**29735 The *qdel* utility shall accept one or more operands that conform to the syntax for a batch  
29736 *job\_identifier* (see Section 3.3.1 (on page 2324)).29737 **STDIN**

29738 Not used.

29739 **INPUT FILES**

29740 None.

29741 **ENVIRONMENT VARIABLES**29742 The following environment variables shall affect the execution of *qdel*:29743 *LANG* Provide a default value for the internationalization variables that are unset or null.  
29744 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
29745 Internationalization Variables for the precedence of internationalization variables  
29746 used to determine the values of locale categories.)29747 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
29748 internationalization variables.29749 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
29750 characters (for example, single-byte as opposed to multi-byte characters in  
29751 arguments).29752 *LC\_MESSAGES*29753 Determine the locale that should be used to affect the format and contents of  
29754 diagnostic messages written to standard error.

- 29755            *LOGNAME* Determine the login name of the user.
- 29756 **ASYNCHRONOUS EVENTS**
- 29757            Default.
- 29758 **STDOUT**
- 29759            An implementation of the *qdel* utility may write informative messages to standard output.
- 29760 **STDERR**
- 29761            The standard error shall be used only for diagnostic messages.
- 29762 **OUTPUT FILES**
- 29763            None.
- 29764 **EXTENDED DESCRIPTION**
- 29765            None.
- 29766 **EXIT STATUS**
- 29767            The following exit values shall be returned:
- 29768            0 Successful completion.
- 29769            >0 An error occurred.
- 29770 **CONSEQUENCES OF ERRORS**
- 29771            In addition to the default behavior, the *qdel* utility shall not be required to write a diagnostic message to standard error when the error reply received from a batch server indicates that the batch *job\_identifier* does not exist on the server. Whether or not the *qdel* utility waits to output the diagnostic message while attempting to locate the job on other servers is implementation-defined.
- 29772
- 29773
- 29774
- 29775
- 29776 **APPLICATION USAGE**
- 29777            None.
- 29778 **EXAMPLES**
- 29779            None.
- 29780 **RATIONALE**
- 29781            The *qdel* utility allows users and administrators to delete jobs.
- 29782            The *qdel* utility provides functionality that is not otherwise available. For example, the *kill* utility of the operating system does not suffice. First, to use the *kill* utility, the user might have to log in on a remote node, because the *kill* utility does not operate across the network. Second, unlike *qdel*, *kill* cannot remove jobs from queues. Lastly, the arguments of the *qdel* utility are job identifiers rather than process identifiers, and so this utility can be passed the output of the *qselect* utility, thus providing users with a means of deleting a list of jobs.
- 29783
- 29784
- 29785
- 29786
- 29787
- 29788            Because a set of jobs can be selected using the *qselect* utility, the *qdel* utility has not been complicated with options that provide for selection of jobs. Instead, the batch jobs to be deleted are identified individually by their job identifiers.
- 29789
- 29790
- 29791            Historically, the *qdel* utility has been a component of NQS, the existing practice on which it is based. However, the *qdel* utility defined in this volume of IEEE Std 1003.1-200x does not provide an option for specifying a signal number to send to the batch job prior to the killing of the process; that capability has been subsumed by the *qsig* utility.
- 29792
- 29793
- 29794
- 29795            A discussion was held about the delays of networking and the possibility that the batch server may never respond, due to a down router, down batch server, or other network mishap. The DESCRIPTION records this under the words “fails to process any job identifier”. In the broad sense, the network problem is also an error, which causes the failure to process the batch job
- 29796
- 29797
- 29798

29799 identifier.

29800 **FUTURE DIRECTIONS**

29801 None.

29802 **SEE ALSO**

29803 *kill, qselect, qsig*, Chapter 3 (on page 2303)

29804 **CHANGE HISTORY**

29805 Derived from IEEE Std 1003.2d-1994.

29806 **Issue 6**

29807 The *LC\_TIME* and *TZ* entries are removed from the ENVIRONMENT VARIABLES section.

## 29808 NAME

29809 qhold — hold batch jobs

## 29810 SYNOPSIS

29811 BE qhold [-h *hold\_list*] *job\_identifier* ...

29812

## 29813 DESCRIPTION

29814 A hold is placed on a batch job by a request to the batch server that manages the batch job. A  
 29815 batch job that has one or more holds is not eligible for execution. The *qhold* utility is a user-  
 29816 accessible client of batch services that requests one or more types of hold to be placed on one or  
 29817 more batch jobs.

29818 The *qhold* utility shall place holds on those batch jobs for which a batch *job\_identifier* is presented  
 29819 to the utility.

29820 The *qhold* utility shall place holds on batch jobs in the order in which their batch *job\_identifiers*  
 29821 are presented to the utility. If the *qhold* utility fails to process any batch *job\_identifier* successfully,  
 29822 the utility shall proceed to process the remaining batch *job\_identifiers*, if any.

29823 The *qhold* utility shall place holds on each batch job by sending a *Hold Job Request* to the batch  
 29824 server that manages the batch job.

29825 The *qhold* utility shall not exit until holds have been placed on the batch job corresponding to  
 29826 each successfully processed batch *job\_identifier*.

## 29827 OPTIONS

29828 The *qhold* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section  
 29829 12.2, Utility Syntax Guidelines.

29830 The following option shall be supported by the implementation:

29831 **-h *hold\_list*** Define the types of holds to be placed on the batch job.

29832 The *qhold* **-h** option shall accept a value for the *hold\_list* option-argument that is a  
 29833 string of alphanumeric characters in the portable character set (see the Base  
 29834 Definitions volume of IEEE Std 1003.1-200x, Section 6.1, Portable Character Set).

29835 The *qhold* utility shall accept a value for the *hold\_list* option-argument that is a  
 29836 string of one or more of the characters 'u', 's', or 'o', or the single character  
 29837 'n'.

29838 For each unique character in the *hold\_list* option-argument, the *qhold* utility shall  
 29839 add a value to the *Hold\_Types* attribute of the batch job as follows, each  
 29840 representing a different hold type:

29841 u USER

29842 s SYSTEM

29843 o OPERATOR

29844 If any of these characters are duplicated in the *hold\_list* option-argument, the  
 29845 duplicates shall be ignored.

29846 An existing *Hold\_Types* attribute can be cleared by the following hold type:

29847 n NO\_HOLD

29848 The *qhold* utility shall consider it an error if any hold type other than 'n' is  
 29849 combined with hold type 'n'.

29850 Strictly conforming applications shall not repeat any of the characters 'u', 's',  
29851 'o', or 'n' within the *hold\_list* option-argument. The *qhold* utility shall permit the  
29852 repetition of characters, but shall not assign additional meaning to the repeated  
29853 characters.

29854 An implementation may define other hold types. The conformance document for  
29855 an implementation shall describe any additional hold types, how they are  
29856 specified, their internal behavior, and how they affect the behavior of the utility.

29857 If the *-h* option is not presented to the *qhold* utility, the implementation shall set  
29858 the *Hold\_Types* attribute to *USER*.

#### 29859 OPERANDS

29860 The *qhold* utility shall accept one or more operands that conform to the syntax for a batch  
29861 *job\_identifier* (see Section 3.3.1 (on page 2324)).

#### 29862 STDIN

29863 Not used.

#### 29864 INPUT FILES

29865 None.

#### 29866 ENVIRONMENT VARIABLES

29867 The following environment variables shall affect the execution of *qhold*:

29868 *LANG* Provide a default value for the internationalization variables that are unset or null.  
29869 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
29870 Internationalization Variables for the precedence of internationalization variables  
29871 used to determine the values of locale categories.)

29872 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
29873 internationalization variables.

29874 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
29875 characters (for example, single-byte as opposed to multi-byte characters in  
29876 arguments).

#### 29877 LC\_MESSAGES

29878 Determine the locale that should be used to affect the format and contents of  
29879 diagnostic messages written to standard error.

29880 *LOGNAME* Determine the login name of the user.

#### 29881 ASYNCHRONOUS EVENTS

29882 Default.

#### 29883 STDOUT

29884 None.

#### 29885 STDERR

29886 The standard error shall be used only for diagnostic messages. |

#### 29887 OUTPUT FILES

29888 None.

#### 29889 EXTENDED DESCRIPTION

29890 None.

29891 **EXIT STATUS**

29892 The following exit values shall be returned:

29893 0 Successful completion.

29894 >0 An error occurred.

29895 **CONSEQUENCES OF ERRORS**

29896 In addition to the default behavior, the *qhold* utility shall not be required to write a diagnostic message to standard error when the error reply received from a batch server indicates that the batch *job\_identifier* does not exist on the server. Whether or not the *qhold* utility waits to output the diagnostic message while attempting to locate the job on other servers is implementation-defined.

29901 **APPLICATION USAGE**

29902 None.

29903 **EXAMPLES**

29904 None.

29905 **RATIONALE**

29906 The *qhold* utility allows users to place a hold on one or more jobs. A hold makes a batch job ineligible for execution.

29908 The *qhold* utility has options that allow the user to specify the type of hold. Should the user wish to place a hold on a set of jobs that meet a selection criteria, such a list of jobs can be acquired using the *qselect* utility.

29911 The *-h* option allows the user to specify the type of hold that is to be placed on the job. This option allows for USER, SYSTEM, OPERATOR, and implementation-defined hold types. The USER and OPERATOR holds are distinct. The batch server that manages the batch job will verify that the user is authorized to set the specified hold for the batch job.

29915 Mail is not required on hold because the administrator has the tools and libraries to build this option if he or she wishes.

29917 Historically, the *qhold* utility has been a part of some existing batch systems, although it has not traditionally been a part of the NQS.

29919 **FUTURE DIRECTIONS**

29920 None.

29921 **SEE ALSO**

29922 *qselect*, Chapter 3 (on page 2303)

29923 **CHANGE HISTORY**

29924 Derived from IEEE Std 1003.2d-1994.

29925 **Issue 6**

29926 The *LC\_TIME* and *TZ* entries are removed from the ENVIRONMENT VARIABLES section.

29927 **NAME**

29928 qmove — move batch jobs

29929 **SYNOPSIS**29930 BE qmove *destination job\_identifier* ...

29931

29932 **DESCRIPTION**

29933 To move a batch job is to remove the batch job from the batch queue in which it resides and  
 29934 instantiate the batch job in another batch queue. A batch job is moved by a request to the batch  
 29935 server that manages the batch job. The *qmove* utility is a user-accessible batch client that requests  
 29936 the movement of one or more batch jobs.

29937 The *qmove* utility shall move those batch jobs, and only those batch jobs, for which a batch  
 29938 *job\_identifier* is presented to the utility.

29939 The *qmove* utility shall move batch jobs in the order in which the corresponding batch  
 29940 *job\_identifiers* are presented to the utility.

29941 If the *qmove* utility fails to process a batch *job\_identifier* successfully, the utility shall proceed to  
 29942 process the remaining batch *job\_identifiers*, if any.

29943 The *qmove* utility shall move batch jobs by sending a *Move Job Request* to the batch server that  
 29944 manages each batch job. The *qmove* utility shall not exit before the batch jobs corresponding to all  
 29945 successfully processed batch *job\_identifiers* have been moved.

29946 **OPTIONS**

29947 None.

29948 **OPERANDS**

29949 The *qmove* utility shall accept one operand that conforms to the syntax for a *destination* (see  
 29950 Section 3.3.2 (on page 2325)).

29951 The *qmove* utility shall accept one or more operands that conform to the syntax for a batch  
 29952 *job\_identifier* (see Section 3.3.1 (on page 2324)).

29953 **STDIN**

29954 Not used.

29955 **INPUT FILES**

29956 None.

29957 **ENVIRONMENT VARIABLES**29958 The following environment variables shall affect the execution of *qmove*:

29959 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 29960 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
 29961 Internationalization Variables for the precedence of internationalization variables  
 29962 used to determine the values of locale categories.)

29963 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 29964 internationalization variables.

29965 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 29966 characters (for example, single-byte as opposed to multi-byte characters in  
 29967 arguments).

29968 *LC\_MESSAGES*

29969 Determine the locale that should be used to affect the format and contents of  
 29970 diagnostic messages written to standard error.

- 29971 **LOGNAME** Determine the login name of the user.
- 29972 **ASYNCHRONOUS EVENTS**
- 29973 Default.
- 29974 **STDOUT**
- 29975 None.
- 29976 **STDERR**
- 29977 The standard error shall be used only for diagnostic messages.
- 29978 **OUTPUT FILES**
- 29979 None.
- 29980 **EXTENDED DESCRIPTION**
- 29981 None.
- 29982 **EXIT STATUS**
- 29983 The following exit values shall be returned:
- 29984 0 Successful completion.
- 29985 >0 An error occurred.
- 29986 **CONSEQUENCES OF ERRORS**
- 29987 In addition to the default behavior, the *qmove* utility shall not be required to write a diagnostic message to standard error when the error reply received from a batch server indicates that the batch *job\_identifier* does not exist on the server. Whether or not the *qmove* utility waits to output the diagnostic message while attempting to locate the job on other servers is implementation-defined.
- 29992 **APPLICATION USAGE**
- 29993 None.
- 29994 **EXAMPLES**
- 29995 None.
- 29996 **RATIONALE**
- 29997 The *qmove* utility allows users to move jobs between queues.
- 29998 The alternative to using the *qmove* utility—deleting the batch job and requeuing it—entails considerably more typing.
- 30000 Since the means of selecting jobs based on attributes has been encapsulated in the *qselect* utility, the only option of the *qmove* utility concerns authorization. The **-u** option provides the user with the convenience of changing the user identifier under which the batch job will execute.
- 30001 Minimalism and consistency has taken precedence over convenience; the **-u** option has been deleted because the equivalent capability exists with the **-u** option of the *qalter* utility.
- 30002
- 30003
- 30004
- 30005 **FUTURE DIRECTIONS**
- 30006 None.
- 30007 **SEE ALSO**
- 30008 *qalter*, *qselect*, Chapter 3 (on page 2303)
- 30009 **CHANGE HISTORY**
- 30010 Derived from IEEE Std 1003.2d-1994.

30011 **Issue 6**

30012

The *LC\_TIME* and *TZ* entries are removed from the ENVIRONMENT VARIABLES section.

## 30013 NAME

30014 qmsg — send message to batch jobs

## 30015 SYNOPSIS

30016 BE qmsg [-E][-O] *message\_string* *job\_identifier* ...

30017

## 30018 DESCRIPTION

30019 To send a message to a batch job is to request that a server write a message string into one or  
 30020 more output files of the batch job. A message is sent to a batch job by a request to the batch  
 30021 server that manages the batch job. The *qmsg* utility is a user-accessible batch client that requests  
 30022 the sending of messages to one or more batch jobs.

30023 The *qmsg* utility shall write messages into the files of batch jobs by sending a *Job Message Request*  
 30024 to the batch server that manages the batch job. The *qmsg* utility shall not directly write the  
 30025 message into the files of the batch job.

30026 The *qmsg* utility shall send a *Job Message Request* for those batch jobs, and only those batch jobs,  
 30027 for which a batch *job\_identifier* is presented to the utility.

30028 The *qmsg* utility shall send *Job Message Requests* for batch jobs in the order in which their batch  
 30029 *job\_identifiers* are presented to the utility.

30030 If the *qmsg* utility fails to process any batch *job\_identifier* successfully, the utility shall proceed to  
 30031 process the remaining batch *job\_identifiers*, if any.

30032 The *qmsg* utility shall not exit before a *Job Message Request* has been sent to the server that  
 30033 manages the batch job that corresponds to each successfully processed batch *job\_identifier*.

## 30034 OPTIONS

30035 The *qmsg* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section  
 30036 12.2, Utility Syntax Guidelines.

30037 The following options shall be supported by the implementation:

30038 **-E** Specify that the message is written to the standard error of each batch job.

30039 The *qmsg* utility shall write the message into the standard error of the batch job.

30040 **-O** Specify that the message is written to the standard output of each batch job.

30041 The *qmsg* utility shall write the message into the standard output of the batch job.

30042 If neither the **-O** nor the **-E** option is presented to the *qmsg* utility, the utility shall write the  
 30043 message into an implementation-defined file. The conformance document for the  
 30044 implementation shall describe the name and location of the implementation-defined file. If both  
 30045 the **-O** and the **-E** options are presented to the *qmsg* utility, then the utility shall write the  
 30046 messages to both standard output and standard error.

## 30047 OPERANDS

30048 The *qmsg* utility shall accept a minimum of two operands, *message\_string* and one or more batch  
 30049 *job\_identifiers*.

30050 The *message\_string* operand shall be the string to be written to one or more output files of the  
 30051 batch job followed by a <newline>. If the string contains <blank>s, then the application shall  
 30052 ensure that the string is quoted. The *message\_string* shall be encoded in the portable character set  
 30053 (see the Base Definitions volume of IEEE Std 1003.1-200x, Section 6.1, Portable Character Set).

30054 All remaining operands are batch *job\_identifiers* that conform to the syntax for a batch  
 30055 *job\_identifier* (see Section 3.3.1 (on page 2324)).

30056 **STDIN**

30057 Not used.

30058 **INPUT FILES**

30059 None.

30060 **ENVIRONMENT VARIABLES**30061 The following environment variables shall affect the execution of *qmsg*:

30062 *LANG* Provide a default value for the internationalization variables that are unset or null.  
30063 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
30064 Internationalization Variables for the precedence of internationalization variables  
30065 used to determine the values of locale categories.)

30066 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
30067 internationalization variables.

30068 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
30069 characters (for example, single-byte as opposed to multi-byte characters in  
30070 arguments).

30071 *LC\_MESSAGES*

30072 Determine the locale that should be used to affect the format and contents of  
30073 diagnostic messages written to standard error.

30074 *LOGNAME* Determine the login name of the user.

30075 **ASYNCHRONOUS EVENTS**

30076 Default.

30077 **STDOUT**

30078 None.

30079 **STDERR**

30080 The standard error shall be used only for diagnostic messages. |

30081 **OUTPUT FILES**

30082 None.

30083 **EXTENDED DESCRIPTION**

30084 None.

30085 **EXIT STATUS**

30086 The following exit values shall be returned:

30087 0 Successful completion.

30088 &gt;0 An error occurred.

30089 **CONSEQUENCES OF ERRORS**

30090 In addition to the default behavior, the *qmsg* utility shall not be required to write a diagnostic  
30091 message to standard error when the error reply received from a batch server indicates that the  
30092 batch *job\_identifier* does not exist on the server. Whether or not the *qmsg* utility waits to output  
30093 the diagnostic message while attempting to locate the job on other servers is implementation-  
30094 defined.

30095 **APPLICATION USAGE**

30096 None.

30097 **EXAMPLES**

30098 None.

30099 **RATIONALE**

30100 The *qmsg* utility allows users to write messages into the output files of running jobs. Users,  
30101 including operators and administrators, have a number of occasions when they want to place  
30102 messages in the output files of a batch job. For example, if a disk that is being used by a batch job  
30103 is showing errors, the operator might note this in the standard error stream of the batch job.

30104 The options of the *qmsg* utility provide users with the means of placing the message in the  
30105 output stream of their choice. The default output stream for the message—if the user does not  
30106 designate an output stream—is implementation-defined, since many implementations will  
30107 provide, as an extension to this volume of IEEE Std 1003.1-200x, a log file that shows the history  
30108 of utility execution.

30109 If users wish to send a message to a set of jobs that meet a selection criteria, the *qselect* utility can  
30110 be used to acquire the appropriate list of job identifiers.

30111 The **-E** option allows users to place the message in the standard error stream of the batch job.

30112 The **-O** option allows users to place the message in the standard output stream of the batch job.

30113 Historically, the *qmsg* utility is an existing practice in the offerings of one or more implementors  
30114 of an NQS-derived batch system. The utility has been found to be useful enough that it deserves  
30115 to be included in this volume of IEEE Std 1003.1-200x.

30116 **FUTURE DIRECTIONS**

30117 None.

30118 **SEE ALSO**30119 *qselect*, Chapter 3 (on page 2303)30120 **CHANGE HISTORY**

30121 Derived from IEEE Std 1003.2d-1994.

30122 **Issue 6**30123 The *LC\_TIME* and *TZ* entries are removed from the ENVIRONMENT VARIABLES section.

30124 **NAME**

30125 qrerun — rerun batch jobs

30126 **SYNOPSIS**30127 BE qrerun *job\_identifier* ...

30128

30129 **DESCRIPTION**

30130 To rerun a batch job is to terminate the session leader of the batch job, delete any associated  
 30131 checkpoint files, and return the batch job to the batch queued state. A batch job is rerun by a  
 30132 request to the batch server that manages the batch job. The *qrerun* utility is a user-accessible  
 30133 batch client that requests the rerunning of one or more batch jobs.

30134 The *qrerun* utility shall rerun those batch jobs for which a batch *job\_identifier* is presented to the  
 30135 utility.

30136 The *qrerun* utility shall rerun batch jobs in the order in which their batch *job\_identifiers* are  
 30137 presented to the utility.

30138 If the *qrerun* utility fails to process any batch *job\_identifier* successfully, the utility shall proceed  
 30139 to process the remaining batch *job\_identifiers*, if any.

30140 The *qrerun* utility shall rerun batch jobs by sending a *Rerun Job Request* to the batch server that  
 30141 manages each batch job.

30142 For each successfully processed batch *job\_identifier*, the *qrerun* utility shall have rerun the  
 30143 corresponding batch batch job at the time the utility exits.

30144 **OPTIONS**

30145 None.

30146 **OPERANDS**

30147 The *qrerun* utility shall accept one or more operands that conform to the syntax for a batch  
 30148 *job\_identifier* (see Section 3.3.1 (on page 2324)).

30149 **STDIN**

30150 Not used.

30151 **INPUT FILES**

30152 None.

30153 **ENVIRONMENT VARIABLES**30154 The following environment variables shall affect the execution of *qrerun*:

30155 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 30156 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
 30157 Internationalization Variables for the precedence of internationalization variables  
 30158 used to determine the values of locale categories.)

30159 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 30160 internationalization variables.

30161 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 30162 characters (for example, single-byte as opposed to multi-byte characters in  
 30163 arguments).

30164 *LC\_MESSAGES*

30165 Determine the locale that should be used to affect the format and contents of  
 30166 diagnostic messages written to standard error.

- 30167            *LOGNAME* Determine the login name of the user.
- 30168 **ASYNCHRONOUS EVENTS**
- 30169            Default.
- 30170 **STDOUT**
- 30171            None.
- 30172 **STDERR**
- 30173            The standard error shall be used only for diagnostic messages.
- 30174 **OUTPUT FILES**
- 30175            None.
- 30176 **EXTENDED DESCRIPTION**
- 30177            None.
- 30178 **EXIT STATUS**
- 30179            The following exit values shall be returned:
- 30180            0 Successful completion.
- 30181            >0 An error occurred.
- 30182 **CONSEQUENCES OF ERRORS**
- 30183            In addition to the default behavior, the *qrerun* utility shall not be required to write a diagnostic message to standard error when the error reply received from a batch server indicates that the batch *job\_identifier* does not exist on the server. Whether or not the *qrerun* utility waits to output the diagnostic message while attempting to locate the job on other servers is implementation-defined.
- 30188 **APPLICATION USAGE**
- 30189            None.
- 30190 **EXAMPLES**
- 30191            None.
- 30192 **RATIONALE**
- 30193            The *qrerun* utility allows users to cause jobs in the running state to exit and rerun.
- 30194            The *qrerun* utility is a new utility, *vis-a-vis* existing practice, that has been defined in this volume of IEEE Std 1003.1-200x to correct user-perceived deficiencies in the existing practice.
- 30196 **FUTURE DIRECTIONS**
- 30197            None.
- 30198 **SEE ALSO**
- 30199            Chapter 3 (on page 2303)
- 30200 **CHANGE HISTORY**
- 30201            Derived from IEEE Std 1003.2d-1994.
- 30202 **Issue 6**
- 30203            The *LC\_TIME* and *TZ* entries are removed from the ENVIRONMENT VARIABLES section.

## 30204 NAME

30205 qrls — release batch jobs

## 30206 SYNOPSIS

30207 BE qrls [-h *hold\_list*] *job\_identifier* ...

30208

## 30209 DESCRIPTION

30210 A batch job might have one or more holds, which prevent the batch job from executing. A batch  
 30211 job from which all the holds have been removed becomes eligible for execution and is said to  
 30212 have been released. A batch job hold is removed by sending a request to the batch server that  
 30213 manages the batch job. The *qrls* utility is a user-accessible client of batch services that requests  
 30214 holds be removed from one or more batch jobs.

30215 The *qrls* utility shall remove one or more holds from those batch jobs for which a batch  
 30216 *job\_identifier* is presented to the utility.

30217 The *qrls* utility shall remove holds from batch jobs in the order in which their batch *job\_identifiers*  
 30218 are presented to the utility.

30219 If the *qrls* utility fails to process a batch *job\_identifier* successfully, the utility shall proceed to  
 30220 process the remaining batch *job\_identifiers*, if any.

30221 The *qrls* utility shall remove holds on each batch job by sending a *Release Job Request* to the batch  
 30222 server that manages the batch job.

30223 The *qrls* utility shall not exit until the holds have been removed from the batch job  
 30224 corresponding to each successfully processed batch *job\_identifier*.

## 30225 OPTIONS

30226 The *qrls* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section  
 30227 12.2, Utility Syntax Guidelines.

30228 The following option shall be supported by the implementation:

30229 **-h *hold\_list*** Define the types of holds to be removed from the batch job.

30230 The *qrls* **-h** option shall accept a value for the *hold\_list* option-argument that is a  
 30231 string of alphanumeric characters in the portable character set (see the Base  
 30232 Definitions volume of IEEE Std 1003.1-200x, Section 6.1, Portable Character Set).

30233 The *qrls* utility shall accept a value for the *hold\_list* option-argument that is a string  
 30234 of one or more of the characters 'u', 's', or 'o', or the single character 'n'.

30235 For each unique character in the *hold\_list* option-argument, the *qrls* utility shall add  
 30236 a value to the *Hold\_Types* attribute of the batch job as follows, each representing a  
 30237 different hold type:

30238 u USER

30239 s SYSTEM

30240 o OPERATOR

30241 If any of these characters are duplicated in the *hold\_list* option-argument, the  
 30242 duplicates shall be ignored.

30243 An existing *Hold\_Types* attribute can be cleared by the following hold type:

30244 n NO\_HOLD

- 30245 The *qrls* utility shall consider it an error if any hold type other than 'n' is  
30246 combined with hold type 'n'.
- 30247 Strictly conforming applications shall not repeat any of the characters 'u', 's',  
30248 'o', or 'n' within the *hold\_list* option-argument. The *qrls* utility shall permit the  
30249 repetition of characters, but shall not assign additional meaning to the repeated  
30250 characters.
- 30251 An implementation may define other hold types. The conformance document for  
30252 an implementation shall describe any additional hold types, how they are  
30253 specified, their internal behavior, and how they affect the behavior of the utility.
- 30254 If the **-h** option is not presented to the *qrls* utility, the implementation shall remove  
30255 the **USER** hold in the *Hold\_Types* attribute.
- 30256 **OPERANDS**
- 30257 The *qrls* utility shall accept one or more operands that conform to the syntax for a batch  
30258 *job\_identifier* (see Section 3.3.1 (on page 2324)).
- 30259 **STDIN**
- 30260 Not used.
- 30261 **INPUT FILES**
- 30262 None.
- 30263 **ENVIRONMENT VARIABLES**
- 30264 The following environment variables shall affect the execution of *qrls*:
- 30265 *LANG* Provide a default value for the internationalization variables that are unset or null.  
30266 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
30267 Internationalization Variables for the precedence of internationalization variables  
30268 used to determine the values of locale categories.)
- 30269 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
30270 internationalization variables.
- 30271 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
30272 characters (for example, single-byte as opposed to multi-byte characters in  
30273 arguments).
- 30274 *LC\_MESSAGES*
- 30275 Determine the locale that should be used to affect the format and contents of  
30276 diagnostic messages written to standard error.
- 30277 *LOGNAME* Determine the login name of the user.
- 30278 **ASYNCHRONOUS EVENTS**
- 30279 Default.
- 30280 **STDOUT**
- 30281 None.
- 30282 **STDERR**
- 30283 The standard error shall be used only for diagnostic messages. |
- 30284 **OUTPUT FILES**
- 30285 None.

30286 **EXTENDED DESCRIPTION**

30287 None.

30288 **EXIT STATUS**

30289 The following exit values shall be returned:

30290 0 Successful completion.

30291 &gt;0 An error occurred.

30292 **CONSEQUENCES OF ERRORS**

30293 In addition to the default behavior, the *qrls* utility shall not be required to write a diagnostic message to standard error when the error reply received from a batch server indicates that the batch *job\_identifier* does not exist on the server. Whether or not the *qrls* utility waits to output the diagnostic message while attempting to locate the job on other servers is implementation-defined.

30298 **APPLICATION USAGE**

30299 None.

30300 **EXAMPLES**

30301 None.

30302 **RATIONALE**30303 The *qrls* utility allows users, operators, and administrators to remove holds from jobs.

30304 The *qrls* utility does not support any job selection options or wildcard arguments. Users may acquire a list of jobs selected by attributes using the *qselect* utility. For example, a user could select all of their held jobs.

30307 The *-h* option allows the user to specify the type of hold that is to be removed. This option allows for USER, SYSTEM, OPERATOR, and implementation-defined hold types. The batch server that manages the batch job will verify whether the user is authorized to remove the specified hold for the batch job. If more than one type of hold has been placed on the batch job, a user may wish to remove only some of them.

30312 Mail is not required on release because the administrator has the tools and libraries to build this option if required.

30314 The *qrls* utility is a new utility *vis-a-vis* existing practice; it has been defined in this volume of IEEE Std 1003.1-200x as the natural complement to the *qhold* utility.

30316 **FUTURE DIRECTIONS**

30317 None.

30318 **SEE ALSO**30319 *qhold*, *qselect*, Chapter 3 (on page 2303)30320 **CHANGE HISTORY**

30321 Derived from IEEE Std 1003.2d-1994.

30322 **Issue 6**30323 The *LC\_TIME* and *TZ* entries are removed from the ENVIRONMENT VARIABLES section.

## 30324 NAME

30325 qselect — select batch jobs

## 30326 SYNOPSIS

```
30327 BE qselect [-a [op]date_time][-A account_string][-c [op]interval]
30328 [-h hold_list][-l resource_list][-N name][-p [op]priority]
30329 [-q destination][-r y|n][-s states][-u user_list]
30330
```

## 30331 DESCRIPTION

30332 To select a set of batch jobs is to return the batch *job\_identifiers* for each batch job that meets a list  
 30333 of selection criteria. A set of batch jobs is selected by a request to a batch server. The *qselect*  
 30334 utility is a user-accessible batch client that requests the selection of batch jobs.

30335 Upon successful completion, the *qselect* utility shall have returned a list of zero or more batch  
 30336 *job\_identifiers* that meet the criteria specified by the options and option-arguments presented to  
 30337 the utility.

30338 The *qselect* utility shall select batch jobs by sending a *Select Jobs Request* to a batch server. The  
 30339 *qselect* utility shall not exit until the server replies to each request generated.

30340 For each option presented to the *qselect* utility, the utility shall restrict the set of selected batch  
 30341 jobs as described in the OPTIONS section.

30342 The *qselect* utility shall not restrict selection of batch jobs except by authorization and as required  
 30343 by the options presented to the utility.

30344 When an option is specified with a mandatory or optional *op* component to the option-  
 30345 argument, then *op* shall specify a relation between the value of a certain batch job attribute and  
 30346 the *value* component of the option-argument. If an *op* is allowable on an option, then the  
 30347 description of the option letter indicates the *op* as either mandatory or optional. Acceptable  
 30348 strings for the *op* component, and the relation the string indicates, are shown in the following  
 30349 list:

30350 .eq. The value represented by the attribute of the batch job is equal to the value represented  
 30351 by the option-argument.

30352 .ge. The value represented by the attribute of the batch job is greater than or equal to the  
 30353 value represented by the option-argument.

30354 .gt. The value represented by the attribute of the batch job is greater than the value  
 30355 represented by the option-argument.

30356 .lt. The value represented by the attribute of the batch job is less than the value  
 30357 represented by the option-argument.

30358 .le. The value represented by the attribute of the batch job is less than or equal to the value  
 30359 represented by the option-argument.

30360 .ne. The value represented by the attribute of the batch job is not equal to the value  
 30361 represented by the option-argument.

## 30362 OPTIONS

30363 The *qselect* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section  
 30364 12.2, Utility Syntax Guidelines.

30365 The following options shall be supported by the implementation:

30366 **-a [op]date\_time**

30367 Restrict selection to a specific time, or a range of times.

- 30368 The *qselect* utility shall select only batch jobs for which the value of the  
30369 *Execution\_Time* attribute is related to the Epoch equivalent of the local time  
30370 expressed by the value of the *date\_time* component of the option-argument in the  
30371 manner indicated by the value of the *op* component of the option-argument.
- 30372 The *qselect* utility shall accept a *date\_time* component of the option-argument that  
30373 conforms to the syntax of the *time* operand of the *touch* utility.
- 30374 If the *op* component of the option-argument is not presented to the *qselect* utility,  
30375 the utility shall select batch jobs for which the *Execution\_Time* attribute is equal to  
30376 the *date\_time* component of the option-argument.
- 30377 When comparing times, the *qselect* utility shall use the following definitions for the  
30378 *op* component of the option-argument:
- 30379 .eq. The time represented by value of the *Execution\_Time* attribute of the batch  
30380 job is equal the time represented by the *date\_time* component of the  
30381 option-argument.
  - 30382 .ge. The time represented by value of the *Execution\_Time* attribute of the batch  
30383 job is after or equal to the time represented by the *date\_time* component of  
30384 the option-argument.
  - 30385 .gt. The time represented by value of the *Execution\_Time* attribute of the batch  
30386 job is after the time represented by the *date\_time* component of the  
30387 option-argument.
  - 30388 .lt. The time represented by value of the *Execution\_Time* attribute of the batch  
30389 job is before the time represented by the *date\_time* component of the  
30390 option-argument.
  - 30391 .le. The time represented by value of the *Execution\_Time* attribute of the batch  
30392 job is before or equal to the time represented by the *date\_time* component  
30393 of the option-argument.
  - 30394 .ne. The time represented by value of the *Execution\_Time* attribute of the batch  
30395 job is not equal to the time represented by the *date\_time* component of the  
30396 option-argument.
- 30397 The *qselect* utility shall accept the defined character strings for the *op* component of  
30398 the option-argument.
- 30399 **-A *account\_string***  
30400 Restrict selection to the batch jobs charging a specified account.
- 30401 The *qselect* utility shall select only batch jobs for which the value of the  
30402 *Account\_Name* attribute of the batch job matches the value of the *account\_string*  
30403 option-argument.
- 30404 The syntax of the *account\_string* option-argument is unspecified.
- 30405 **-c [*op*]*interval***  
30406 Restrict selection to batch jobs within a range of checkpoint intervals.
- 30407 The *qselect* utility shall select only batch jobs for which the value of the *Checkpoint*  
30408 attribute relates to the value of the *interval* component of the option-argument in  
30409 the manner indicated by the value of the *op* component of the option-argument.
- 30410 If the *op* component of the option-argument is omitted, the *qselect* utility shall  
30411 select batch jobs for which the value of the *Checkpoint* attribute is equal to the value

30412 of the *interval* component of the option-argument.

30413 When comparing checkpoint intervals, the *qselect* utility shall use the following  
30414 definitions for the *op* component of the option-argument:

30415 .eq. The value of the *Checkpoint* attribute of the batch job equals the value of  
30416 the *interval* component of the option-argument.

30417 .ge. The value of the *Checkpoint* attribute of the batch job is greater than or  
30418 equal to the value of the *interval* component option-argument.

30419 .gt. The value of the *Checkpoint* attribute of the batch job is greater than the  
30420 value of the *interval* component option-argument.

30421 .lt. The value of the *Checkpoint* attribute of the batch job is less than the value  
30422 of the *interval* component option-argument.

30423 .le. The value of the *Checkpoint* attribute of the batch job is less than or equal  
30424 to the value of the *interval* component option-argument.

30425 .ne. The value of the *Checkpoint* attribute of the batch job does not equal the  
30426 value of the *interval* component option-argument.

30427 The *qselect* utility shall accept the defined character strings for the *op* component of  
30428 the option-argument.

30429 The ordering relationship for the values of the interval option-argument is defined  
30430 to be:

30431 'n' .gt. 's' .gt. 'c=minutes' .ge. 'c'

30432 When comparing *Checkpoint* attributes with an interval having the value of the  
30433 single character 'u', only equality or inequality are valid comparisons.

30434 **-h hold\_list** Restrict selection to batch jobs that have a specific type of hold.

30435 The *qselect* utility shall select only batch jobs for which the value of the *Hold\_Types*  
30436 attribute matches the value of the *hold\_list* option-argument.

30437 The *qselect* **-h** option shall accept a value for the *hold\_list* option-argument that is a  
30438 string of alphanumeric characters in the portable character set (see the Base  
30439 Definitions volume of IEEE Std 1003.1-200x, Section 6.1, Portable Character Set).

30440 The *qselect* utility shall accept a value for the *hold\_list* option-argument that is a  
30441 string of one or more of the characters 'u', 's', or 'o', or the single character  
30442 'n'.

30443 Each unique character in the *hold\_list* option-argument of the *qselect* utility is  
30444 defined as follows, each representing a different hold type:

30445 u USER

30446 s SYSTEM

30447 o OPERATOR

30448 If any of these characters are duplicated in the *hold\_list* option-argument, the  
30449 duplicates shall be ignored.

30450 The *qselect* utility shall consider it an error if any hold type other than 'n' is  
30451 combined with hold type 'n'.

30452 Strictly conforming applications shall not repeat any of the characters 'u', 's',  
 30453 'o', or 'n' within the *hold\_list* option-argument. The *qselect* utility shall permit  
 30454 the repetition of characters, but shall not assign additional meaning to the repeated  
 30455 characters.

30456 An implementation may define other hold types. The conformance document for  
 30457 an implementation shall describe any additional hold types, how they are  
 30458 specified, their internal behavior, and how they affect the behavior of the utility.

30459 **-I *resource\_list***  
 30460 Restrict selection to batch jobs with specified resource limits and attributes.

30461 The *qselect* utility shall accept a *resource\_list* option-argument with the following  
 30462 syntax:

30463 *resource\_name op value [ , , resource\_name op value , , ... ]*

30464 When comparing resource values, the *qselect* utility shall use the following  
 30465 definitions for the *op* component of the option-argument:

30466 *.eq.* The value of the resource of the same name in the *Resource\_List* attribute  
 30467 of the batch job equals the value of the value component of the option-  
 30468 argument.

30469 *.ge.* The value of the resource of the same name in the *Resource\_List* attribute  
 30470 of the batch job is greater than or equal to the value of the *value*  
 30471 component of the option-argument.

30472 *.gt.* The value of the resource of the same name in the *Resource\_List* attribute  
 30473 of the batch job is greater than the value of the value component of the  
 30474 option-argument.

30475 *.lt.* The value of the resource of the same name in the *Resource\_List* attribute  
 30476 of the batch job is less than the value of the value component of the  
 30477 option-argument.

30478 *.ne.* The value of the resource of the same name in the *Resource\_List* attribute  
 30479 of the batch job does not equal the value of the value component of the  
 30480 option-argument.

30481 *.le.* The value of the resource of the same name in the *Resource\_List* attribute  
 30482 of the batch job is less than or equal to the value of the *value* component  
 30483 of the option-argument.

30484 When comparing the limit of a *Resource\_List* attribute with the *value* component of  
 30485 the option-argument, if the limit, the value, or both are non-numeric, only equality  
 30486 or inequality are valid comparisons.

30487 The *qselect* utility shall select only batch jobs for which the values of the  
 30488 *resource\_names* listed in the *resource\_list* option-argument match the corresponding  
 30489 limits of the *Resource\_List* attribute of the batch job.

30490 Limits of *resource\_names* present in the *Resource\_List* attribute of the batch job that  
 30491 have no corresponding values in the *resource\_list* option-argument shall not be  
 30492 considered when selecting batch jobs.

30493 **-N *name*** Restrict selection to batch jobs with a specified name.

30494 The *qselect* utility shall select only batch jobs for which the value of the *Job\_Name*  
 30495 attribute matches the value of the *name* option-argument. The string specified in

- 30496 the *name* option-argument shall be passed, uninterpreted, to the server. This allows  
30497 an implementation to match “wildcard” patterns against batch job names.
- 30498 An implementation shall describe in the conformance document the format it  
30499 supports for matching against the *Job\_Name* attribute.
- 30500 **-p [op]priority**
- 30501 Restrict selection to batch jobs of the specified priority or range of priorities.
- 30502 The *qselect* utility shall select only batch jobs for which the value of the *Priority*  
30503 attribute of the batch job relates to the value of the *priority* component of the  
30504 option-argument in the manner indicated by the value of the *op* component of the  
30505 option-argument.
- 30506 If the *op* component of the option-argument is omitted, the *qselect* utility shall  
30507 select batch jobs for which the value of the *Priority* attribute of the batch job is  
30508 equal to the value of the *priority* component of the option-argument.
- 30509 When comparing priority values, the *qselect* utility shall use the following  
30510 definitions for the *op* component of the option-argument:
- 30511 .eq. The value of the *Priority* attribute of the batch job equals the value of the  
30512 *priority* component of the option-argument.
- 30513 .ge. The value of the *Priority* attribute of the batch job is greater than or equal  
30514 to the value of the *priority* component option-argument.
- 30515 .gt. The value of the *Priority* attribute of the batch job is greater than the value  
30516 of the *priority* component option-argument.
- 30517 .lt. The value of the *Priority* attribute of the batch job is less than the value of  
30518 the *priority* component option-argument.
- 30519 .lte. The value of the *Priority* attribute of the batch job is less than or equal to  
30520 the value of the *priority* component option-argument.
- 30521 .ne. The value of the *Priority* attribute of the batch job does not equal the value  
30522 of the *priority* component option-argument.
- 30523 **-q destination**
- 30524 Restrict selection to the specified batch queue or server, or both.
- 30525 The *qselect* utility shall select only batch jobs that are located at the destination  
30526 indicated by the value of the *destination* option-argument.
- 30527 The destination defines a batch queue, a server, or a batch queue at a server.
- 30528 The *qselect* utility shall accept an option-argument for the **-q** option that conforms  
30529 to the syntax for a destination. If the **-q** option is not presented to the *qselect* utility,  
30530 the utility shall select batch jobs from all batch queues at the default batch server.
- 30531 If the option-argument describes only a batch queue, the *qselect* utility shall select  
30532 only batch jobs from the batch queue of the specified name at the default batch  
30533 server. The means by which *qselect* determines the default server is  
30534 implementation-defined.
- 30535 If the option-argument describes only a batch server, the *qselect* utility shall select  
30536 batch jobs from all the batch queues at that batch server.
- 30537 If the option-argument describes both a batch queue and a batch server, the *qselect*  
30538 utility shall select only batch jobs from the specified batch queue at the specified

- 30539 server.
- 30540 **-r y|n** Restrict selection to batch jobs with the specified rerunability status.
- 30541 The *qselect* utility shall select only batch jobs for which the value of the *Rerunable*  
30542 attribute of the batch job matches the value of the option-argument.
- 30543 The *qselect* utility shall accept a value for the option-argument that consists of  
30544 either the single character 'y' or the single character 'n'. The character 'y'  
30545 represents the value TRUE, and the character 'n' represents the value FALSE.
- 30546 **-s states** Restrict selection to batch jobs in the specified states.
- 30547 The *qselect* utility shall accept an option-argument that consists of any combination  
30548 of the characters 'e', 'q', 'r', 'w', 'h', and 't'.
- 30549 Conforming applications shall not repeat any character in the option-argument.  
30550 The *qselect* utility shall permit the repetition of characters in the option-argument,  
30551 but shall not assign additional meaning to repeated characters.
- 30552 The *qselect* utility shall interpret the characters in the *states* option-argument as  
30553 follows:
- 30554 e Represents the EXITING state.
- 30555 q Represents the QUEUED state.
- 30556 r Represents the RUNNING state.
- 30557 t Represents the TRANSITING state.
- 30558 h Represents the HELD state.
- 30559 w Represents the WAITING state.
- 30560 For each character in the *states* option-argument, the *qselect* utility shall select batch  
30561 jobs in the corresponding state.
- 30562 **-u user\_list** Restrict selection to batch jobs owned by the specified user names.
- 30563 The *qselect* utility shall select only the batch jobs of those users specified in the  
30564 *user\_list* option-argument.
- 30565 The *qselect* utility shall accept a *user\_list* option-argument that conforms to the  
30566 following syntax:
- 30567 *username*[@*host*][, , *username*[@*host*], , . . . ]
- 30568 The *qselect* utility shall accept only one user name that is missing a corresponding  
30569 host name. The *qselect* utility shall accept only one user name per named host.
- 30570 **OPERANDS**
- 30571 None.
- 30572 **STDIN**
- 30573 Not used.
- 30574 **INPUT FILES**
- 30575 None.

30576 **ENVIRONMENT VARIABLES**

30577 The following environment variables shall affect the execution of *qselect*:

30578 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 30579 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
 30580 Internationalization Variables for the precedence of internationalization variables  
 30581 used to determine the values of locale categories.)

30582 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 30583 internationalization variables.

30584 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 30585 characters (for example, single-byte as opposed to multi-byte characters in  
 30586 arguments).

30587 *LC\_MESSAGES*

30588 Determine the locale that should be used to affect the format and contents of  
 30589 diagnostic messages written to standard error.

30590 *LOGNAME* Determine the login name of the user.

30591 *TZ* Determine the timezone used to interpret the *date-time* option-argument. If *TZ* is  
 30592 unset or null, an unspecified default timezone shall be used.

30593 **ASYNCHRONOUS EVENTS**

30594 Default.

30595 **STDOUT**

30596 The *qselect* utility shall write zero or more batch *job\_identifiers* to standard output.

30597 The *qselect* utility shall separate the batch *job\_identifiers* written to standard output by white  
 30598 space.

30599 The *qselect* utility shall write batch *job\_identifiers* in the following format:

30600 *sequence\_number.server\_name@server*

30601 **STDERR**

30602 The standard error shall be used only for diagnostic messages. |

30603 **OUTPUT FILES**

30604 None.

30605 **EXTENDED DESCRIPTION**

30606 None.

30607 **EXIT STATUS**

30608 The following exit values shall be returned:

30609 0 Successful completion.

30610 >0 An error occurred.

30611 **CONSEQUENCES OF ERRORS**

30612 Default.

30613 **APPLICATION USAGE**

30614 None.

30615 **EXAMPLES**

30616 The following example shows how a user might use the *qselect* utility in conjunction with the  
30617 *qdel* utility to delete all of his or her jobs in the queued state without affecting any jobs that are  
30618 already running:

30619 `qdel $(qselect -s q)`

30620 or:

30621 `qselect -s q || xargs qdel`30622 **RATIONALE**

30623 The *qselect* utility allows users to acquire a list of job identifiers that match user-specified  
30624 selection criteria. The list of identifiers returned by the *qselect* utility conforms to the syntax of  
30625 the batch job identifier list processed by a utility such as *qmove*, *qdel*, and *qrls*. The *qselect* utility is  
30626 thus a powerful tool for causing another batch system utility to act upon a set of jobs that match  
30627 a list of selection criteria.

30628 The options of the *qselect* utility let the user apply a number of useful filters for selecting jobs.  
30629 Each option further restricts the selection of jobs. Many of the selection options allow the  
30630 specification of a relational operator. The FORTRAN-like syntax of the operator—that is,  
30631 ".lt.", was chosen rather than the C-like "<=" meta-characters.

30632 The *-a* option allows users to restrict the selected jobs to those that have been submitted (or  
30633 altered) to wait until a particular time. The time period is determined by the argument of this  
30634 option, which includes both a time and an operator—it is thus possible to select jobs waiting  
30635 until a specific time, jobs waiting until after a certain time, or those waiting for a time before the  
30636 specified time.

30637 The *-A* option allows users to restrict the selected jobs to those that have been submitted (or  
30638 altered) to charge a particular account.

30639 The *-c* option allows users to restrict the selected jobs to those whose checkpointing interval  
30640 falls within the specified range.

30641 The *-l* option allows users to select those jobs whose resource limits fall within the range  
30642 indicated by the value of the option. For example, a user could select those jobs for which the  
30643 CPU time limit is greater than two hours.

30644 The *-N* option allows users to select jobs by job name. For instance, all the parts of a task that  
30645 have been divided in parallel jobs might be given the same name, and thus manipulated as a  
30646 group by means of this option.

30647 The *-q* option allows users to select jobs in a specified queue.

30648 The *-r* option allows users to select only those jobs with a specified rerun criteria. For instance, a  
30649 user might select only those jobs that can be rerun for use with the *qrerun* utility.

30650 The *-s* option allows users to select only those jobs that are in a certain state.

30651 The *-u* option allows users to select jobs that have been submitted to execute under a particular  
30652 account.

30653 The selection criteria provided by the options of the *qselect* utility allow users to select jobs based  
30654 on all the appropriate attributes that can be assigned to jobs by the *qsub* utility.

30655 Historically, the *qselect* utility has not been a part of existing practice; it is an improvement that  
30656 has been introduced in this volume of IEEE Std 1003.1-200x.

30657 **FUTURE DIRECTIONS**

30658           None.

30659 **SEE ALSO**

30660           *qdel, qrerun, qrls, qselect, qsub, touch*, Chapter 3 (on page 2303)

30661 **CHANGE HISTORY**

30662           Derived from IEEE Std 1003.2d-1994.

## 30663 NAME

30664 qsig — signal batch jobs

## 30665 SYNOPSIS

30666 BE `qsig [-s signal] job_identifier ...`

30667

## 30668 DESCRIPTION

30669 To signal a batch job is to send a signal to the session leader of the batch job. A batch job is  
 30670 signaled by sending a request to the batch server that manages the batch job. The *qsig* utility is a  
 30671 user-accessible batch client that requests the signaling of a batch job.

30672 The *qsig* utility shall signal those batch jobs for which a batch *job\_identifier* is presented to the  
 30673 utility. The *qsig* utility shall not signal any batch jobs whose batch *job\_identifiers* are not  
 30674 presented to the utility.

30675 The *qsig* utility shall signal batch jobs in the order in which the corresponding batch  
 30676 *job\_identifiers* are presented to the utility. If the *qsig* utility fails to process a batch *job\_identifier*  
 30677 successfully, the utility shall proceed to process the remaining batch *job\_identifiers*, if any.

30678 The *qsig* utility shall signal batch jobs by sending a *Signal Job Request* to the batch server that  
 30679 manages the batch job.

30680 For each successfully processed batch *job\_identifier*, the *qsig* utility shall have received a  
 30681 completion reply to each *Signal Job Request* sent to a batch server at the time the utility exits.

## 30682 OPTIONS

30683 The *qsig* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section  
 30684 12.2, Utility Syntax Guidelines.

30685 The following option shall be supported by the implementation:

30686 `-s signal` Define the signal to be sent to the batch job.

30687 The *qsig* utility shall accept a *signal* option-argument that is either a symbolic  
 30688 signal name or an unsigned integer signal number (see the POSIX.1-1990 standard,  
 30689 Section 3.3.1.1). The *qsig* utility shall accept signal names for which the SIG prefix  
 30690 has been omitted.

30691 If the *signal* option-argument is a signal name, the *qsig* utility shall send that name.

30692 If the *signal* option-argument is a number, the *qsig* utility shall send the signal  
 30693 value represented by the number.

30694 If the `-s` option is not presented to the *qsig* utility, the utility shall send the signal  
 30695 SIGTERM to each signaled batch job.

## 30696 OPERANDS

30697 The *qsig* utility shall accept one or more operands that conform to the syntax for a batch  
 30698 *job\_identifier* (see Section 3.3.1 (on page 2324)).

## 30699 STDIN

30700 Not used.

## 30701 INPUT FILES

30702 None.

30703 **ENVIRONMENT VARIABLES**

30704 The following environment variables shall affect the execution of *qsig*:

30705 *LANG* Provide a default value for the internationalization variables that are unset or null.  
30706 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
30707 Internationalization Variables for the precedence of internationalization variables  
30708 used to determine the values of locale categories.)

30709 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
30710 internationalization variables.

30711 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
30712 characters (for example, single-byte as opposed to multi-byte characters in  
30713 arguments).

30714 *LC\_MESSAGES*  
30715 Determine the locale that should be used to affect the format and contents of  
30716 diagnostic messages written to standard error.

30717 *LOGNAME* Determine the login name of the user.

30718 **ASYNCHRONOUS EVENTS**

30719 Default.

30720 **STDOUT**

30721 An implementation of the *qsig* utility may write informative messages to standard output.

30722 **STDERR**

30723 The standard error shall be used only for diagnostic messages.

30724 **OUTPUT FILES**

30725 None.

30726 **EXTENDED DESCRIPTION**

30727 None.

30728 **EXIT STATUS**

30729 The following exit values shall be returned:

30730 0 Successful completion.

30731 >0 An error occurred.

30732 **CONSEQUENCES OF ERRORS**

30733 In addition to the default behavior, the *qsig* utility shall not be required to write a diagnostic  
30734 message to standard error when the error reply received from a batch server indicates that the  
30735 batch *job\_identifier* does not exist on the server. Whether or not the *qsig* utility waits to output the  
30736 diagnostic message while attempting to locate the batch job on other servers is implementation-  
30737 defined.

30738 **APPLICATION USAGE**

30739 None.

30740 **EXAMPLES**

30741 None.

30742 **RATIONALE**

30743 The *qsig* utility allows users to signal batch jobs.

30744 A user may be unable to signal a batch job with the *kill* utility of the operating system for a  
30745 number of reasons. First, the process ID of the batch job may be unknown to the user. Second,

- 30746 the processes of the batch job may be on a remote node. However, by virtue of communication  
30747 between batch nodes, the *qsig* utility can arrange for the signaling of a process.
- 30748 Because a batch job that is not running cannot be signaled, and because the signal may not  
30749 terminate the batch job, the *qsig* utility is not a substitute for the *qdel* utility.
- 30750 The options of the *qsig* utility allow the user to specify the signal that is to be sent to the batch  
30751 job.
- 30752 The **-s** option allows users to specify a signal by name or by number, and thus override the  
30753 default signal. The POSIX.1-1990 standard defines signals by both name and number.
- 30754 The *qsig* utility is a new utility, *vis-a-vis* existing practice; it has been defined in this volume of  
30755 IEEE Std 1003.1-200x in response to user-perceived shortcomings in existing practice.
- 30756 **FUTURE DIRECTIONS**
- 30757 None.
- 30758 **SEE ALSO**
- 30759 *kill*, *qdel*, Chapter 3 (on page 2303)
- 30760 **CHANGE HISTORY**
- 30761 Derived from IEEE Std 1003.2d-1994.
- 30762 **Issue 6**
- 30763 The *LC\_TIME* and *TZ* entries are removed from the ENVIRONMENT VARIABLES section.

30764 **NAME**

30765 qstat — show status of batch jobs

30766 **SYNOPSIS**30767 BE qstat [-f] *job\_identifier* ...30768 qstat -Q [-f] *destination* ...30769 qstat -B [-f] *server\_name* ...

30770

30771 **DESCRIPTION**

30772 The status of a batch job, batch queue, or batch server is obtained by a request to the server. The  
 30773 *qstat* utility is a user-accessible batch client that requests the status of one or more batch jobs,  
 30774 batch queues, or servers, and writes the status information to standard output.

30775 For each successfully processed batch *job\_identifier*, the *qstat* utility shall display information  
 30776 about the corresponding batch job.

30777 For each successfully processed destination, the *qstat* utility shall display information about the  
 30778 corresponding batch queue.

30779 For each successfully processed server name, the *qstat* utility shall display information about the  
 30780 corresponding server.

30781 The *qstat* utility shall acquire batch job status information by sending a *Job Status Request* to a  
 30782 batch server. The *qstat* utility shall acquire batch queue status information by sending a *Queue*  
 30783 *Status Request* to a batch server. The *qstat* utility shall acquire server status information by  
 30784 sending a *Server Status Request* to a batch server.

30785 **OPTIONS**

30786 The *qstat* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section  
 30787 12.2, Utility Syntax Guidelines.

30788 The following options shall be supported by the implementation:

30789 **-f** Specify that a full display is produced.

30790 The minimum contents of a full display are specified in the STDOUT section.

30791 Additional contents and format of a full display are implementation-defined.

30792 **-Q** Specify that the operand is a destination.

30793 The *qstat* utility shall display information about each batch queue at each  
 30794 destination identified as an operand.

30795 **-B** Specify that the operand is a server name.

30796 The *qstat* utility shall display information about each server identified as an  
 30797 operand.

30798 **OPERANDS**

30799 If the **-Q** option is presented to the *qstat* utility, the utility shall accept one or more operands that  
 30800 conform to the syntax for a destination (see Section 3.3.2 (on page 2325)).

30801 If the **-B** option is presented to the *qstat* utility, the utility shall accept one or more *server\_name*  
 30802 operands.

30803 If neither the **-B** nor the **-Q** option is presented to the *qstat* utility, the utility shall accept one or  
 30804 more operands that conform to the syntax for a batch *job\_identifier* (see Section 3.3.1 (on page  
 30805 2324)).

30806 **STDIN**

30807 Not used.

30808 **INPUT FILES**

30809 None.

30810 **ENVIRONMENT VARIABLES**30811 The following environment variables shall affect the execution of *qstat*:30812 *HOME* Determine the pathname of the user's home directory.

30813 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 30814 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
 30815 Internationalization Variables for the precedence of internationalization variables  
 30816 used to determine the values of locale categories.)

30817 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 30818 internationalization variables.

30819 *LC\_COLLATE*

30820 Determine the locale for the behavior of ranges, equivalence classes and multi-  
 30821 character collating elements within regular expressions.

30822 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 30823 characters (for example, single-byte as opposed to multi-byte characters in  
 30824 arguments).

30825 *LC\_MESSAGES*

30826 Determine the locale that should be used to affect the format and contents of  
 30827 diagnostic messages written to standard error.

30828 *LC\_NUMERIC*

30829 Determine the locale for selecting the radix character used when writing floating-  
 30830 point formatted output.

30831 **ASYNCHRONOUS EVENTS**

30832 Default.

30833 **STDOUT**

30834 If an operand presented to the *qstat* utility is a batch *job\_identifier* and the *-f* option is not  
 30835 specified, the *qstat* utility shall display the following items on a single line, in the stated order,  
 30836 with white space between each item, for each successfully processed operand:

- 30837 • The batch *job\_identifier*
- 30838 • The batch job name
- 30839 • The *Job\_Owner* attribute
- 30840 • The CPU time used by the batch job
- 30841 • The batch job state
- 30842 • The batch job location

30843 If an operand presented to the *qstat* utility is a batch *job\_identifier* and the *-f* option is specified,  
 30844 the *qstat* utility shall display the following items for each success fully processed operand:

- 30845 • The batch *job\_identifier*
- 30846 • The batch job name

- 30847 • The *Job\_Owner* attribute
  - 30848 • The execution user ID
  - 30849 • The CPU time used by the batch job
  - 30850 • The batch job state
  - 30851 • The batch job location
  - 30852 • Additional implementation-defined information, if any, about the batch job or batch queue
- 30853 If an operand presented to the *qstat* utility is a destination, the **-Q** option is specified, and the **-f**  
 30854 option is not specified, the *qstat* utility shall display the following items on a single line, in the  
 30855 stated order, with white space between each item, for each successfully processed operand:
- 30856 • The batch queue name
  - 30857 • The maximum number of batch jobs that shall be run in the batch queue concurrently |
  - 30858 • The total number of batch jobs in the batch queue
  - 30859 • The status of the batch queue
  - 30860 • For each state, the number of batch jobs in that state in the batch queue and the name of the  
 30861 state
  - 30862 • The type of batch queue (execution or routing)
- 30863 If the operands presented to the *qstat* utility are destinations, the **-Q** option is specified, and the  
 30864 **-f** option is specified, the *qstat* utility shall display the following items for each successfully  
 30865 processed operand:
- 30866 • The batch queue name
  - 30867 • The maximum number of batch jobs that shall be run in the batch queue concurrently |
  - 30868 • The total number of batch jobs in the batch queue
  - 30869 • The status of the batch queue
  - 30870 • For each state, the number of batch jobs in that state in the batch queue and the name of the  
 30871 state
  - 30872 • The type of batch queue (execution or routing)
  - 30873 • Additional implementation-defined information, if any, about the batch queue
- 30874 If the operands presented to the *qstat* utility are batch server names, the **-B** option is specified,  
 30875 and the **-f** option is not specified, the *qstat* utility shall display the following items on a single  
 30876 line, in the stated order, with white space between each item, for each successfully processed  
 30877 operand:
- 30878 • The batch server name
  - 30879 • The maximum number of batch jobs that shall be run in the batch queue concurrently |
  - 30880 • The total number of batch jobs managed by the batch server
  - 30881 • The status of the batch server
  - 30882 • For each state, the number of batch jobs in that state and the name of the state
- 30883 If the operands presented to the *qstat* utility are server names, the **-B** option is specified, and the  
 30884 **-f** option is specified, the *qstat* utility shall display the following items for each successfully  
 30885 processed operand:

- 30886           • The server name
- 30887           • The maximum number of batch jobs that shall be run in the batch queue concurrently |
- 30888           • The total number of batch jobs managed by the server
- 30889           • The status of the server
- 30890           • For each state, the number of batch jobs in that state and the name of the state
- 30891           • Additional implementation-defined information, if any, about the server
- 30892 **STDERR**
- 30893           The standard error shall be used only for diagnostic messages. |
- 30894 **OUTPUT FILES**
- 30895           None.
- 30896 **EXTENDED DESCRIPTION**
- 30897           None.
- 30898 **EXIT STATUS**
- 30899           The following exit values shall be returned:
- 30900           0   Successful completion.
- 30901           >0  An error occurred.
- 30902 **CONSEQUENCES OF ERRORS**
- 30903           In addition to the default behavior, the *qstat* utility shall not be required to write a diagnostic message to standard error when the error reply received from a batch server indicates that the batch *job\_identifier* does not exist on the server. Whether or not the *qstat* utility waits to output the diagnostic message while attempting to locate the batch job on other servers is implementation-defined.
- 30904
- 30905
- 30906
- 30907
- 30908 **APPLICATION USAGE**
- 30909           None.
- 30910 **EXAMPLES**
- 30911           None.
- 30912 **RATIONALE**
- 30913           The *qstat* utility allows users to display the status of jobs and listing the batch jobs in queues.
- 30914           The operands of the *qstat* utility may be either job identifiers, queues (specified as destination identifiers), or batch server names. The **-Q** and **-B** options, or absence thereof, indicate the nature of the operands.
- 30915
- 30916
- 30917           The other options of the *qstat* utility allow the user to control the amount of information displayed and the format in which it is displayed. Should a user wish to display the status of a set of jobs that match a selection criteria, the *qselect* utility may be used to acquire such a list.
- 30918
- 30919
- 30920           The **-f** option allows users to request a “full” display in an implementation-defined format.
- 30921           Historically, the *qstat* utility has been a part of the NQS and its derivatives, the existing practice on which it is based.
- 30922
- 30923 **FUTURE DIRECTIONS**
- 30924           None.

30925 **SEE ALSO**

30926 *qselect*, Chapter 3 (on page 2303)

30927 **CHANGE HISTORY**

30928 Derived from IEEE Std 1003.2d-1994.

30929 **Issue 6**

30930 IEEE PASC Interpretation 1003.2 #191 is applied, removing the following ENVIRONMENT VARIABLES listed as affecting *qstat*: *COLUMNS*, *LINES*, *LOGNAME*, *TERM*, and *TZ*.

30932 The *LC\_TIME* entry is also removed from the ENVIRONMENT VARIABLES section.

## 30933 NAME

30934 qsub — submit a script

## 30935 SYNOPSIS

```

30936 BE qsub [-a date_time][-A account_string][-c interval]
30937 [-C directive_prefix][-e path_name][-h][-j join_list][-k keep_list]
30938 [-m mail_options][-M mail_list][-N name]
30939 [-o path_name][-p priority][-q destination][-r y|n]
30940 [-S path_name_list][-u user_list][-v variable_list][-V]
30941 [-z][script]
30942

```

## 30943 DESCRIPTION

30944 To submit a script is to create a batch job that executes the script. A script is submitted by a  
 30945 request to a batch server. The *qsub* utility is a user-accessible batch client that submits a script.

30946 Upon successful completion, the *qsub* utility shall have created a batch job that will execute the  
 30947 submitted script.

30948 The *qsub* utility shall submit a script by sending a *Queue Job Request* to a batch server.

30949 The *qsub* utility shall place the value of the following environment variables in the *Variable\_List*  
 30950 attribute of the batch job: *HOME*, *LANG*, *LOGNAME*, *PATH*, *MAIL*, *SHELL*, and *TZ*. The name  
 30951 of the environment variable shall be the current name prefixed with the string *PBS\_O\_*.

30952 **Note:** If the current value of the *HOME* variable in the environment space of the *qsub* utility is  
 30953 */aa/bb/cc*, then *qsub* shall place *PBS\_O\_HOME=/aa/bb/cc* in the *Variable\_List* attribute of the  
 30954 batch job.

30955 In addition to the variables described above, the *qsub* utility shall add the following variables  
 30956 with the indicated values to the variable list:

30957 *PBS\_O\_WORKDIR* The absolute path of the current working directory of the *qsub* utility  
 30958 process.

30959 *PBS\_O\_HOST* The name of the host on which the *qsub* utility is running.

## 30960 OPTIONS

30961 The *qsub* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section  
 30962 12.2, Utility Syntax Guidelines.

30963 The following options shall be supported by the implementation:

30964 **-a *date\_time*** Define the time at which a batch job becomes eligible for execution.

30965 The *qsub* utility shall accept an option-argument that conforms to the syntax of the  
 30966 *time* operand of the *touch* utility.

30967

**Table 4-18** Environment Variable Values (Utilities)

30968

| Variable Name        | Value at qsub Time        |
|----------------------|---------------------------|
| <i>PBS_O_HOME</i>    | <i>HOME</i>               |
| <i>PBS_O_HOST</i>    | Client host name          |
| <i>PBS_O_LANG</i>    | <i>LANG</i>               |
| <i>PBS_O_LOGNAME</i> | <i>LOGNAME</i>            |
| <i>PBS_O_PATH</i>    | <i>PATH</i>               |
| <i>PBS_O_MAIL</i>    | <i>MAIL</i>               |
| <i>PBS_O_SHELL</i>   | <i>SHELL</i>              |
| <i>PBS_O_TZ</i>      | <i>TZ</i>                 |
| <i>PBS_O_WORKDIR</i> | Current working directory |

30969

30970

30971

30972

30973

30974

30975

30976

30977

30978

30979

**Note:** The server that initiates execution of the batch job will add other variables to the batch job's environment; see Section 3.2.2.1 (on page 2308).

30980

30981

30982

30983

The *qsub* utility shall set the *Execution\_Time* attribute of the batch job to the number of seconds since the Epoch that is equivalent to the local time expressed by the value of the *date\_time* option-argument. The Epoch is defined in the Base Definitions volume of IEEE Std 1003.1-200x, Section 3.149, Epoch.

30984

30985

30986

If the *-a* option is not presented to the *qsub* utility, the utility shall set the *Execution\_Time* attribute of the batch job to a time (number of seconds since the Epoch) that is earlier than the time at which the utility exits.

30987

**-A** *account\_string*

30988

30989

Define the account to which the resource consumption of the batch job should be charged.

30990

The syntax of the *account\_string* option-argument is unspecified.

30991

30992

The *qsub* utility shall set the *Account\_Name* attribute of the batch job to the value of the *account\_string* option-argument.

30993

30994

If the *-A* option is not presented to the *qsub* utility, the utility shall omit the *Account\_Name* attribute from the attributes of the batch job.

30995

**-c** *interval*

Define whether the batch job should be checkpointed, and if so, how often.

30996

30997

The *qsub* utility shall accept a value for the interval option-argument that is one of the following:

30998

30999

*n* No checkpointing shall be performed on the batch batch job (NO\_CHECKPOINT).

31000

31001

*s* Checkpointing shall be performed only when the batch server is shut down (CHECKPOINT\_AT\_SHUTDOWN).

31002

31003

31004

*c* Automatic periodic checkpointing shall be performed at the *Minimum\_Cpu\_Interval* attribute of the batch queue, in units of CPU minutes (CHECKPOINT\_AT\_MIN\_CPU\_INTERVAL).

31005

31006

31007

31008

*c=minutes* Automatic periodic checkpointing shall be performed every *minutes* of CPU time, or every *Minimum\_Cpu\_Interval* minutes, whichever is greater. The *minutes* argument shall conform to the syntax for unsigned integers and shall be greater than zero.

31009

31010

The *qsub* utility shall set the *Checkpoint* attribute of the batch job to the value of the *interval* option-argument.

31011 If the `-c` option is not presented to the *qsub* utility, the utility shall set the  
 31012 *Checkpoint* attribute of the batch job to the single character 'u'  
 31013 (CHECKPOINT\_UNSPECIFIED).

31014 **-C *directive\_prefix***  
 31015 Define the prefix that declares a directive to the *qsub* utility within the script.

31016 The *directive\_prefix* is not a batch job attribute; it affects the behavior of the *qsub*  
 31017 utility.

31018 If the `-C` option is presented to the *qsub* utility, and the value of the *directive\_prefix*  
 31019 option-argument is the null string, the utility shall not scan the script file for  
 31020 directives. If the `-C` option is not presented to the *qsub* utility, then the value of the  
 31021 *PBS\_DPREFIX* environment variable is used. If the environment variable is not  
 31022 defined, then #PBS encoded in the portable character set is the default.

31023 **-e *path\_name*** Define the path to be used for the standard error stream of the batch job.

31024 The *qsub* utility shall accept a *path\_name* option-argument which can be preceded  
 31025 by a host name element of the form *hostname*..

31026 If the *path\_name* option-argument constitutes an absolute pathname, the *qsub*  
 31027 utility shall set the *Error\_Path* attribute of the batch job to the value of the  
 31028 *path\_name* option-argument.

31029 If the *path\_name* option-argument constitutes a relative pathname and no host  
 31030 name element is specified, the *qsub* utility shall set the *Error\_Path* attribute of the  
 31031 batch job to the value of the absolute pathname derived by expanding the  
 31032 *path\_name* option-argument relative to the current directory of the process  
 31033 executing *qsub*.

31034 If the *path\_name* option-argument constitutes a relative pathname and a host name  
 31035 element is specified, the *qsub* utility shall set the *Error\_Path* attribute of the batch  
 31036 job to the value of the *path\_name* option-argument without expansion. The host  
 31037 name element shall be included.

31038 If the *path\_name* option-argument does not include a host name element, the *qsub*  
 31039 utility shall prefix the pathname with *hostname*., where *hostname* is the name of the  
 31040 host upon which the *qsub* utility is being executed.

31041 If the `-e` option is not presented to the *qsub* utility, the utility shall set the  
 31042 *Error\_Path* attribute of the batch job to the host name and path of the current  
 31043 directory of the submitting process and the default filename.

31044 The default filename for standard error has the following format:

31045 *job\_name . esequence\_number*

31046 **-h** Specify that a USER hold is applied to the batch job.

31047 The *qsub* utility shall set the value of the *Hold\_Types* attribute of the batch job to the  
 31048 value USER.

31049 If the `-h` option is not presented to the *qsub* utility, the utility shall set the  
 31050 *Hold\_Types* attribute of the batch job to the value NO\_HOLD.

31051 **-j *join\_list*** Define which streams of the batch job are to be merged. The *qsub* `-j` option shall  
 31052 accept a value for the *join\_list* option-argument that is a string of alphanumeric  
 31053 characters in the portable character set (see the Base Definitions volume of  
 31054 IEEE Std 1003.1-200x, Section 6.1, Portable Character Set).

- 31055 The *qsub* utility shall accept a *join\_list* option-argument that consists of one or  
31056 more of the characters 'e' and 'o' or the single character 'n'.
- 31057 All of the other batch job output streams specified will be merged into the output  
31058 stream represented by the character listed first in the *join\_list* option-argument.
- 31059 For each unique character in the *join\_list* option-argument, the *qsub* utility shall  
31060 add a value to the *Join\_Path* attribute of the batch job as follows, each representing  
31061 a different batch job stream to join:
- 31062 e The standard error of the batch batch job (JOIN\_STD\_ERROR).
  - 31063 o The standard output of the batch batch job (JOIN\_STD\_OUTPUT).
- 31064 An existing *Join\_Path* attribute can be cleared by the following join type:
- 31065 n NO\_JOIN
- 31066 If 'n' is specified, then no files are joined. The *qsub* utility shall consider it an error  
31067 if any join type other than 'n' is combined with join type 'n'.
- 31068 Strictly conforming applications shall not repeat any of the characters 'e', 'o', or  
31069 'n' within the *join\_list* option-argument. The *qsub* utility shall permit the  
31070 repetition of characters, but shall not assign additional meaning to the repeated  
31071 characters.
- 31072 An implementation may define other join types. The conformance document for an  
31073 implementation shall describe any additional batch job streams, how they are  
31074 specified, their internal behavior, and how they affect the behavior of the utility.
- 31075 If the **-j** option is not presented to the *qsub* utility, the utility shall set the value of  
31076 the *Join\_Path* attribute of the batch job to NO\_JOIN.
- 31077 **-k keep\_list** Define which output of the batch job to retain on the execution host.
- 31078 The *qsub* **-k** option shall accept a value for the *keep\_list* option-argument that is a  
31079 string of alphanumeric characters in the portable character set (see the Base  
31080 Definitions volume of IEEE Std 1003.1-200x, Section 6.1, Portable Character Set).
- 31081 The *qsub* utility shall accept a *keep\_list* option-argument that consists of one or  
31082 more of the characters 'e' and 'o' or the single character 'n'.
- 31083 For each unique character in the *keep\_list* option-argument, the *qsub* utility shall  
31084 add a value to the *Keep\_Files* attribute of the batch job as follows, each representing  
31085 a different batch job stream to keep:
- 31086 e The standard error of the batch batch job (KEEP\_STD\_ERROR).
  - 31087 o The standard output of the batch batch job (KEEP\_STD\_OUTPUT).
- 31088 If both 'e' and 'o' are specified, then both files are retained. An existing  
31089 *Keep\_Files* attribute can be cleared by the following keep type:
- 31090 n NO\_KEEP
- 31091 If 'n' is specified, then no files are retained. The *qsub* utility shall consider it an  
31092 error if any keep type other than 'n' is combined with keep type 'n'.
- 31093 Strictly conforming applications shall not repeat any of the characters 'e', 'o', or  
31094 'n' within the *keep\_list* option-argument. The *qsub* utility shall permit the  
31095 repetition of characters, but shall not assign additional meaning to the repeated  
31096 characters.

- 31097 An implementation may define other keep types. The conformance document for  
 31098 an implementation shall describe any additional keep types, how they are  
 31099 specified, their internal behavior, and how they affect the behavior of the utility. If  
 31100 the **-k** option is not presented to the *qsub* utility, the utility shall set the *Keep\_Files*  
 31101 attribute of the batch job to the value NO\_KEEP.
- 31102 **-m** *mail\_options*
- 31103 Define the points in the execution of the batch job at which the batch server that  
 31104 manages the batch job shall send mail about a change in the state of the batch job.
- 31105 The *qsub -m* option shall accept a value for the *mail\_options* option-argument that  
 31106 is a string of alphanumeric characters in the portable character set (see the Base  
 31107 Definitions volume of IEEE Std 1003.1-200x, Section 6.1, Portable Character Set).
- 31108 The *qsub* utility shall accept a value for the *mail\_options* option-argument that is a  
 31109 string of one or more of the characters 'e', 'b', and 'a', or the single character  
 31110 'n'.
- 31111 For each unique character in the *mail\_options* option-argument, the *qsub* utility shall  
 31112 add a value to the *Mail\_Users* attribute of the batch job as follows, each  
 31113 representing a different time during the life of a batch job at which to send mail:
- 31114 e MAIL\_AT\_EXIT
- 31115 b MAIL\_AT\_BEGINNING
- 31116 a MAIL\_AT\_ABORT
- 31117 If any of these characters are duplicated in the *mail\_options* option-argument, the  
 31118 duplicates shall be ignored.
- 31119 An existing *Mail\_Points* attribute can be cleared by the following mail type:
- 31120 n NO\_MAIL
- 31121 If 'n' is specified, then mail is not sent. The *qsub* utility shall consider it an error if  
 31122 any mail type other than 'n' is combined with mail type 'n'.
- 31123 Strictly conforming applications shall not repeat any of the characters 'e', 'b',  
 31124 'a', or 'n' within the *mail\_options* option-argument.
- 31125 The *qsub* utility shall permit the repetition of characters, but shall not assign  
 31126 additional meaning to the repeated characters. An implementation may define  
 31127 other mail types. The conformance document for an implementation shall describe  
 31128 any additional mail types, how they are specified, their internal behavior, and how  
 31129 they affect the behavior of the utility.
- 31130 If the **-m** option is not presented to the *qsub* utility, the utility shall set the  
 31131 *Mail\_Points* attribute to the value MAIL\_AT\_ABORT.
- 31132 **-M** *mail\_list* Define the list of users to which a batch server that executes the batch job shall  
 31133 send mail, if the server sends mail about the batch job.
- 31134 The syntax of the *mail\_list* option-argument is unspecified.
- 31135 If the implementation of the *qsub* utility uses a name service to locate users, the  
 31136 utility should accept the syntax used by the name service.
- 31137 If the implementation of the *qsub* utility does not use a name service to locate  
 31138 users, the implementation should accept the following syntax for user names:

- 31139 *mail\_address*[ , , *mail\_address* , , ... ]
- 31140 The interpretation of *mail\_address* is implementation-defined.
- 31141 The *qsub* utility shall set the *Mail\_Users* attribute of the batch job to the value of the  
31142 *mail\_list* option-argument.
- 31143 If the **-M** option is not presented to the *qsub* utility, the utility shall place only the  
31144 user name and host name for the current process in the *Mail\_Users* attribute of the  
31145 batch job.
- 31146 **-N name** Define the name of the batch job.
- 31147 The *qsub* **-N** option shall accept a value for the *name* option-argument that is a  
31148 string of up to 15 alphanumeric characters in the portable character set (see the  
31149 Base Definitions volume of IEEE Std 1003.1-200x, Section 6.1, Portable Character  
31150 Set) where the first character is alphabetic.
- 31151 The *qsub* utility shall set the value of the *Job\_Name* attribute of the batch job to the  
31152 value of the *name* option-argument.
- 31153 If the **-N** option is not presented to the *qsub* utility, the utility shall set the  
31154 *Job\_Name* attribute of the batch job to the name of the *script* argument from which  
31155 the directory specification if any, has been removed.
- 31156 If the **-N** option is not presented to the *qsub* utility, and the script is read from  
31157 standard input, the utility shall set the *Job\_Name* attribute of the batch job to the  
31158 value STDIN.
- 31159 **-o path\_name** Define the path for the standard output of the batch job.
- 31160 The *qsub* utility shall accept a *path\_name* option-argument that conforms to the  
31161 syntax of the *path\_name* element defined in the System Interfaces volume of  
31162 IEEE Std 1003.1-200x, which can be preceded by a host name element of the form  
31163 *hostname*:
- 31164 If the *path\_name* option-argument constitutes an absolute pathname, the *qsub*  
31165 utility shall set the *Output\_Path* attribute of the batch job to the value of the  
31166 *path\_name* option-argument without expansion.
- 31167 If the *path\_name* option-argument constitutes a relative pathname and no host  
31168 name element is specified, the *qsub* utility shall set the *Output\_Path* attribute of the  
31169 batch job to the pathname derived by expanding the value of the *path\_name*  
31170 option-argument relative to the current directory of the process executing the *qsub*.
- 31171 If the *path\_name* option-argument constitutes a relative pathname and a host name  
31172 element is specified, the *qsub* utility shall set the *Output\_Path* attribute of the batch  
31173 job to the value of the *path\_name* option-argument without expansion.
- 31174 If the *path\_name* option-argument does not specify a host name element, the *qsub*  
31175 utility shall prefix the pathname with *hostname*:, where *hostname* is the name of the  
31176 host upon which the *qsub* utility is executing.
- 31177 If the **-o** option is not presented to the *qsub* utility, the utility shall set the  
31178 *Output\_Path* attribute of the batch job to the host name and path of the current  
31179 directory of the submitting process and the default filename.
- 31180 The default filename for standard output has the following format:
- 31181 *job\_name.osequence\_number*

|       |                                 |                                                                                                                                                                                                                                      |
|-------|---------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 31182 | <b>-p <i>priority</i></b>       | Define the priority the batch job should have relative to other batch jobs owned by the batch server.                                                                                                                                |
| 31183 |                                 |                                                                                                                                                                                                                                      |
| 31184 |                                 | The <i>qsub</i> utility shall set the <i>Priority</i> attribute of the batch job to the value of the <i>priority</i> option-argument.                                                                                                |
| 31185 |                                 |                                                                                                                                                                                                                                      |
| 31186 |                                 | If the <b>-p</b> option is not presented to the <i>qsub</i> utility, the value of the <i>Priority</i> attribute is implementation-defined.                                                                                           |
| 31187 |                                 |                                                                                                                                                                                                                                      |
| 31188 |                                 | The <i>qsub</i> utility shall accept a value for the <i>priority</i> option-argument that conforms to the syntax for signed decimal integers, and which is not less than -1 024 and not greater than 1 023.                          |
| 31189 |                                 |                                                                                                                                                                                                                                      |
| 31190 |                                 |                                                                                                                                                                                                                                      |
| 31191 | <b>-q <i>destination</i></b>    | Define the destination of the batch job.                                                                                                                                                                                             |
| 31192 |                                 |                                                                                                                                                                                                                                      |
| 31193 |                                 | The destination is not a batch job attribute; it determines the batch server, and possibly the batch queue, to which the <i>qsub</i> utility batch queues the batch job.                                                             |
| 31194 |                                 |                                                                                                                                                                                                                                      |
| 31195 |                                 | The <i>qsub</i> utility shall submit the script to the batch server named by the <i>destination</i> option-argument or the server that owns the batch queue named in the <i>destination</i> option-argument.                         |
| 31196 |                                 |                                                                                                                                                                                                                                      |
| 31197 |                                 |                                                                                                                                                                                                                                      |
| 31198 |                                 | The <i>qsub</i> utility shall accept an option-argument for the <b>-q</b> option that conforms to the syntax for a destination (see Section 3.3.2 (on page 2325)).                                                                   |
| 31199 |                                 |                                                                                                                                                                                                                                      |
| 31200 |                                 | If the <b>-q</b> option is not presented to the <i>qsub</i> utility, the <i>qsub</i> utility shall submit the batch job to the default destination. The mechanism for determining the default destination is implementation-defined. |
| 31201 |                                 |                                                                                                                                                                                                                                      |
| 31202 |                                 |                                                                                                                                                                                                                                      |
| 31203 | <b>-r <i>y</i>   <i>n</i></b>   | Define whether the batch job is rerunnable.                                                                                                                                                                                          |
| 31204 |                                 |                                                                                                                                                                                                                                      |
| 31205 |                                 | If the value of the option-argument is <i>y</i> , the <i>qsub</i> utility shall set the <i>Rerunnable</i> attribute of the batch job to TRUE.                                                                                        |
| 31206 |                                 | If the value of the option-argument is <i>n</i> , the <i>qsub</i> utility shall set the <i>Rerunnable</i> attribute of the batch job to FALSE.                                                                                       |
| 31207 |                                 |                                                                                                                                                                                                                                      |
| 31208 |                                 | If the <b>-r</b> option is not presented to the <i>qsub</i> utility, the utility shall set the <i>Rerunnable</i> attribute of the batch job to TRUE.                                                                                 |
| 31209 |                                 |                                                                                                                                                                                                                                      |
| 31210 | <b>-S <i>path_name_list</i></b> | Define the pathname to the shell under which the batch job is to execute.                                                                                                                                                            |
| 31211 |                                 |                                                                                                                                                                                                                                      |
| 31212 |                                 | The <i>qsub</i> utility shall accept a <i>path_name_list</i> option-argument that conforms to the following syntax:                                                                                                                  |
| 31213 |                                 |                                                                                                                                                                                                                                      |
| 31214 |                                 | <i>pathname</i> [@ <i>host</i> ][, , <i>pathname</i> [@ <i>host</i> ], , . . . ]                                                                                                                                                     |
| 31215 |                                 | The <i>qsub</i> utility shall allow only one pathname for a given host name. The <i>qsub</i> utility shall allow only one pathname that is missing a corresponding host name.                                                        |
| 31216 |                                 |                                                                                                                                                                                                                                      |
| 31217 |                                 | The <i>qsub</i> utility shall add a value to the <i>Shell_Path_List</i> attribute of the batch job for each entry in the <i>path_name_list</i> option-argument.                                                                      |
| 31218 |                                 |                                                                                                                                                                                                                                      |
| 31219 |                                 | If the <b>-S</b> option is not presented to the <i>qsub</i> utility, the utility shall set the <i>Shell_Path_List</i> attribute of the batch job to the null string.                                                                 |
| 31220 |                                 |                                                                                                                                                                                                                                      |
| 31221 |                                 | The conformance document for an implementation shall describe the mechanism used to set the default shell and determine the current value of the default shell.                                                                      |
| 31222 |                                 |                                                                                                                                                                                                                                      |
| 31223 |                                 | An implementation shall provide a means for the installation to set the default shell to the login shell of the user under which the batch job is to execute. See                                                                    |
| 31224 |                                 |                                                                                                                                                                                                                                      |

- 31225 Section 3.3.3 (on page 2325) for a means of removing *keyword=value* (and  
31226 *value@keyword*) pairs and other general rules for list-oriented batch job attributes.
- 31227 **-u *user\_list*** Define the user name under which the batch job is to execute.
- 31228 The *qsub* utility shall accept a *user\_list* option-argument that conforms to the  
31229 following syntax:
- 31230 *username[@host][, ,username[@host], , ...]*
- 31231 The *qsub* utility shall accept only one user name that is missing a corresponding  
31232 host name. The *qsub* utility shall accept only one user name per named host.
- 31233 The *qsub* utility shall add a value to the *User\_List* attribute of the batch job for each  
31234 entry in the *user\_list* option-argument.
- 31235 If the **-u** option is not presented to the *qsub* utility, the utility shall set the *User\_List*  
31236 attribute of the batch job to the user name from which the utility is executing. See  
31237 Section 3.3.3 (on page 2325) for a means of removing *keyword=value* (and  
31238 *value@keyword*) pairs and other general rules for list-oriented batch job attributes.
- 31239 **-v *variable\_list***
- 31240 Add to the list of variables that are exported to the session leader of the batch job.
- 31241 A *variable\_list* is a set of strings of either the form *<variable>* or *<variable=value>*,  
31242 delimited by commas.
- 31243 If the **-v** option is presented to the *qsub* utility, the utility shall also add, to the  
31244 environment *Variable\_List* attribute of the batch job, every variable named in the  
31245 environment *variable\_list* option-argument and, optionally, values of specified  
31246 variables.
- 31247 If a value is not provided on the command line, the *qsub* utility shall set the value  
31248 of each variable in the environment *Variable\_List* attribute of the batch job to the  
31249 value of the corresponding environment variable for the process in which the  
31250 utility is executing; see Table 4-18 (on page 3003).
- 31251 A conforming application shall not repeat a variable in the environment  
31252 *variable\_list* option-argument.
- 31253 The *qsub* utility shall not repeat a variable in the environment *Variable\_List*  
31254 attribute of the batch job. See Section 3.3.3 (on page 2325) for a means of removing  
31255 *keyword=value* (and *value@keyword*) pairs and other general rules for list-oriented  
31256 batch job attributes.
- 31257 **-V** Specify that all of the environment variables of the process are exported to the  
31258 context of the batch job.
- 31259 The *qsub* utility shall place every environment variable in the process in which the  
31260 utility is executing in the list and shall set the value of each variable in the attribute  
31261 to the value of that variable in the process.
- 31262 **-z** Specify that the utility does not write the batch *job\_identifier* of the created batch  
31263 job to standard output.
- 31264 If the **-z** option is presented to the *qsub* utility, the utility shall not write the batch  
31265 *job\_identifier* of the created batch job to standard output.
- 31266 If the **-z** option is not presented to the *qsub* utility, the utility shall write the  
31267 identifier of the created batch job to standard output.

31268 **OPERANDS**

- 31269 The *qsub* utility shall accept a *script* operand that indicates the path to the script of the batch job.
- 31270 If the *script* operand is not presented to the *qsub* utility, or if the operand is the single-character  
31271 string ' - ', the utility shall read the script from standard input.
- 31272 If the script represents a partial path, the *qsub* utility shall expand the path relative to the current  
31273 directory of the process executing the utility.

31274 **STDIN**

- 31275 The *qsub* utility reads the script of the batch job from standard input if the script operand is  
31276 omitted or is the single character ' - '.

31277 **INPUT FILES**

- 31278 In addition to binding the file indicated by the *script* operand to the batch job, the *qsub* utility  
31279 reads the script file and acts on directives in the script.

31280 **ENVIRONMENT VARIABLES**

- 31281 The following environment variables shall affect the execution of *qsub*:

- 31282 *LANG* Provide a default value for the internationalization variables that are unset or null.  
31283 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
31284 Internationalization Variables for the precedence of internationalization variables  
31285 used to determine the values of locale categories.)
- 31286 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
31287 internationalization variables.
- 31288 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
31289 characters (for example, single-byte as opposed to multi-byte characters in  
31290 arguments).
- 31291 *LC\_MESSAGES*  
31292 Determine the locale that should be used to affect the format and contents of  
31293 diagnostic messages written to standard error.
- 31294 *LOGNAME* Determine the login name of the user.
- 31295 *PBS\_DPREFIX*  
31296 Determine the default prefix for directives within the script.
- 31297 *SHELL* Determine the pathname of the preferred command language interpreter of the  
31298 user.
- 31299 *TZ* Determine the timezone used to interpret the *date-time* option-argument. If *TZ* is  
31300 unset or null, an unspecified default timezone shall be used.

31301 **ASYNCHRONOUS EVENTS**

- 31302 Once created, a batch job exists until it exits, aborts, or is deleted.
- 31303 After a batch job is created by the *qsub* utility, batch servers might route, execute, modify, or  
31304 delete the batch job.

31305 **STDOUT**

- 31306 The *qsub* utility writes the batch *job\_identifier* assigned to the batch job to standard output, unless  
31307 the *-z* option is specified.

31308 **STDERR**

31309 The standard error shall be used only for diagnostic messages.

31310 **OUTPUT FILES**

31311 None.

31312 **EXTENDED DESCRIPTION**31313 **Script Preservation**

31314 The *qsub* utility shall make the script available to the server executing the batch job in such a way  
31315 that the server executes the script as it exists at the time of submission.

31316 The *qsub* utility can send a copy of the script to the server with the *Queue Job Request* or store a  
31317 temporary copy of the script in a location specified to the server.

31318 **Option Specification**

31319 A script can contain directives to the *qsub* utility.

31320 The *qsub* utility shall scan the lines of the script for directives, skipping blank lines, until the first  
31321 line that begins with a string other than the directive string; if directives occur on subsequent  
31322 lines, the utility shall ignore those directives.

31323 Lines are separated by a <newline>. If the first line of the script begins with "#!" or a colon  
31324 (' : '), then it is skipped. The *qsub* utility shall process a line in the script as a directive if and  
31325 only if the string of characters from the first non-white-space character on the line until the first  
31326 <space> or <tab> on the line match the directive prefix. If a line in the script contains a directive  
31327 and the final characters of the line are backslash ('\ ') and <newline>, then the next line shall be  
31328 interpreted as a continuation of that directive.

31329 The *qsub* utility shall process the options and option-arguments contained on the directive prefix  
31330 line using the same syntax as if the options were input on the *qsub* utility.

31331 The *qsub* utility shall continue to process a directive prefix line until after a <newline> is  
31332 encountered. An implementation may ignore lines which, according to the syntax of the shell  
31333 that will interpret the script, are comments. An implementation shall describe in the  
31334 conformance document the format of any shell comments that it will recognize.

31335 If an option is present in both a directive and the arguments to the *qsub* utility, the utility shall  
31336 ignore the option and the corresponding option-argument, if any, in the directive.

31337 If an option that is present in the directive is not present in the arguments to the *qsub* utility, the  
31338 utility shall process the option and the option-argument, if any.

31339 In order of preference, the *qsub* utility shall select the directive prefix from one of the following  
31340 sources:

- 31341 • If the **-C** option is presented to the utility, the value of the *directive\_prefix* option-argument
- 31342 • If the environment variable *PBS\_DPREFIX* is defined, the value of that variable
- 31343 • The four-character string "#PBS" encoded in the portable character set

31344 If the **-C** option is present in the script file it shall be ignored.

31345 **EXIT STATUS**

31346 The following exit values shall be returned:

31347 0 Successful completion.

31348 >0 An error occurred.

### 31349 CONSEQUENCES OF ERRORS

31350 Default.

### 31351 APPLICATION USAGE

31352 None.

### 31353 EXAMPLES

31354 None.

### 31355 RATIONALE

31356 The *qsub* utility allows users to create a batch job that will process the script specified as the  
31357 operand of the utility.

31358 The options of the *qsub* utility allow users to control many aspects of the queuing and execution  
31359 of a batch job.

31360 The **-a** option allows users to designate the time after which the batch job will become eligible to  
31361 run. By specifying an execution time, users can take advantage of resources at off-peak hours,  
31362 synchronize jobs with chronologically predictable events, and perhaps take advantage of off-  
31363 peak pricing of computing time. For these reasons and others, a timing option is existing practice  
31364 on the part of almost every batch system, including NQS.

31365 The **-A** option allows users to specify the account that will be charged for the batch job. Support  
31366 for account is not mandatory for conforming batch servers.

31367 The **-C** option allows users to prescribe the prefix for directives within the script file. The default  
31368 prefix "#PBS" may be inappropriate if the script will be interpreted with an alternate shell, as  
31369 specified by the **-S** option.

31370 The **-c** option allows users to establish the checkpointing interval for their jobs. A checkpointing  
31371 system, which is not defined by this volume of IEEE Std 1003.1-200x, allows recovery of a batch  
31372 job at the most recent checkpoint in the event of a crash. Checkpointing is typically used for jobs  
31373 that consume expensive computing time or must meet a critical schedule. Users should be  
31374 allowed to make the tradeoff between the overhead of checkpointing and the risk to the timely  
31375 completion of the batch job; therefore, this volume of IEEE Std 1003.1-200x provides the  
31376 checkpointing interval option. Support for checkpointing is optional for batch servers.

31377 The **-e** option allows users to redirect the standard error streams of their jobs to a non-default  
31378 path. For example, if the submitted script generally produces a great deal of useless error output,  
31379 a user might redirect the standard error output to the null device. Or, if the file system holding  
31380 the default location (the home directory of the user) has too little free space, the user might  
31381 redirect the standard error stream to a file in another file system.

31382 The **-h** option allows users to create a batch job that is held until explicitly released. The ability  
31383 to create a held job is useful when some external event must complete before the batch job can  
31384 execute. For example, the user might submit a held job and release it when the system load has  
31385 dropped.

31386 The **-j** option allows users to merge the standard error of a batch job into its standard output  
31387 stream, which has the advantage of showing the sequential relationship between output and  
31388 error messages.

31389 The **-m** option allows users to designate those points in the execution of a batch job at which  
31390 mail will be sent to the submitting user, or to the account(s) indicated by the **-M** option. By  
31391 requesting mail notification at points of interest in the life of a job, the submitting user, or other  
31392 designated users, can track the progress of a batch job.

- 31393 The **-N** option allows users to associate a name with the batch job. The job name in no way  
31394 affects the processing of the batch job, but rather serves as a mnemonic handle for users. For  
31395 example, the batch job name can help the user distinguish between multiple jobs listed by the  
31396 *qstat* utility.
- 31397 The **-o** option allows users to redirect the standard output stream. A user might, for example,  
31398 wish to redirect to the null device the standard output stream of a job that produces copious yet  
31399 superfluous output.
- 31400 The **-P** option allows users to designate the relative priority of a batch job for selection from a  
31401 queue.
- 31402 The **-q** option allows users to specify an initial queue for the batch job. If the user specifies a  
31403 routing queue, the batch server routes the batch job to another queue for execution or  
31404 further routing. If the user specifies a non-routing queue, the batch server of the queue  
31405 eventually executes the batch job.
- 31406 The **-r** option allows users to control whether the submitted job will be rerun if the controlling  
31407 batch node fails during execution of the batch job. The **-r** option likewise allows users to  
31408 indicate whether or not the batch job is eligible to be rerun by the *qrerun* utility. Some jobs cannot  
31409 be correctly rerun because of changes they make in the state of databases or other aspects of  
31410 their environment. This volume of IEEE Std 1003.1-200x specifies that the default, if the **-r**  
31411 option is not presented to the utility, will be that the batch job cannot be rerun, since the result of  
31412 rerunning a non-rerunable job might be catastrophic.
- 31413 The **-S** option allows users to specify the program (usually a shell) that will be invoked to  
31414 process the script of the batch job. This option has been modified to allow a list of shell names  
31415 and locations associated with different hosts.
- 31416 The **-u** option is useful when the submitting user is authorized to use more than one account on  
31417 a given host, in which case the **-u** option allows the user to select from among those accounts.  
31418 The option-argument is a list of user-host pairs, so that the submitting user can provide different  
31419 user identifiers for different nodes in the event the batch job is routed. The **-u** option provides a  
31420 lot of flexibility to accommodate sites with complex account structures. Users that have the  
31421 same user identifier on all the hosts they are authorized to use will not need to use the **-u** option.
- 31422 The **-V** option allows users to export all their current environment variables, as of the time the  
31423 batch job is submitted, to the context of the processes of the batch job.
- 31424 The **-v** option allows users to export specific environment variables from their current process  
31425 to the processes of the batch job.
- 31426 The **-z** option allows users to suppress the writing of the batch job identifier to standard output.  
31427 The **-z** option is an existing NQS practice that has been standardized.
- 31428 Historically, the *qsub* utility has served the batch job-submission function in the NQS system, the  
31429 existing practice on which it is based. Some changes and additions have been made to the *qsub*  
31430 utility in this volume of IEEE Std 1003.1-200x, *vis-a-vis* NQS, as a result of the growing pool of  
31431 experience with distributed batch systems.
- 31432 The set of features of the *qsub* utility as defined in this volume of IEEE Std 1003.1-200x appears to  
31433 incorporate all the common existing practice on potentially POSIX-conformant platforms.
- 31434 **FUTURE DIRECTIONS**
- 31435 None.

31436 **SEE ALSO**

31437 *qrerun, qstat, touch*, Chapter 3 (on page 2303)

31438 **CHANGE HISTORY**

31439 Derived from IEEE Std 1003.2d-1994.

31440 **Issue 6**

31441 The **-I** option has been removed as there is no portable description of the resources that are  
31442 allowed or required by the batch job.

31443 **NAME**

31444 read — read a line from standard input

31445 **SYNOPSIS**

31446 read [-r] var...

31447 **DESCRIPTION**31448 The *read* utility shall read a single line from standard input.

31449 By default, unless the *-r* option is specified, backslash ('\*\*') shall act as an escape character, as  
 31450 described in Section 2.2.1 (on page 2232). If standard input is a terminal device and the invoking  
 31451 shell is interactive, *read* shall prompt for a continuation line when:

- 31452 • The shell reads an input line ending with a backslash, unless the *-r* option is specified.
- 31453 • A here-document is not terminated after a <newline> is entered.

31454 The line shall be split into fields as in the shell (see Section 2.6.5 (on page 2243)); the first field  
 31455 shall be assigned to the first variable *var*, the second field to the second variable *var*, and so on. If  
 31456 there are fewer *var* operands specified than there are fields, the leftover fields and their  
 31457 intervening separators shall be assigned to the last *var*. If there are fewer fields than *vars*, the  
 31458 remaining *vars* shall be set to empty strings.

31459 The setting of variables specified by the *var* operands shall affect the current shell execution  
 31460 environment; see Section 2.12 (on page 2263). If it is called in a subshell or separate utility  
 31461 execution environment, such as one of the following:

```
31462 (read foo)
31463 nohup read ...
31464 find . -exec read ... \;
```

31465 it shall not affect the shell variables in the caller's environment.

31466 **OPTIONS**

31467 The *read* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section  
 31468 12.2, Utility Syntax Guidelines.

31469 The following option is supported:

31470 *-r* Do not treat a backslash character in any special way. Consider each backslash to  
 31471 be part of the input line.

31472 **OPERANDS**

31473 The following operand shall be supported:

31474 *var* The name of an existing or nonexisting shell variable.

31475 **STDIN**

31476 The standard input shall be a text file.

31477 **INPUT FILES**

31478 None.

31479 **ENVIRONMENT VARIABLES**31480 The following environment variables shall affect the execution of *read*:

31481 *IFS* Determine the internal field separators used to delimit fields; see Section 2.5.3 (on  
 31482 page 2236).

31483 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 31484 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
 31485 Internationalization Variables for the precedence of internationalization variables

31486 used to determine the values of locale categories.)

31487 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
31488 internationalization variables.

31489 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
31490 characters (for example, single-byte as opposed to multi-byte characters in  
31491 arguments).

31492 **LC\_MESSAGES**  
31493 Determine the locale that should be used to affect the format and contents of  
31494 diagnostic messages written to standard error.

31495 XSI **NLSPATH** Determine the location of message catalogs for the processing of **LC\_MESSAGES**.

31496 **PS2** Provide the prompt string that an interactive shell shall write to standard error  
31497 when a line ending with a backslash is read and the **-r** option was not specified, or  
31498 if a here-document is not terminated after a <newline> is entered.

31499 **ASYNCHRONOUS EVENTS**

31500 Default.

31501 **STDOUT**

31502 Not used.

31503 **STDERR**

31504 The standard error shall be used for diagnostic messages and prompts for continued input. |

31505 **OUTPUT FILES**

31506 None.

31507 **EXTENDED DESCRIPTION**

31508 None.

31509 **EXIT STATUS**

31510 The following exit values shall be returned:

31511 0 Successful completion.

31512 >0 End-of-file was detected or an error occurred.

31513 **CONSEQUENCES OF ERRORS**

31514 Default.

31515 **APPLICATION USAGE**

31516 The **-r** option is included to enable *read* to subsume the purpose of the *line* utility, which is not  
31517 included in IEEE Std 1003.1-200x.

31518 The results are undefined if an end-of-file is detected following a backslash at the end of a line  
31519 when **-r** is not specified.

31520 **EXAMPLES**

31521 The following command:

```
31522 while read -r xx yy
31523 do
31524     printf "%s %s\n" "$yy" "$xx"
31525 done < input_file
```

31526 prints a file with the first field of each line moved to the end of the line.

31527 **RATIONALE**

31528           The *read* utility historically has been a shell built-in. It was separated off into its own utility to  
31529           take advantage of the richer description of functionality introduced by this volume of  
31530           IEEE Std 1003.1-200x.

31531           Since *read* affects the current shell execution environment, it is generally provided as a shell  
31532           regular built-in. If it is called in a subshell or separate utility execution environment, such as one  
31533           of the following:

```
31534           (read foo)
31535           nohup read ...
31536           find . -exec read ... \;
```

31537           it does not affect the shell variables in the environment of the caller.

31538 **FUTURE DIRECTIONS**

31539           None.

31540 **SEE ALSO**

31541           None.

31542 **CHANGE HISTORY**

31543           First released in Issue 2.

## 31544 NAME

31545 renice — set nice values of running processes

## 31546 SYNOPSIS

31547 UP `renice -n increment [-g | -p | -u] ID ...`

31548

## 31549 DESCRIPTION

31550 The *renice* utility shall request that the nice values (see the Base Definitions volume of  
 31551 IEEE Std 1003.1-200x, Section 3.239, Nice Value) of one or more running processes be changed.  
 31552 By default, the applicable processes are specified by their process IDs. When a process group is  
 31553 specified (see `-g`), the request shall apply to all processes in the process group.

31554 The nice value shall be bounded in an implementation-defined manner. If the requested  
 31555 *increment* would raise or lower the nice value of the executed utility beyond implementation-  
 31556 defined limits, then the limit whose value was exceeded shall be used.

31557 When a user is *reniced*, the request applies to all processes whose saved set-user-ID matches the  
 31558 user ID corresponding to the user.

31559 Regardless of which options are supplied or any other factor, *renice* shall not alter the nice values  
 31560 of any process unless the user requesting such a change has appropriate privileges to do so for  
 31561 the specified process. If the user lacks appropriate privileges to perform the requested action, the  
 31562 utility shall return an error status.

31563 The saved set-user-ID of the user's process shall be checked instead of its effective user ID when  
 31564 *renice* attempts to determine the user ID of the process in order to determine whether the user  
 31565 has appropriate privileges.

## 31566 OPTIONS

31567 The *renice* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section  
 31568 12.2, Utility Syntax Guidelines.

31569 The following options shall be supported:

31570 `-g` Interpret all operands as unsigned decimal integer process group IDs.

31571 `-n increment` Specify how the nice value of the specified process or processes is to be adjusted.  
 31572 The *increment* option-argument is a positive or negative decimal integer that shall  
 31573 be used to modify the nice value of the specified process or processes.

31574 Positive *increment* values shall cause a lower nice value. Negative *increment* values  
 31575 may require appropriate privileges and shall cause a higher nice value.

31576 `-p` Interpret all operands as unsigned decimal integer process IDs. The `-p` option is  
 31577 the default if no options are specified.

31578 `-u` Interpret all operands as users. If a user exists with a user name equal to the  
 31579 operand, then the user ID of that user is used in further processing. Otherwise, if  
 31580 the operand represents an unsigned decimal integer, it shall be used as the numeric  
 31581 user ID of the user.

## 31582 OPERANDS

31583 The following operands shall be supported:

31584 *ID* A process ID, process group ID, or user name/user ID, depending on the option  
 31585 selected.

31586 **STDIN**

31587 Not used.

31588 **INPUT FILES**

31589 None.

31590 **ENVIRONMENT VARIABLES**31591 The following environment variables shall affect the execution of *renice*:

31592 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 31593 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
 31594 Internationalization Variables for the precedence of internationalization variables  
 31595 used to determine the values of locale categories.)

31596 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 31597 internationalization variables.

31598 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 31599 characters (for example, single-byte as opposed to multi-byte characters in  
 31600 arguments).

31601 *LC\_MESSAGES*

31602 Determine the locale that should be used to affect the format and contents of  
 31603 diagnostic messages written to standard error.

31604 *NSI* *NLS\_PATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

31605 **ASYNCHRONOUS EVENTS**

31606 Default.

31607 **STDOUT**

31608 Not used.

31609 **STDERR**

31610 The standard error shall be used only for diagnostic messages. |

31611 **OUTPUT FILES**

31612 None.

31613 **EXTENDED DESCRIPTION**

31614 None.

31615 **EXIT STATUS**

31616 The following exit values shall be returned:

31617 0 Successful completion.

31618 &gt;0 An error occurred.

31619 **CONSEQUENCES OF ERRORS**

31620 Default.

31621 **APPLICATION USAGE**

31622 None.

31623 **EXAMPLES**

31624 1. Adjust the nice value so that process IDs 987 and 32 would have a lower nice value:

31625 `renice -n 5 -p 987 32`31626 2. Adjust the nice value so that group IDs 324 and 76 would have a higher nice value, if the  
31627 user has the appropriate privileges to do so:31628 `renice -n -4 -g 324 76`31629 3. Adjust the nice value so that numeric user ID 8 and user **sas** would have a lower nice  
31630 value:31631 `renice -n 4 -u 8 sas`31632 Useful nice value increments on historical systems include 19 or 20 (the affected processes run  
31633 only when nothing else in the system attempts to run) and any negative number (to make  
31634 processes run faster).31635 **RATIONALE**31636 The *gid*, *pid*, and *user* specifications do not fit either the definition of operand or option-  
31637 argument. However, for clarity, they have been included in the OPTIONS section, rather than  
31638 the OPERANDS section.31639 The definition of nice value is not intended to suggest that all processes in a system have  
31640 priorities that are comparable. Scheduling policy extensions such as the realtime priorities in the  
31641 System Interfaces volume of IEEE Std 1003.1-200x make the notion of a single underlying  
31642 priority for all scheduling policies problematic. Some implementations may implement the *nice*-  
31643 related features to affect all processes on the system, others to affect just the general time-  
31644 sharing activities implied by this volume of IEEE Std 1003.1-200x, and others may have no effect  
31645 at all. Because of the use of “implementation-defined” in *nice* and *renice*, a wide range of  
31646 implementation strategies are possible.31647 Originally, this utility was written in the historical manner, using the term “nice value”. This  
31648 was always a point of concern with users because it was never intuitively obvious what this  
31649 meant. With a newer version of *renice*, which used the term “system scheduling priority”, it was  
31650 hoped that novice users could better understand what this utility was meant to do. Also, it  
31651 would be easier to document what the utility was meant to do. Unfortunately, the addition of  
31652 the POSIX realtime scheduling capabilities introduced the concepts of process and thread  
31653 scheduling priorities that were totally unaffected by the *nice/renice* utilities or the  
31654 *nice()/setpriority()* functions. Continuing to use the term “system scheduling priority” would  
31655 have incorrectly suggested that these utilities and functions were indeed affecting these realtime  
31656 priorities. It was decided to revert to the historical term “nice value” to reference this unrelated  
31657 process attribute.31658 Although this utility has use by system administrators (and in fact appears in the system  
31659 administration portion of the BSD documentation), the standard developers considered that it  
31660 was very useful for individual end users to control their own processes.31661 **FUTURE DIRECTIONS**

31662 None.

31663 **SEE ALSO**31664 *nice*31665 **CHANGE HISTORY**

31666 First released in Issue 4.

31667 **Issue 5**31668 In the SYNOPSIS, an ellipsis is added to the `-u` option in all three obsolescent forms.31669 **Issue 6**

31670 This utility is now marked as part of the User Portability Utilities option.

31671 The APPLICATION USAGE section is added.

31672 The obsolescent forms of the SYNOPSIS are removed.

31673 Text previously conditional on `POSIX_SAVED_IDS` is mandatory in this issue. This is a FIPS  
31674 requirement.

## 31675 NAME

31676 rm — remove directory entries

## 31677 SYNOPSIS

31678 rm [-fiRr] *file*...

## 31679 DESCRIPTION

31680 The *rm* utility shall remove the directory entry specified by each *file* argument.

31681 If either of the files dot or dot-dot are specified as the basename portion of an operand (that is,  
 31682 the final pathname component), *rm* shall write a diagnostic message to standard error and do  
 31683 nothing more with such operands.

31684 For each *file* the following steps shall be taken:

- 31685 1. If the *file* does not exist:
    - 31686 a. If the **-f** option is not specified, *rm* shall write a diagnostic message to standard error. |
    - 31687 b. Go on to any remaining *files*.
  - 31688 2. If *file* is of type directory, the following steps shall be taken:
    - 31689 a. If neither the **-R** option nor the **-r** option is specified, *rm* shall write a diagnostic |
    - 31690 message to standard error, do nothing more with *file*, and go on to any remaining |
    - 31691 files.
    - 31692 b. If the **-f** option is not specified, and either the permissions of *file* do not permit |
    - 31693 writing and the standard input is a terminal or the **-i** option is specified, *rm* shall |
    - 31694 write a prompt to standard error and read a line from the standard input. If the |
    - 31695 response is not affirmative, *rm* shall do nothing more with the current file and go on |
    - 31696 to any remaining files.
    - 31697 c. For each entry contained in *file*, other than dot or dot-dot, the four steps listed here (1 |
    - 31698 to 4) shall be taken with the entry as if it were a *file* operand. The *rm* utility shall not |
    - 31699 traverse directories by following symbolic links into other parts of the hierarchy, but |
    - 31700 shall remove the links themselves.
    - 31701 d. If the **-i** option is specified, *rm* shall write a prompt to standard error and read a line |
    - 31702 from the standard input. If the response is not affirmative, *rm* shall do nothing more |
    - 31703 with the current file, and go on to any remaining files.
  - 31704 3. If *file* is not of type directory, the **-f** option is not specified, and either the permissions of |
  - 31705 *file* do not permit writing and the standard input is a terminal or the **-i** option is specified, |
  - 31706 *rm* shall write a prompt to the standard error and read a line from the standard input. If the |
  - 31707 response is not affirmative, *rm* shall do nothing more with the current file and go on to any |
  - 31708 remaining files.
  - 31709 4. If the current file is a directory, *rm* shall perform actions equivalent to the *rmdir*() function |
  - 31710 defined in the System Interfaces volume of IEEE Std 1003.1-200x called with a pathname of |
  - 31711 the current file used as the *path* argument. If the current file is not a directory, *rm* shall |
  - 31712 perform actions equivalent to the *unlink*() function defined in the System Interfaces |
  - 31713 volume of IEEE Std 1003.1-200x called with a pathname of the current file used as the *path* |
  - 31714 argument.
- 31715 If this fails for any reason, *rm* shall write a diagnostic message to standard error, do  
 31716 nothing more with the current file, and go on to any remaining files.

31717 The *rm* utility shall be able to descend to arbitrary depths in a file hierarchy, and shall not fail  
 31718 due to path length limitations (unless an operand specified by the user exceeds system

31719 limitations).

### 31720 OPTIONS

31721 The *rm* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2,  
31722 Utility Syntax Guidelines.

31723 The following options shall be supported:

31724 **-f** Do not prompt for confirmation. Do not write diagnostic messages or modify the  
31725 exit status in the case of nonexistent operands. Any previous occurrences of the **-i**  
31726 option shall be ignored.

31727 **-i** Prompt for confirmation as described previously. Any previous occurrences of the  
31728 **-f** option shall be ignored.

31729 **-R** Remove file hierarchies. See the DESCRIPTION.

31730 **-r** Equivalent to **-R**.

### 31731 OPERANDS

31732 The following operand shall be supported:

31733 *file* A pathname of a directory entry to be removed.

### 31734 STDIN

31735 The standard input shall be used to read an input line in response to each prompt specified in |  
31736 the STDOUT section. Otherwise, the standard input shall not be used. |

### 31737 INPUT FILES

31738 None.

### 31739 ENVIRONMENT VARIABLES

31740 The following environment variables shall affect the execution of *rm*:

31741 *LANG* Provide a default value for the internationalization variables that are unset or null.  
31742 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
31743 Internationalization Variables for the precedence of internationalization variables  
31744 used to determine the values of locale categories.)

31745 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
31746 internationalization variables.

31747 *LC\_COLLATE*

31748 Determine the locale for the behavior of ranges, equivalence classes, and multi-  
31749 character collating elements used in the extended regular expression defined for  
31750 the **yesexpr** locale keyword in the *LC\_MESSAGES* category.

31751 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
31752 characters (for example, single-byte as opposed to multi-byte characters in  
31753 arguments) and the behavior of character classes within regular expressions used  
31754 in the extended regular expression defined for the **yesexpr** locale keyword in the  
31755 *LC\_MESSAGES* category.

31756 *LC\_MESSAGES*

31757 Determine the locale for the processing of affirmative responses that should be  
31758 used to affect the format and contents of diagnostic messages written to standard  
31759 error.

31760 *XSI* *NLS\_PATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

31761 **ASYNCHRONOUS EVENTS**

31762 Default.

31763 **STDOUT**

31764 Not used.

31765 **STDERR**

31766 Prompts shall be written to standard error under the conditions specified in the DESCRIPTION and OPTIONS sections. The prompts shall contain the *file* pathname, but their format is otherwise unspecified. The standard error also shall be used for diagnostic messages.

31769 **OUTPUT FILES**

31770 None.

31771 **EXTENDED DESCRIPTION**

31772 None.

31773 **EXIT STATUS**

31774 The following exit values shall be returned:

31775 0 All of the named directory entries for which *rm* performed actions equivalent to *rmdir()* or *unlink()* functions were removed.

31777 &gt;0 An error occurred.

31778 **CONSEQUENCES OF ERRORS**

31779 Default.

31780 **APPLICATION USAGE**

31781 The *rm* utility is forbidden to remove the names dot and dot-dot in order to avoid the consequences of inadvertently doing something like:

31783 `rm -r .*`

31784 Some implementations do not permit the removal of the last link to an executable binary file that is being executed; see the [EBUSY] error in the *unlink()* function defined in the System Interfaces volume of IEEE Std 1003.1-200x. Thus, the *rm* utility can fail to remove such files.

31787 The *-i* option causes *rm* to prompt and read the standard input even if the standard input is not a terminal, but in the absence of *-i* the mode prompting is not done when the standard input is not a terminal.

31790 **EXAMPLES**

31791 1. The following command:

31792 `rm a.out core`31793 removes the directory entries: **a.out** and **core**.

31794 2. The following command:

31795 `rm -Rf junk`31796 removes the directory **junk** and all its contents, without prompting.31797 **RATIONALE**

31798 For absolute clarity, paragraphs (2b) and (3) in the DESCRIPTION of *rm* describing the behavior when prompting for confirmation, should be interpreted in the following manner:

31800 `if ((NOT f_option) AND`31801 `((not_writable AND input_is_terminal) OR i_option))`

31802 The exact format of the interactive prompts is unspecified. Only the general nature of the  
31803 contents of prompts are specified because implementations may desire more descriptive  
31804 prompts than those used on historical implementations. Therefore, an application not using the  
31805 `-f` option, or using the `-i` option, relies on the system to provide the most suitable dialog directly  
31806 with the user, based on the behavior specified.

31807 The `-r` option is historical practice on all known systems. The synonym `-R` option is provided  
31808 for consistency with the other utilities in this volume of IEEE Std 1003.1-200x that provide  
31809 options requesting recursive descent through the file hierarchy.

31810 The behavior of the `-f` option in historical versions of *rm* is inconsistent. In general, along with  
31811 “forcing” the unlink without prompting for permission, it always causes diagnostic messages to  
31812 be suppressed and the exit status to be unmodified for nonexistent operands and files that  
31813 cannot be unlinked. In some versions, however, the `-f` option suppresses usage messages and  
31814 system errors as well. Suppressing such messages is not a service to either shell scripts or users.

31815 It is less clear that error messages regarding files that cannot be unlinked (removed) should be  
31816 suppressed. Although this is historical practice, this volume of IEEE Std 1003.1-200x does not  
31817 permit the `-f` option to suppress such messages.

31818 When given the `-r` and `-i` options, historical versions of *rm* prompt the user twice for each  
31819 directory, once before removing its contents and once before actually attempting to delete the  
31820 directory entry that names it. This allows the user to “prune” the file hierarchy walk. Historical  
31821 versions of *rm* were inconsistent in that some did not do the former prompt for directories  
31822 named on the command line and others had obscure prompting behavior when the `-i` option  
31823 was specified and the permissions of the file did not permit writing. The POSIX Shell and  
31824 Utilities *rm* differs little from historic practice, but does require that prompts be consistent.  
31825 Historical versions of *rm* were also inconsistent in that prompts were done to both standard  
31826 output and standard error. This volume of IEEE Std 1003.1-200x requires that prompts be done  
31827 to standard error, for consistency with *cp* and *mv*, and to allow historical extensions to *rm* that  
31828 provide an option to list deleted files on standard output.

31829 The *rm* utility is required to descend to arbitrary depths so that any file hierarchy may be  
31830 deleted. This means, for example, that the *rm* utility cannot run out of file descriptors during its  
31831 descent (that is, if the number of file descriptors is limited, *rm* cannot be implemented in the  
31832 historical fashion where one file descriptor is used per directory level). Also, *rm* is not permitted  
31833 to fail because of path length restrictions, unless an operand specified by the user is longer than  
31834 `{PATH_MAX}`.

31835 The *rm* utility removes symbolic links themselves, not the files they refer to, as a consequence of  
31836 the dependence on the *unlink()* functionality, per the DESCRIPTION. When removing  
31837 hierarchies with `-r` or `-R`, the prohibition on following symbolic links has to be made explicit.

#### 31838 FUTURE DIRECTIONS

31839 None.

#### 31840 SEE ALSO

31841 *rmdir*, the System Interfaces volume of IEEE Std 1003.1-200x, *remove()*, *unlink()*

#### 31842 CHANGE HISTORY

31843 First released in Issue 2.

#### 31844 Issue 5

31845 FUTURE DIRECTIONS section added.

31846 **Issue 6**

31847

31848

Text is added to clarify actions relating to symbolic links as specified in the IEEE P1003.2b draft standard.

31849 **NAME**31850 rmdel — remove a delta from an SCCS file (**DEVELOPMENT**)31851 **SYNOPSIS**31852 xSI `rmdel -r SID file...`

31853

31854 **DESCRIPTION**

31855 The *rmdel* utility shall remove the delta specified by the SID from each named SCCS file. The  
 31856 delta to be removed shall be the most recent delta in its branch in the delta chain of each named  
 31857 SCCS file. In addition, the application shall ensure that the SID specified is not that of a version  
 31858 being edited for the purpose of making a delta; that is, if a *p-file* (see *get* (on page 2675)) exists for  
 31859 the named SCCS file, the SID specified shall not appear in any entry of the *p-file*.

31860 Removal of a delta shall be restricted to:

- 31861 1. The user who made the delta
- 31862 2. The owner of the SCCS file
- 31863 3. The owner of the directory containing the SCCS file

31864 **OPTIONS**

31865 The *rmdel* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section  
 31866 12.2, Utility Syntax Guidelines.

31867 The following option shall be supported:

31868 **-r** *SID* Specify the SCCS identification string (*SID*) of the delta to be deleted.31869 **OPERANDS**

31870 The following operand shall be supported:

31871 *file* A pathname of an existing SCCS file or a directory. If *file* is a directory, the *rmdel*  
 31872 utility shall behave as though each file in the directory were specified as a named  
 31873 file, except that non-SCCS files (last component of the pathname does not begin  
 31874 with *s.*) and unreadable files shall be silently ignored.

31875 If exactly one *file* operand appears, and it is *'-'*, the standard input shall be read;  
 31876 each line of the standard input is taken to be the name of an SCCS file to be  
 31877 processed. Non-SCCS files and unreadable files shall be silently ignored.

31878 **STDIN**

31879 The standard input shall be a text file used only when the *file* operand is specified as *'-'*. Each  
 31880 line of the text file shall be interpreted as an SCCS pathname.

31881 **INPUT FILES**

31882 The SCCS files shall be files of unspecified format.

31883 **ENVIRONMENT VARIABLES**31884 The following environment variables shall affect the execution of *rmdel*:

31885 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 31886 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
 31887 Internationalization Variables for the precedence of internationalization variables  
 31888 used to determine the values of locale categories.)

31889 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 31890 internationalization variables.

- 31891 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
31892 characters (for example, single-byte as opposed to multi-byte characters in  
31893 arguments and input files).
- 31894 *LC\_MESSAGES*  
31895 Determine the locale that should be used to affect the format and contents of  
31896 diagnostic messages written to standard error.
- 31897 *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 31898 **ASYNCHRONOUS EVENTS**  
31899 Default.
- 31900 **STDOUT**  
31901 Not used.
- 31902 **STDERR**  
31903 The standard error shall be used only for diagnostic messages. |
- 31904 **OUTPUT FILES**  
31905 The SCCS files shall be files of unspecified format. During processing of a *file*, a temporary *x-file*, |  
31906 as described in *admin* (on page 2328), may be created and deleted; a locking *z-file*, as described in  
31907 *get* (on page 2675), may be created and deleted.
- 31908 **EXTENDED DESCRIPTION**  
31909 None.
- 31910 **EXIT STATUS**  
31911 The following exit values shall be returned:  
31912 0 Successful completion.  
31913 >0 An error occurred.
- 31914 **CONSEQUENCES OF ERRORS**  
31915 Default.
- 31916 **APPLICATION USAGE**  
31917 None.
- 31918 **EXAMPLES**  
31919 None.
- 31920 **RATIONALE**  
31921 None.
- 31922 **FUTURE DIRECTIONS**  
31923 None.
- 31924 **SEE ALSO**  
31925 *delta, get, prs*
- 31926 **CHANGE HISTORY**  
31927 First released in Issue 2.
- 31928 **Issue 6**  
31929 The normative text is reworded to avoid use of the term “must” for application requirements.  
31930 The normative text is reworded to emphasize the term “shall” for implementation requirements.

31931 **NAME**31932            **rmdir** — remove directories31933 **SYNOPSIS**31934            **rmdir** [-p] *dir*...31935 **DESCRIPTION**31936            The *rmdir* utility shall remove the directory entry specified by each *dir* operand, which, in order  
31937            to succeed, the application shall ensure refers to an empty directory.31938            Directories shall be processed in the order specified. If a directory and a subdirectory of that  
31939            directory are specified in a single invocation of the *rmdir* utility, the application shall specify the  
31940            subdirectory before the parent directory so that the parent directory will be empty when the  
31941            *rmdir* utility tries to remove it.31942 **OPTIONS**31943            The *rmdir* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section  
31944            12.2, Utility Syntax Guidelines.

31945            The following option shall be supported:

31946            **-p**            Remove all directories in a pathname. For each *dir* operand:

- 31947                            1. The directory entry it names shall be removed.
- 
- 31948                            2. If the
- dir*
- operand includes more than one pathname component, effects
- 
- 31949                            equivalent to the following command shall occur:

31950                            **rmdir -p \$(dirname *dir*)**31951 **OPERANDS**

31952            The following operand shall be supported:

31953            ***dir***            A pathname of an empty directory to be removed.31954 **STDIN**

31955            Not used.

31956 **INPUT FILES**

31957            None.

31958 **ENVIRONMENT VARIABLES**31959            The following environment variables shall affect the execution of *rmdir*:31960            **LANG**            Provide a default value for the internationalization variables that are unset or null.  
31961                            (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
31962                            Internationalization Variables for the precedence of internationalization variables  
31963                            used to determine the values of locale categories.)31964            **LC\_ALL**        If set to a non-empty string value, override the values of all the other  
31965                            internationalization variables.31966            **LC\_CTYPE**     Determine the locale for the interpretation of sequences of bytes of text data as  
31967                            characters (for example, single-byte as opposed to multi-byte characters in  
31968                            arguments).31969            **LC\_MESSAGES**31970                            Determine the locale that should be used to affect the format and contents of  
31971                            diagnostic messages written to standard error.31972 **XSI**            **NLS\_PATH**     Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

31973 **ASYNCHRONOUS EVENTS**

31974 Default.

31975 **STDOUT**

31976 Not used.

31977 **STDERR**

31978 The standard error shall be used only for diagnostic messages.

31979 **OUTPUT FILES**

31980 None.

31981 **EXTENDED DESCRIPTION**

31982 None.

31983 **EXIT STATUS**

31984 The following exit values shall be returned:

31985 0 Each directory entry specified by a *dir* operand was removed successfully.

31986 &gt;0 An error occurred.

31987 **CONSEQUENCES OF ERRORS**

31988 Default.

31989 **APPLICATION USAGE**31990 The definition of an empty directory is one that contains, at most, directory entries for dot and dot-dot.  
3199131992 **EXAMPLES**31993 If a directory **a** in the current directory is empty except it contains a directory **b** and **a/b** is empty  
31994 except it contains a directory **c**:31995 `rmdir -p a/b/c`

31996 removes all three directories.

31997 **RATIONALE**31998 On historical System V systems, the `-p` option also caused a message to be written to the  
31999 standard output. The message indicated whether the whole path was removed or whether part  
32000 of the path remained for some reason. The **STDERR** section requires this diagnostic when the  
32001 entire path specified by a *dir* operand is not removed, but does not allow the status message  
32002 reporting success to be written as a diagnostic.32003 The *rmdir* utility on System V also included an `-s` option that suppressed the informational  
32004 message output by the `-p` option. This option has been omitted because the informational  
32005 message is not specified by this volume of IEEE Std 1003.1-200x.32006 **FUTURE DIRECTIONS**

32007 None.

32008 **SEE ALSO**32009 *rm*, the System Interfaces volume of IEEE Std 1003.1-200x, *remove()*, *rmdir()*, *unlink()*32010 **CHANGE HISTORY**

32011 First released in Issue 2.

32012 **Issue 6**

32013 The normative text is reworded to avoid use of the term “must” for application requirements.

32014 **NAME**32015 `sact` — print current SCCS file-editing activity (**DEVELOPMENT**)32016 **SYNOPSIS**32017 XSI `sact file...`

32018

32019 **DESCRIPTION**

32020 The *sact* utility shall inform the user of any impending deltas to a named SCCS file by writing a  
 32021 list to standard output. This situation occurs when *get -e* has been executed previously without  
 32022 a subsequent execution of *delta*, *unget*, or *sccs unedit*.

32023 **OPTIONS**

32024 None.

32025 **OPERANDS**

32026 The following operand shall be supported:

32027 *file* A pathname of an existing SCCS file or a directory. If *file* is a directory, the *sact*  
 32028 utility shall behave as though each file in the directory were specified as a named  
 32029 file, except that non-SCCS files (last component of the pathname does not begin  
 32030 with *s.*) and unreadable files shall be silently ignored.

32031 If exactly one *file* operand appears, and it is *'-'*, the standard input shall be read;  
 32032 each line of the standard input shall be taken to be the name of an SCCS file to be  
 32033 processed. Non-SCCS files and unreadable files shall be silently ignored.

32034 **STDIN**

32035 The standard input shall be a text file used only when the *file* operand is specified as *'-'*. Each  
 32036 line of the text file shall be interpreted as an SCCS pathname.

32037 **INPUT FILES**

32038 Any SCCS files interrogated are files of an unspecified format.

32039 **ENVIRONMENT VARIABLES**32040 The following environment variables shall affect the execution of *sact*:

32041 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 32042 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
 32043 Internationalization Variables for the precedence of internationalization variables  
 32044 used to determine the values of locale categories.)

32045 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 32046 internationalization variables.

32047 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 32048 characters (for example, single-byte as opposed to multi-byte characters in  
 32049 arguments and input files).

32050 *LC\_MESSAGES*

32051 Determine the locale that should be used to affect the format and contents of  
 32052 diagnostic messages written to standard error.

32053 *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

32054 **ASYNCHRONOUS EVENTS**

32055 Default.

32056 **STDOUT**

32057 The output for each named file shall consist of a line in the following format:

32058 "%sΔ%sΔ%sΔ%sΔ%s\n", <SID>, <new SID>, <login>, <date>, <time>

32059 <SID> Specifies the SID of a delta that currently exists in the SCCS file to which changes  
32060 are made to make the new delta.

32061 <new SID> Specifies the SID for the new delta to be created.

32062 <login> Contains the login name of the user who makes the delta (that is, who executed a  
32063 *get* for editing).

32064 <date> Contains the date that *get -e* was executed, in the format used by the *prs :D:* data  
32065 keyword.

32066 <time> Contains the time that *get -e* was executed, in the format used by the *prs :T:* data  
32067 keyword.

32068 If there is more than one named file or if a directory or standard input is named, each pathname  
32069 shall be written before each of the preceding lines:

32070 "\n%s:\n", <pathname>

32071 **STDERR**

32072 The standard error shall be used only for optional informative messages concerning SCCS files |  
32073 with no impending deltas, and for diagnostic messages. |

32074 **OUTPUT FILES**

32075 None.

32076 **EXTENDED DESCRIPTION**

32077 None.

32078 **EXIT STATUS**

32079 The following exit values shall be returned:

32080 0 Successful completion.

32081 >0 An error occurred.

32082 **CONSEQUENCES OF ERRORS**

32083 Default.

32084 **APPLICATION USAGE**

32085 None.

32086 **EXAMPLES**

32087 None.

32088 **RATIONALE**

32089 None.

32090 **FUTURE DIRECTIONS**

32091 None.

32092 **SEE ALSO**

32093 *delta, get, unget*

32094 **CHANGE HISTORY**

32095 First released in Issue 2.

32096 **Issue 6**

32097 The normative text is reworded to emphasize the term “shall” for implementation requirements.

## 32098 NAME

32099 `sccs` — front end for the SCCS subsystem (**DEVELOPMENT**)

## 32100 SYNOPSIS

32101 xSI `sccs [-r][-d path][-p path] command [options...][operands...]`

32102

## 32103 DESCRIPTION

32104 The `sccs` utility is a front end to the SCCS programs. It also includes the capability to run set-  
32105 user-id to another user to provide additional protection.32106 The `sccs` utility shall invoke the specified *command* with the specified *options* and *operands*. By  
32107 default, each of the *operands* shall be modified by prefixing it with the string "SCCS/s. ".32108 The *command* can be the name of one of the SCCS utilities in this volume of IEEE Std 1003.1-200x  
32109 (*admin*, *delta*, *get*, *prs*, *rmdel*, *sact*, *unget*, *val*, or *what*) or one of the pseudo-utilities listed in the  
32110 EXTENDED DESCRIPTION section.

## 32111 OPTIONS

32112 The `sccs` utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section  
32113 12.2, Utility Syntax Guidelines, except that *options* operands are actually options to be passed to  
32114 the utility named by *command*. When the portion of the command:32115 `command [options ... ] [operands ... ]`32116 is considered, all of the pseudo-utilities used as *command* shall support the Utility Syntax  
32117 Guidelines. Any of the other SCCS utilities that can be invoked in this manner support the  
32118 Guidelines to the extent indicated by their individual OPTIONS sections.32119 The following options shall be supported preceding the *command* operand:32120 **-d path** A pathname of a directory to be used as a root directory for the SCCS files. The |  
32121 default shall be the current directory. The **-d** option shall take precedence over the |  
32122 *PROJECTDIR* variable. See **-p**.32123 **-p path** A pathname of a directory in which the SCCS files are located. The default shall be |  
32124 the **SCCS** directory. |32125 The **-p** option differs from the **-d** option in that the **-d** option-argument shall be |  
32126 prefixed to the entire pathname and the **-p** option-argument shall be inserted |  
32127 before the final component of the pathname. For example: |32128 `sccs -d /x -p y get a/b`

32129 converts to:

32130 `get /x/a/y/s.b`

32131 This allows the creation of aliases such as:

32132 `alias syssccs="sccs -d /usr/src"`

32133 which is used as:

32134 `syssccs get cmd/who.c`32135 **-r** Invoke *command* with the real user ID of the process, not any effective user ID that  
32136 the `sccs` utility is set to. Certain commands (*admin*, **check**, **clean**, **diffs**, **info**, *rmdel*,  
32137 and **tell**) cannot be run set-user-ID by all users, since this would allow anyone to  
32138 change the authorizations. These commands are always run as the real user.

32139 **OPERANDS**

32140 The following operands shall be supported:

32141 *command* An SCCS utility name or the name of one of the pseudo-utilities listed in the  
32142 EXTENDED DESCRIPTION section.

32143 *options* An option or option-argument to be passed to *command*.

32144 *operands* An operand to be passed to *command*.

32145 **STDIN**

32146 See the utility description for the specified *command*.

32147 **INPUT FILES**

32148 See the utility description for the specified *command*.

32149 **ENVIRONMENT VARIABLES**

32150 The following environment variables shall affect the execution of *sccs*:

32151 *LANG* Provide a default value for the internationalization variables that are unset or null.  
32152 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
32153 Internationalization Variables for the precedence of internationalization variables  
32154 used to determine the values of locale categories.)

32155 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
32156 internationalization variables.

32157 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
32158 characters (for example, single-byte as opposed to multi-byte characters in  
32159 arguments and input files).

32160 *LC\_MESSAGES*

32161 Determine the locale that should be used to affect the format and contents of  
32162 diagnostic messages written to standard error.

32163 *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

32164 *PROJECTDIR*

32165 Provide a default value for the *-d path* option. If the value of *PROJECTDIR* begins  
32166 with a slash, it shall be considered an absolute pathname; otherwise, the value of  
32167 *PROJECTDIR* is treated as a user name and that user's initial working directory  
32168 shall be examined for a subdirectory *src* or *source*. If such a directory is found, it  
32169 shall be used. Otherwise, the value shall be used as a relative pathname.

32170 Additional environment variable effects may be found in the utility description for the specified  
32171 *command*.

32172 **ASYNCHRONOUS EVENTS**

32173 Default.

32174 **STDOUT**

32175 See the utility description for the specified *command*.

32176 **STDERR**

32177 See the utility description for the specified *command*.

32178 **OUTPUT FILES**

32179 See the utility description for the specified *command*.

## 32180 EXTENDED DESCRIPTION

32181 The following pseudo-utilities shall be supported as *command* operands. All options referred to  
 32182 in the following list are values given in the *options* operands following *command*.

32183 **check** Equivalent to **info**, except that nothing shall be printed if nothing is being edited, and a  
 32184 non-zero exit status shall be returned if anything is being edited. The intent is to have  
 32185 this included in an “install” entry in a makefile to ensure that everything is included  
 32186 into the SCCS file before a version is installed.

32187 **clean** Remove everything from the current directory that can be recreated from SCCS files,  
 32188 but do not remove any files being edited. If the **-b** option is given, branches shall be  
 32189 ignored in the determination of whether they are being edited; this is dangerous if  
 32190 branches are kept in the same directory.

32191 **create** Create an SCCS file, taking the initial contents from the file of the same name. Any  
 32192 options to *admin* are accepted. If the creation is successful, the original files shall be  
 32193 renamed by prefixing the basenames with a comma. These renamed files should be  
 32194 removed after it has been verified that the SCCS files have been created successfully.

32195 **delget** Perform a *delta* on the named files and then *get* new versions. The new versions shall  
 32196 have ID keywords expanded and shall not be editable. Any **-m**, **-p**, **-r**, **-s**, and **-y**  
 32197 options shall be passed to *delta*, and any **-b**, **-c**, **-e**, **-i**, **-k**, **-l**, **-s**, and **-x** options shall be  
 32198 passed to *get*.

32199 **deledit** Equivalent to **delget**, except that the *get* phase shall include the **-e** option. This option  
 32200 is useful for making a checkpoint of the current editing phase. The same options shall  
 32201 be passed to *delta* as described above, and all the options listed for *get* above except **-e**  
 32202 shall be passed to **edit**.

32203 **diffs** Write a difference listing between the current version of the files checked out for  
 32204 editing and the versions in SCCS format. Any **-r**, **-c**, **-i**, **-x**, and **-t** options shall be  
 32205 passed to *get*; any **-l**, **-s**, **-e**, **-f**, **-h**, and **-b** options shall be passed to *diff*. A **-C** option  
 32206 shall be passed to *diff* as **-c**.

32207 **edit** Equivalent to *get -e*.

32208 **fix** Remove the named delta, but leave a copy of the delta with the changes that were in it.  
 32209 It is useful for fixing small compiler bugs, and so on. The application shall ensure that it  
 32210 is followed by a **-r** *SID* option. Since **fix** does not leave audit trails, it should be used  
 32211 carefully.

32212 **info** Write a listing of all files being edited. If the **-b** option is given, branches (that is, SIDs  
 32213 with two or fewer components) shall be ignored. If a **-u** *user* option is given, then only  
 32214 files being edited by the named user shall be listed. A **-U** option shall be equivalent to  
 32215 **-u**<*current user*>.

32216 **print** Write out verbose information about the named files, equivalent to *sccs prs*.

32217 **tell** Write a <newline>-separated list of the files being edited to standard output. Takes the  
 32218 **-b**, **-u**, and **-U** options like **info** and **check**.

32219 **unedit** This is the opposite of an **edit** or a *get -e*. It should be used with caution, since any  
 32220 changes made since the *get* are lost.

## 32221 EXIT STATUS

32222 The following exit values shall be returned:

32223 0 Successful completion.

32224 >0 An error occurred.

### 32225 CONSEQUENCES OF ERRORS

32226 Default.

### 32227 APPLICATION USAGE

32228 Many of the SCCS utilities take directory names as operands as well as specific filenames. The  
 32229 pseudo-utilities supported by `sccs` are not described as having this capability, but are not  
 32230 prohibited from doing so.

### 32231 EXAMPLES

32232 1. To get a file for editing, edit it and produce a new delta:

```
32233 sccs get -e file.c
```

```
32234 ex file.c
```

```
32235 sccs delta file.c
```

32236 2. To get a file from another directory:

```
32237 sccs -p /usr/src/sccs/s. get cc.c
```

32238 or:

```
32239 sccs get /usr/src/sccs/s.cc.c
```

32240 3. To make a delta of a large number of files in the current directory:

```
32241 sccs delta *.c
```

32242 4. To get a list of files being edited that are not on branches:

```
32243 sccs info -b
```

32244 5. To delta everything being edited by the current user:

```
32245 sccs delta $(sccs tell -U)
```

32246 6. In a makefile, to get source files from an SCCS file if it does not already exist:

```
32247 SRCS = <list of source files>
```

```
32248 $(SRCS):
```

```
32249 sccs get $(REL) $@
```

### 32250 RATIONALE

32251 SCCS and its associated utilities are part of the XSI Development Utilities option within the XSI  
 32252 extension.

32253 SCCS is an abbreviation for Source Code Control System. It is a maintenance and enhancement  
 32254 tracking tool. When a file is put under SCCS, the source code control system maintains the file  
 32255 and, when changes are made, identifies and stores them in the file with the original source code  
 32256 and/or documentation. As other changes are made, they too are identified and retained in the  
 32257 file.

32258 Retrieval of the original and any set of changes is possible. Any version of the file as it develops  
 32259 can be reconstructed for inspection or additional modification. History data can be stored with  
 32260 each version, documenting why the changes were made, who made them, and when they were  
 32261 made.

32262 **FUTURE DIRECTIONS**

32263 None.

32264 **SEE ALSO**32265 *admin, delta, get, make, prs, rmdel, sact, unget, val, what*32266 **CHANGE HISTORY**

32267 First released in Issue 4.

32268 **Issue 6**

32269 In the ENVIRONMENT VARIABLES section, the *PROJECTDIR* description is updated from  
32270 “otherwise, the home directory of a user of that name is examined” to “otherwise, the value of  
32271 *PROJECTDIR* is treated as a user name and that user’s initial working directory is examined”.

32272 The normative text is reworded to avoid use of the term “must” for application requirements.

32273 The normative text is reworded to emphasize the term “shall” for implementation requirements.

32274 **NAME**32275        *sed* — stream editor32276 **SYNOPSIS**32277        *sed* [-n] *script*[*file...*]32278        *sed* [-n][-e *script*][...[-f *script\_file*][...*file...*]32279 **DESCRIPTION**

32280        The *sed* utility is a stream editor that shall read one or more text files, make editing changes  
 32281        according to a script of editing commands, and write the results to standard output. The script  
 32282        shall be obtained from either the *script* operand string or a combination of the option-arguments  
 32283        from the *-e script* and *-f script\_file* options.

32284 **OPTIONS**

32285        The *sed* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section  
 32286        12.2, Utility Syntax Guidelines, except that the order of presentation of the *-e* and *-f* options is  
 32287        significant.

32288        The following options shall be supported:

32289        *-e script*     Add the editing commands specified by the *script* option-argument to the end of  
 32290        the script of editing commands. The *script* option-argument shall have the same  
 32291        properties as the *script* operand, described in the OPERANDS section.

32292        *-f script\_file* Add the editing commands in the file *script\_file* to the end of the script.

32293        *-n*            Suppress the default output (in which each line, after it is examined for editing, is  
 32294        written to standard output). Only lines explicitly selected for output are written.

32295        Multiple *-e* and *-f* options may be specified. All commands shall be added to the script in the  
 32296        order specified, regardless of their origin.

32297 **OPERANDS**

32298        The following operands shall be supported:

32299        *file*         A pathname of a file whose contents are read and edited. If multiple *file* operands  
 32300        are specified, the named files shall be read in the order specified and the  
 32301        concatenation shall be edited. If no *file* operands are specified, the standard input  
 32302        shall be used.

32303        *script*        A string to be used as the script of editing commands. The application shall not  
 32304        present a *script* that violates the restrictions of a text file except that the final  
 32305        character need not be a <newline>.

32306 **STDIN**

32307        The standard input shall be used only if no *file* operands are specified. See the INPUT FILES  
 32308        section.

32309 **INPUT FILES**

32310        The input files shall be text files. The *script\_files* named by the *-f* option shall consist of editing  
 32311        commands.

32312 **ENVIRONMENT VARIABLES**

32313        The following environment variables shall affect the execution of *sed*:

32314        *LANG*         Provide a default value for the internationalization variables that are unset or null.  
 32315        (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
 32316        Internationalization Variables for the precedence of internationalization variables  
 32317        used to determine the values of locale categories.)

- 32318 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
32319 internationalization variables.
- 32320 *LC\_COLLATE*  
32321 Determine the locale for the behavior of ranges, equivalence classes, and multi-  
32322 character collating elements within regular expressions.
- 32323 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
32324 characters (for example, single-byte as opposed to multi-byte characters in  
32325 arguments and input files), and the behavior of character classes within regular  
32326 expressions.
- 32327 *LC\_MESSAGES*  
32328 Determine the locale that should be used to affect the format and contents of  
32329 diagnostic messages written to standard error.
- 32330 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 32331 **ASYNCHRONOUS EVENTS**  
32332 Default.
- 32333 **STDOUT**  
32334 The input files shall be written to standard output, with the editing commands specified in the  
32335 script applied. If the *-n* option is specified, only those input lines selected by the script shall be  
32336 written to standard output.
- 32337 **STDERR**  
32338 The standard error shall be used only for diagnostic messages. |
- 32339 **OUTPUT FILES**  
32340 The output files shall be text files whose formats are dependent on the editing commands given.
- 32341 **EXTENDED DESCRIPTION**  
32342 The *script* shall consist of editing commands of the following form:  
32343 *[address[ , address]]function*  
32344 where *function* represents a single-character command verb from the list in **Editing Commands**  
32345 **in sed** (on page 3041), followed by any applicable arguments.  
32346 The command can be preceded by <blank>s and/or semicolons. The function can be preceded  
32347 by <blank>s. These optional characters shall have no effect. |
- 32348 In default operation, *sed* cyclically shall append a line of input, less its terminating <newline>, |  
32349 into the pattern space. Normally the pattern space will be empty, unless a **D** command |  
32350 terminated the last cycle. The *sed* utility shall then apply in sequence all commands whose |  
32351 addresses select that pattern space, and at the end of the script copy the pattern space to |  
32352 standard output (except when *-n* is specified) and delete the pattern space. Whenever the |  
32353 pattern space is written to standard output or a named file, *sed* shall immediately follow it with a |  
32354 <newline>.
- 32355 Some of the editing commands use a hold space to save all or part of the pattern space for  
32356 subsequent retrieval. The pattern and hold spaces shall each be able to hold at least 8 192 bytes.

32357 **Addresses in sed**

32358 An address is either a decimal number that counts input lines cumulatively across files, a '\$' character that addresses the last line of input, or a context address (which consists of a BRE, as described in **Regular Expressions in sed**, preceded and followed by a delimiter, usually a slash).

32361 An editing command with no addresses shall select every pattern space.

32362 An editing command with one address shall select each pattern space that matches the address.

32363 An editing command with two addresses shall select the inclusive range from the first pattern space that matches the first address through the next pattern space that matches the second. (If the second address is a number less than or equal to the line number first selected, only one line shall be selected.) Starting at the first line following the selected range, *sed* shall look again for the first address. Thereafter, the process shall be repeated. Omitting either or both of the address components in the following form produces undefined results:

32369 [*address* [ , *address* ] ]

32370 **Regular Expressions in sed**

32371 The *sed* utility shall support the BREs described in the Base Definitions volume of IEEE Std 1003.1-200x, Section 9.3, Basic Regular Expressions, with the following additions:

- 32373 • In a context address, the construction "\cBREc", where *c* is any character other than  
32374 backslash or <newline>, shall be identical to "/BRE/". If the character designated by *c*  
32375 appears following a backslash, then it shall be considered to be that literal character, which  
32376 shall not terminate the BRE. For example, in the context address "\xabc\xdefx", the  
32377 second *x* stands for itself, so that the BRE is "abcxdef".
- 32378 • The escape sequence '\n' shall match a <newline> embedded in the pattern space. A literal  
32379 <newline> shall not be used in the BRE of a context address or in the substitute function.
- 32380 • If an RE is empty (that is, no pattern is specified) *sed* shall behave as if the last RE used in the  
32381 last command applied (either as an address or as part of a substitute command) was  
32382 specified.

32383 **Editing Commands in sed**

32384 In the following list of editing commands, the maximum number of permissible addresses for  
32385 each function is indicated by [*0addr*], [*1addr*], or [*2addr*], representing zero, one, or two  
32386 addresses.

32387 The argument *text* shall consist of one or more lines. Each embedded <newline> in the text shall  
32388 be preceded by a backslash. Other backslashes in text shall be removed, and the following  
32389 character shall be treated literally.

32390 The **r** and **w** command verbs, and the **w** flag to the **s** command, take an optional *rfile* (or *wfile*)  
32391 parameter, separated from the command verb letter or flag by one or more <blank>;  
32392 implementations may allow zero separation as an extension.

32393 The argument *rfile* or the argument *wfile* shall terminate the editing command. Each *wfile* shall be  
32394 created before processing begins. Implementations shall support at least ten *wfile* arguments in  
32395 the script; the actual number (greater than or equal to 10) that is supported by the  
32396 implementation is unspecified. The use of the *wfile* parameter shall cause that file to be initially  
32397 created, if it does not exist, or shall replace the contents of an existing file.

32398 The **b**, **r**, **s**, **t**, **w**, **y**, and **:** command verbs shall accept additional arguments. The following  
32399 synopses indicate which arguments shall be separated from the command verbs by a single

32400 <space>.

32401 The **a** and **r** commands schedule text for later output. The text specified for the **a** command, and  
 32402 the contents of the file specified for the **r** command, shall be written to standard output just  
 32403 before the next attempt to fetch a line of input when executing the **N** or **n** commands, or when  
 32404 reaching the end of the script. If written when reaching the end of the script, and the **-n** option  
 32405 was not specified, the text shall be written after copying the pattern space to standard output.  
 32406 The contents of the file specified for the **r** command shall be as of the time the output is written,  
 32407 not the time the **r** command is applied. The text shall be output in the order in which the **a** and **r**  
 32408 commands were applied to the input.

32409 Command verbs other than **{**, **a**, **b**, **c**, **i**, **r**, **t**, **w**, **:**, and **#** can be followed by a semicolon, optional  
 32410 <blank>s, and another command verb. However, when the **s** command verb is used with the **w**  
 32411 flag, following it with another command in this manner produces undefined results.

32412 A function can be preceded by one or more **'!'** characters, in which case the function shall be  
 32413 applied if the addresses do not select the pattern space. Zero or more <blank>s shall be accepted  
 32414 before the first **'!'** character. It is unspecified whether <blank>s can follow a **'!'** character, and  
 32415 conforming applications shall not follow a **'!'** character with <blank>s.

32416 **[2addr]{function**  
 32417 **function**  
 32418 ...  
 32419 **}** Execute a list of *sed* functions only when the pattern space is selected. The list of  
 32420 *sed* functions shall be surrounded by braces and separated by <newline>s, and |  
 32421 conform to the following rules. The braces can be preceded or followed by |  
 32422 <blank>s. The functions can be preceded by <blank>s, but shall not be followed  
 32423 by <blank>s. The <right-brace> shall be preceded by a <newline> and can be  
 32424 preceded or followed by <blank>s.

32425 **[1addr]a\**  
 32426 **text** Write text to standard output as described previously.

32427 **[2addr]b [label]**  
 32428 Branch to the **:** function bearing the *label*. If *label* is not specified, branch to the end  
 32429 of the script. The implementation shall support *labels* recognized as unique up to  
 32430 at least 8 characters; the actual length (greater than or equal to 8) that shall be  
 32431 supported by the implementation is unspecified. It is unspecified whether  
 32432 exceeding a label length causes an error or a silent truncation.

32433 **[2addr]c\**  
 32434 **text** Delete the pattern space. With a 0 or 1 address or at the end of a 2-address range,  
 32435 place *text* on the output and start the next cycle.

32436 **[2addr]d** Delete the pattern space and start the next cycle.

32437 **[2addr]D** Delete the initial segment of the pattern space through the first <newline> and  
 32438 start the next cycle.

32439 **[2addr]g** Replace the contents of the pattern space by the contents of the hold space.

32440 **[2addr]G** Append to the pattern space a <newline> followed by the contents of the hold  
 32441 space.

32442 **[2addr]h** Replace the contents of the hold space with the contents of the pattern space.

32443 **[2addr]H** Append to the hold space a <newline> followed by the contents of the pattern  
 32444 space.

|       |                                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|-------|---------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 32445 | <b>[1addr]i\</b>                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 32446 | <i>text</i>                           | Write <i>text</i> to standard output.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 32447 | <b>[2addr]l</b>                       | (The letter ell.) Write the pattern space to standard output in a visually unambiguous form. The characters listed in the Base Definitions volume of IEEE Std 1003.1-200x, Table 5-1, Escape Sequences and Associated Actions ('\\', '\a', '\b', '\f', '\r', '\t', '\v') shall be written as the corresponding escape sequence; the '\n' in that table is not applicable. Non-printable characters not in that table shall be written as one three-digit octal number (with a preceding backslash) for each byte in the character (most significant byte first). If the size of a byte on the system is greater than 9 bits, the format used for non-printable characters is implementation-defined. |
| 32456 |                                       | Long lines shall be folded, with the point of folding indicated by writing a backslash followed by a <newline>; the length at which folding occurs is unspecified, but should be appropriate for the output device. The end of each line shall be marked with a '\$'.                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 32457 |                                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 32458 |                                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 32459 |                                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 32460 | <b>[2addr]n</b>                       | Write the pattern space to standard output if the default output has not been suppressed, and replace the pattern space with the next line of input, less its terminating <newline>.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 32461 |                                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 32462 |                                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 32463 |                                       | If no next line of input is available, the <b>n</b> command verb shall branch to the end of the script and quit without starting a new cycle.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 32464 |                                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 32465 | <b>[2addr]N</b>                       | Append the next line of input, less its terminating <newline>, to the pattern space, using an embedded <newline> to separate the appended material from the original material. Note that the current line number changes.                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 32466 |                                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 32467 |                                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 32468 |                                       | If no next line of input is available, the <b>N</b> command verb shall branch to the end of the script and quit without starting a new cycle or copying the pattern space to standard output.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 32469 |                                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 32470 |                                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 32471 | <b>[2addr]p</b>                       | Write the pattern space to standard output.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 32472 | <b>[2addr]P</b>                       | Write the pattern space, up to the first <newline>, to standard output.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 32473 | <b>[1addr]q</b>                       | Branch to the end of the script and quit without starting a new cycle.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 32474 | <b>[1addr]r rfile</b>                 | Copy the contents of <i>rfile</i> to standard output as described previously. If <i>rfile</i> does not exist or cannot be read, it shall be treated as if it were an empty file, causing no error condition.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 32475 |                                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 32476 |                                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 32477 | <b>[2addr]s/BRE/replacement/flags</b> |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 32478 |                                       | Substitute the replacement string for instances of the BRE in the pattern space. Any character other than backslash or <newline> can be used instead of a slash to delimit the BRE and the replacement. Within the BRE and the replacement, the BRE delimiter itself can be used as a literal character if it is preceded by a backslash.                                                                                                                                                                                                                                                                                                                                                            |
| 32479 |                                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 32480 |                                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 32481 |                                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 32482 |                                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 32483 |                                       | The replacement string shall be scanned from beginning to end. An ampersand ('&') appearing in the replacement shall be replaced by the string matching the BRE. The special meaning of '&' in this context can be suppressed by preceding it by a backslash. The characters "\n", where <i>n</i> is a digit, shall be replaced by the text matched by the corresponding backreference expression. The special meaning of "\n" where <i>n</i> is a digit in this context, can be suppressed by preceding it by a backslash. For each other backslash ('\') encountered, the following character shall lose its special meaning (if any). The meaning of a '\ ' immediately followed                  |
| 32484 |                                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 32485 |                                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 32486 |                                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 32487 |                                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 32488 |                                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 32489 |                                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 32490 |                                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |

|       |                                  |                                                                                                                  |
|-------|----------------------------------|------------------------------------------------------------------------------------------------------------------|
| 32491 |                                  | by any character other than '&', '\', a digit, or the delimiter character used for                               |
| 32492 |                                  | this command, is unspecified.                                                                                    |
| 32493 |                                  | A line can be split by substituting a <newline> into it. The application shall escape                            |
| 32494 |                                  | the <newline> in the replacement by preceding it by a backslash. A substitution                                  |
| 32495 |                                  | shall be considered to have been performed even if the replacement string is                                     |
| 32496 |                                  | identical to the string that it replaces. Any backslash used to alter the default                                |
| 32497 |                                  | meaning of a subsequent character shall be discarded from the BRE or the                                         |
| 32498 |                                  | replacement before evaluating the BRE or using the replacement.                                                  |
| 32499 |                                  | The value of <i>flags</i> shall be zero or more of:                                                              |
| 32500 | <i>n</i>                         | Substitute for the <i>n</i> th occurrence only of the BRE found within the                                       |
| 32501 |                                  | pattern space.                                                                                                   |
| 32502 | <i>g</i>                         | Globally substitute for all non-overlapping instances of the BRE                                                 |
| 32503 |                                  | rather than just the first one. If both <i>g</i> and <i>n</i> are specified, the results                         |
| 32504 |                                  | are unspecified.                                                                                                 |
| 32505 | <i>p</i>                         | Write the pattern space to standard output if a replacement was                                                  |
| 32506 |                                  | made.                                                                                                            |
| 32507 | <i>w wfile</i>                   | Write. Append the pattern space to <i>wfile</i> if a replacement was made.                                       |
| 32508 |                                  | A conforming application shall precede the <i>wfile</i> argument with one                                        |
| 32509 |                                  | or more <blank>s. If the <i>w</i> flag is not the last flag value given in a                                     |
| 32510 |                                  | concatenation of multiple flag values, the results are undefined.                                                |
| 32511 | <b>[2addr]t [label]</b>          |                                                                                                                  |
| 32512 |                                  | Test. Branch to the : command verb bearing the <i>label</i> if any substitutions have been                       |
| 32513 |                                  | made since the most recent reading of an input line or execution of a <i>t</i> . If <i>label</i> is              |
| 32514 |                                  | not specified, branch to the end of the script.                                                                  |
| 32515 | <b>[2addr]w wfile</b>            |                                                                                                                  |
| 32516 |                                  | Append (write) the pattern space to <i>wfile</i> .                                                               |
| 32517 | <b>[2addr]x</b>                  | Exchange the contents of the pattern and hold spaces.                                                            |
| 32518 | <b>[2addr]y/string1/string2/</b> |                                                                                                                  |
| 32519 |                                  | Replace all occurrences of characters in <i>string1</i> with the corresponding characters                        |
| 32520 |                                  | in <i>string2</i> . If a backslash followed by an 'n' appear in <i>string1</i> or <i>string2</i> , the two       |
| 32521 |                                  | characters shall be handled as a single <newline>. If the number of characters in                                |
| 32522 |                                  | <i>string1</i> and <i>string2</i> are not equal, or if any of the characters in <i>string1</i> appear more       |
| 32523 |                                  | than once, the results are undefined. Any character other than backslash or                                      |
| 32524 |                                  | <newline> can be used instead of slash to delimit the strings. If the delimiter is not                           |
| 32525 |                                  | <i>n</i> , within <i>string1</i> and <i>string2</i> , the delimiter itself can be used as a literal character if |
| 32526 |                                  | it is preceded by a backslash. If a backslash character is immediately followed by a                             |
| 32527 |                                  | backslash character in <i>string1</i> or <i>string2</i> , the two backslash characters shall be                  |
| 32528 |                                  | counted as a single literal backslash character. The meaning of a backslash                                      |
| 32529 |                                  | followed by any character that is not 'n', a backslash, or the delimiter character is                            |
| 32530 |                                  | undefined.                                                                                                       |
| 32531 | <b>[0addr]:label</b>             | Do nothing. This command bears a <i>label</i> to which the <i>b</i> and <i>t</i> commands branch.                |
| 32532 | <b>[1addr]=</b>                  | Write the following to standard output:                                                                          |
| 32533 |                                  | "%d\n", <current line number>                                                                                    |
| 32534 | <b>[0addr]</b>                   | Ignore this empty command.                                                                                       |

32535            [*0addr*]#     Ignore the '#' and the remainder of the line (treat them as a comment), with the  
 32536                            single exception that if the first two characters in the script are "#n", the default  
 32537                            output shall be suppressed; this shall be the equivalent of specifying **-n** on the  
 32538                            command line.

### 32539 EXIT STATUS

32540            The following exit values shall be returned:

32541            0    Successful completion.

32542            >0  An error occurred.

### 32543 CONSEQUENCES OF ERRORS

32544            Default.

### 32545 APPLICATION USAGE

32546            Regular expressions match entire strings, not just individual lines, but a <newline> is matched  
 32547            by '\n' in a *sed* RE; a <newline> is not allowed by the general definition of regular expression in  
 32548            IEEE Std 1003.1-200x. Also note that '\n' cannot be used to match a <newline> at the end of an  
 32549            arbitrary input line; <newline>s appear in the pattern space as a result of the **N** editing  
 32550            command.

### 32551 EXAMPLES

32552            This *sed* script simulates the BSD *cat -s* command, squeezing excess blank lines from standard  
 32553            input.

```
32554            sed -n '
32555            # Write non-empty lines.
32556            ./ {
32557                p
32558                d
32559            }
32560            # Write a single empty line, then look for more empty lines.
32561            /^$/ p
32562            # Get next line, discard the held <newline> (empty line),
32563            # and look for more empty lines.
32564            :Empty
32565            /^$/ {
32566                N
32567                s/./ /
32568                b Empty
32569            }
32570            # Write the non-empty line before going back to search
32571            # for the first in a set of empty lines.
32572            p
32573            '
```

### 32574 RATIONALE

32575            This volume of IEEE Std 1003.1-200x requires implementations to support at least ten distinct  
 32576            *wfiles*, matching historical practice on many implementations. Implementations are encouraged  
 32577            to support more, but conforming applications should not exceed this limit. |

32578            The exit status codes specified here are different from those in System V. System V returns 2 for  
 32579            garbled *sed* commands, but returns zero with its usage message or if the input file could not be  
 32580            opened. The standard developers considered this to be a bug.

32581 The manner in which the **I** command writes non-printable characters was changed to avoid the  
 32582 historical backspace-overstrike method, and other requirements to achieve unambiguous output  
 32583 were added. See the RATIONALE for *ed* (on page 2537) for details of the format chosen, which is  
 32584 the same as that chosen for *sed*.

32585 This volume of IEEE Std 1003.1-200x requires implementations to provide pattern and hold  
 32586 spaces of at least 8192 bytes, larger than the 4000 bytes spaces used by some historical  
 32587 implementations, but less than the 20480 bytes limit used in an early proposal. Implementations  
 32588 are encouraged to allocate dynamically larger pattern and hold spaces as needed.

32589 The requirements for acceptance of <blank>s and <space>s in command lines has been made  
 32590 more explicit than in early proposals to describe clearly the historical practice and to remove  
 32591 confusion about the phrase “protect initial blanks [*sic*] and tabs from the stripping that is done  
 32592 on every script line” that appears in much of the historical documentation of the *sed* utility  
 32593 description of text. (Not all implementations are known to have stripped <blank>s from text  
 32594 lines, although they all have allowed leading <blank>s preceding the address on a command  
 32595 line.)

32596 The treatment of ‘#’ comments differs from the SVID which only allows a comment as the first  
 32597 line of the script, but matches BSD-derived implementations. The comment character is treated  
 32598 as a command, and it has the same properties in terms of being accepted with leading <blank>s;  
 32599 the BSD implementation has historically supported this.

32600 Early proposals required that a *script\_file* have at least one non-comment line. Some historical  
 32601 implementations have behaved in unexpected ways if this were not the case. The standard  
 32602 developers considered that this was incorrect behavior and that application developers should  
 32603 not have to avoid this feature. A correct implementation of this volume of IEEE Std 1003.1-200x  
 32604 shall permit *script\_files* that consist only of comment lines.

32605 Early proposals indicated that if **-e** and **-f** options were intermixed, all **-e** options were  
 32606 processed before any **-f** options. This has been changed to process them in the order presented  
 32607 because it matches historical practice and is more intuitive.

32608 The treatment of the **p** flag to the **s** command differs between System V and BSD-based systems  
 32609 when the default output is suppressed. In the two examples:

```
32610 echo a | sed 's/a/A/p'
32611 echo a | sed -n 's/a/A/p'
```

32612 This volume of IEEE Std 1003.1-200x, BSD, System V documentation, and the SVID indicate that  
 32613 the first example should write two lines with **A**, whereas the second should write one. Some  
 32614 System V systems write the **A** only once in both examples because the **p** flag is ignored if the **-n**  
 32615 option is not specified.

32616 This is a case of a diametrical difference between systems that could not be reconciled through  
 32617 the compromise of declaring the behavior to be unspecified. The SVID/BSD/System V  
 32618 documentation behavior was adopted for this volume of IEEE Std 1003.1-200x because:

- 32619 • No known documentation for any historic system describes the interaction between the **p**  
 32620 flag and the **-n** option.
- 32621 • The selected behavior is more correct as there is no technical justification for any interaction  
 32622 between the **p** flag and the **-n** option. A relationship between **-n** and the **p** flag might imply  
 32623 that they are only used together, but this ignores valid scripts that interrupt the cyclical  
 32624 nature of the processing through the use of the **D**, **d**, **q**, or branching commands. Such scripts  
 32625 rely on the **p** suffix to write the pattern space because they do not make use of the default  
 32626 output at the “bottom” of the script.

- 32627       • Because the `-n` option makes the `p` flag unnecessary, any interaction would only be useful if  
 32628 *sed* scripts were written to run both with and without the `-n` option. This is believed to be  
 32629 unlikely. It is even more unlikely that programmers have coded the `p` flag expecting it to be  
 32630 unnecessary. Because the interaction was not documented, the likelihood of a programmer  
 32631 discovering the interaction and depending on it is further decreased.
- 32632       • Finally, scripts that break under the specified behavior produce too much output instead of  
 32633 too little, which is easier to diagnose and correct.
- 32634       The form of the substitute command that uses the `n` suffix was limited to the first 512 matches in  
 32635 an early proposal. This limit has been removed because there is no reason an editor processing  
 32636 lines of `{LINE_MAX}` length should have this restriction. The command `s/a/A/2047` should be  
 32637 able to substitute the 2047th occurrence of `a` on a line.
- 32638       The `b`, `t`, and `:` commands are documented to ignore leading white space, but no mention is  
 32639 made of trailing white space. Historical implementations of *sed* assigned different locations to  
 32640 the labels `'x'` and `"x "`. This is not useful, and leads to subtle programming errors, but it is  
 32641 historical practice, and changing it could theoretically break working scripts. Implementors are  
 32642 encouraged to provide warning messages about labels that are never used or jumps to labels  
 32643 that do not exist.
- 32644       Historically, the *sed* `!` and `}` editing commands did not permit multiple commands on a single  
 32645 line using a semicolon as a command delimiter. Implementations are permitted, but not  
 32646 required, to support this extension.
- 32647 **FUTURE DIRECTIONS**
- 32648       None.
- 32649 **SEE ALSO**
- 32650       *awk*, *ed*, *grep*
- 32651 **CHANGE HISTORY**
- 32652       First released in Issue 2.
- 32653 **Issue 5**
- 32654       FUTURE DIRECTIONS section added.
- 32655 **Issue 6**
- 32656       The following new requirements on POSIX implementations derive from alignment with the  
 32657 Single UNIX Specification:
- 32658       • Implementations are required to support at least ten *wfile* arguments in an editing command.
- 32659       The EXTENDED DESCRIPTION is changed to align with the IEEE P1003.2b draft standard.
- 32660       IEEE PASC Interpretation 1003.2 #190 is applied. |
- 32661       IEEE PASC Interpretation 1003.2 #203 is applied, clarifying the meaning of the backslash escape |  
 32662 sequences in a replacement string for a BRE. |

## 32663 NAME

32664 sh — shell, the standard command language interpreter

## 32665 SYNOPSIS

32666 sh [-abCefhimnuvx][-o *option*][+abCefhimnuvx][+o *option*]  
32667 [*command\_file* [*argument...*]]

32668 sh -c[-abCefhimnuvx][-o *option*][+abCefhimnuvx][+o *option*]*command\_string*  
32669 [*command\_name* [*argument...*]]

32670 sh -s[-abCefhimnuvx][-o *option*][+abCefhimnuvx][+o *option*][*argument*]

## 32671 DESCRIPTION

32672 The *sh* utility is a command language interpreter that shall execute commands read from a  
32673 command line string, the standard input, or a specified file. The application shall ensure that the  
32674 commands to be executed are expressed in the language described in Chapter 2 (on page 2231).

32675 Pathname expansion shall not fail due to the size of a file.

32676 Shell input and output redirections have an implementation-defined offset maximum that is  
32677 established in the open file description.

## 32678 OPTIONS

32679 The *sh* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2,  
32680 Utility Syntax Guidelines, with an extension for support of a leading plus sign ('+') as noted  
32681 below.

32682 The **-a**, **-b**, **-C**, **-e**, **-f**, **-m**, **-n**, **-o *option***, **-u**, **-v**, and **-x** options are described as part of the *set*  
32683 utility in Section 2.14 (on page 2266). The option letters derived from the *set* special built-in shall  
32684 also be accepted with a leading plus sign ('+') instead of a leading hyphen (meaning the reverse  
32685 case of the option as described in this volume of IEEE Std 1003.1-200x).

32686 The following additional options shall be supported:

32687 **-c** Read commands from the *command\_string* operand. Set the value of special  
32688 parameter 0 (see Section 2.5.2 (on page 2235)) from the value of the *command\_name*  
32689 operand and the positional parameters (\$1, \$2, and so on) in sequence from the  
32690 remaining *argument* operands. No commands shall be read from the standard  
32691 input.

32692 **-i** Specify that the shell is *interactive*; see below. An implementation may treat  
32693 specifying the **-i** option as an error if the real user ID of the calling process does  
32694 not equal the effective user ID or if the real group ID does not equal the effective  
32695 group ID.

32696 **-s** Read commands from the standard input.

32697 If there are no operands and the **-c** option is not specified, the **-s** option shall be assumed.

32698 If the **-i** option is present, or if there are no operands and the shell's standard input and standard  
32699 error are attached to a terminal, the shell is considered to be *interactive*.

## 32700 OPERANDS

32701 The following operands shall be supported:

32702 **-** A single hyphen shall be treated as the first operand and then ignored. If both **'-'** |  
32703 and **"--"** are given as arguments, or if other operands precede the single hyphen,  
32704 the results are undefined.

32705 *argument* The positional parameters (\$1, \$2, and so on) shall be set to *arguments*, if any.

32706 *command\_file* The pathname of a file containing commands. If the pathname contains one or  
 32707 more slash characters, the implementation attempts to read that file; the file need  
 32708 not be executable. If the pathname does not contain a slash character:

- The implementation shall attempt to read that file from the current working directory; the file need not be executable.

- If the file is not in the current working directory, the implementation may perform a search for an executable file using the value of *PATH*, as described in Section 2.9.1.1 (on page 2249).

32714 Special parameter 0 (see Section 2.5.2 (on page 2235)) shall be set to the value of  
 32715 *command\_file*. If *sh* is called using a synopsis form that omits *command\_file*, special  
 32716 parameter 0 shall be set to the value of the first argument passed to *sh* from its  
 32717 parent (for example, *argv[0]* for a C program), which is normally a pathname used  
 32718 to execute the *sh* utility.

32719 *command\_name*

32720 A string assigned to special parameter 0 when executing the commands in  
 32721 *command\_string*. If *command\_name* is not specified, special parameter 0 shall be set  
 32722 to the value of the first argument passed to *sh* from its parent (for example, *argv[0]*  
 32723 for a C program), which is normally a pathname used to execute the *sh* utility.

32724 *command\_string*

32725 A string that shall be interpreted by the shell as one or more commands, as if the  
 32726 string were the argument to the *system()* function defined in the System Interfaces  
 32727 volume of IEEE Std 1003.1-200x. If the *command\_string* operand is an empty string,  
 32728 *sh* shall exit with a zero exit status.

## 32729 STDIN

32730 The standard input shall be used only if one of the following is true:

- The *-s* option is specified.
- The *-c* option is not specified and no operands are specified.
- The script executes one or more commands that require input from standard input (such as a *read* command that does not redirect its input).

32735 See the INPUT FILES section.

32736 When the shell is using standard input and it invokes a command that also uses standard input,  
 32737 the shell shall ensure that the standard input file pointer points directly after the command it has  
 32738 read when the command begins execution. It shall not read ahead in such a manner that any  
 32739 characters intended to be read by the invoked command are consumed by the shell (whether  
 32740 interpreted by the shell or not) or that characters that are not read by the invoked command are  
 32741 not seen by the shell. When the command expecting to read standard input is started  
 32742 asynchronously by an interactive shell, it is unspecified whether characters are read by the  
 32743 command or interpreted by the shell.

32744 If the standard input to *sh* is a FIFO or terminal device and is set to non-blocking reads, then *sh*  
 32745 shall enable blocking reads on standard input. This shall remain in effect when the command  
 32746 completes.

## 32747 INPUT FILES

32748 The input file shall be a text file, except that line lengths shall be unlimited. If the input file is  
 32749 empty or consists solely of blank lines or comments, or both, *sh* shall exit with a zero exit status.

## 32750 ENVIRONMENT VARIABLES

32751 The following environment variables shall affect the execution of *sh*:

32752 *ENV* This variable, when and only when an interactive shell is invoked, shall be  
 32753 subjected to parameter expansion (see Section 2.6.2 (on page 2239)) by the shell,  
 32754 and the resulting value shall be used as a pathname of a file containing shell  
 32755 commands to execute in the current environment. The file need not be executable.  
 32756 If the expanded value of *ENV* is not an absolute pathname, the results are  
 32757 unspecified. *ENV* shall be ignored if the real and effective user IDs or real and  
 32758 effective group IDs of the process are different.

32759 *FCEDIT* This variable, when expanded by the shell, shall determine the default value for |  
 32760 the *-e editor* option's *editor* option-argument. If *FCEDIT* is null or unset, *ed* shall be |  
 32761 used as the editor. This volume of IEEE Std 1003.1-200x specifies the effects of this  
 32762 variable only for systems supporting the User Portability Utilities option.

32763 *HISTFILE* Determine a pathname naming a command history file. If the *HISTFILE* variable is  
 32764 not set, the shell may attempt to access or create a file *.sh\_history* in the directory  
 32765 referred to by the *HOME* environment variable. If the shell cannot obtain both read  
 32766 and write access to, or create, the history file, it shall use an unspecified  
 32767 mechanism that allows the history to operate properly. (References to history  
 32768 "file" in this section shall be understood to mean this unspecified mechanism in  
 32769 such cases.) An implementation may choose to access this variable only when  
 32770 initializing the history file; this initialization shall occur when *fc* or *sh* first attempt  
 32771 to retrieve entries from, or add entries to, the file, as the result of commands issued  
 32772 by the user, the file named by the *ENV* variable, or implementation-defined system  
 32773 start-up files. Implementations may choose to disable the history list mechanism  
 32774 for users with appropriate privileges who do not set *HISTFILE*; the specific  
 32775 circumstances under which this occurs are implementation-defined. If more than  
 32776 one instance of the shell is using the same history file, it is unspecified how  
 32777 updates to the history file from those shells interact. As entries are deleted from  
 32778 the history file, they shall be deleted oldest first. It is unspecified when history file  
 32779 entries are physically removed from the history file. This volume of  
 32780 IEEE Std 1003.1-200x specifies the effects of this variable only for systems  
 32781 supporting the User Portability Utilities option.

32782 *HISTSIZE* Determine a decimal number representing the limit to the number of previous  
 32783 commands that are accessible. If this variable is unset, an unspecified default  
 32784 greater than or equal to 128 shall be used. The maximum number of commands in  
 32785 the history list is unspecified, but shall be at least 128. An implementation may  
 32786 choose to access this variable only when initializing the history file, as described  
 32787 under *HISTFILE*. Therefore, it is unspecified whether changes made to *HISTSIZE*  
 32788 after the history file has been initialized are effective.

32789 *HOME* Determine the pathname of the user's home directory. The contents of *HOME* are  
 32790 used in Tilde Expansion as described in Section 2.6.1 (on page 2239). This volume  
 32791 of IEEE Std 1003.1-200x specifies the effects of this variable only for systems  
 32792 supporting the User Portability Utilities option.

32793 *IFS* *Input field separators*: a string treated as a list of characters that shall be used for  
 32794 field splitting and to split lines into words with the *read* command. See Section  
 32795 2.6.5 (on page 2243). If *IFS* is not set, the shell shall behave as if the value of *IFS* |  
 32796 were <space>, <tab>, and <newline>. Implementations may ignore the value of |  
 32797 *IFS* in the environment at the time *sh* is invoked, treating *IFS* as if it were not set.

|           |                    |                                                                                                  |
|-----------|--------------------|--------------------------------------------------------------------------------------------------|
| 32798     | <i>LANG</i>        | Provide a default value for the internationalization variables that are unset or null.           |
| 32799     |                    | (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,                           |
| 32800     |                    | Internationalization Variables for the precedence of internationalization variables              |
| 32801     |                    | used to determine the values of locale categories.)                                              |
| 32802     | <i>LC_ALL</i>      | If set to a non-empty string value, override the values of all the other                         |
| 32803     |                    | internationalization variables.                                                                  |
| 32804     | <i>LC_COLLATE</i>  |                                                                                                  |
| 32805     |                    | Determine the behavior of range expressions, equivalence classes and multi-                      |
| 32806     |                    | character collating elements within pattern matching.                                            |
| 32807     | <i>LC_CTYPE</i>    | Determine the locale for the interpretation of sequences of bytes of text data as                |
| 32808     |                    | characters (for example, single-byte as opposed to multi-byte characters in                      |
| 32809     |                    | arguments and input files), which characters are defined as letters (character class             |
| 32810     |                    | <b>alpha</b> ), and the behavior of character classes within pattern matching.                   |
| 32811     | <i>LC_MESSAGES</i> |                                                                                                  |
| 32812     |                    | Determine the locale that should be used to affect the format and contents of                    |
| 32813     |                    | diagnostic messages written to standard error.                                                   |
| 32814     | <i>MAIL</i>        | Determine a pathname of the user's mailbox file for purposes of incoming mail                    |
| 32815     |                    | notification. If this variable is set, the shell shall inform the user if the file named by      |
| 32816     |                    | the variable is created or if its modification time has changed. Informing the user              |
| 32817     |                    | shall be accomplished by writing a string of unspecified format to standard error                |
| 32818     |                    | prior to the writing of the next primary prompt string. Such check shall be                      |
| 32819     |                    | performed only after the completion of the interval defined by the <i>MAILCHECK</i>              |
| 32820     |                    | variable after the last such check. The user shall be informed only if <i>MAIL</i> is set        |
| 32821     |                    | and <i>MAILPATH</i> is not set. This volume of IEEE Std 1003.1-200x specifies the effects        |
| 32822     |                    | of this variable only for systems supporting the User Portability Utilities option.              |
| 32823     | <i>MAILCHECK</i>   |                                                                                                  |
| 32824     |                    | Establish a decimal integer value that specifies how often (in seconds) the shell                |
| 32825     |                    | shall check for the arrival of mail in the files specified by the <i>MAILPATH</i> or <i>MAIL</i> |
| 32826     |                    | variables. The default value shall be 600 seconds. If set to zero, the shell shall check         |
| 32827     |                    | before issuing each primary prompt. This volume of IEEE Std 1003.1-200x specifies                |
| 32828     |                    | the effects of this variable only for systems supporting the User Portability Utilities          |
| 32829     |                    | option.                                                                                          |
| 32830     | <i>MAILPATH</i>    | Provide a list of pathnames and optional messages separated by colons. If this                   |
| 32831     |                    | variable is set, the shell shall inform the user if any of the files named by the                |
| 32832     |                    | variable are created or if any of their modification times change. (See the preceding            |
| 32833     |                    | entry for <i>MAIL</i> for descriptions of mail arrival and user informing.) Each                 |
| 32834     |                    | pathname can be followed by ' <i>%</i> ' and a string that shall be subjected to parameter       |
| 32835     |                    | expansion and written to standard error when the modification time changes. If a                 |
| 32836     |                    | ' <i>%</i> ' character in the pathname is preceded by a backslash, it shall be treated as a      |
| 32837     |                    | literal ' <i>%</i> ' in the pathname. The default message is unspecified.                        |
| 32838     |                    | The <i>MAILPATH</i> environment variable takes precedence over the <i>MAIL</i> variable.         |
| 32839     |                    | This volume of IEEE Std 1003.1-200x specifies the effects of this variable only for              |
| 32840     |                    | systems supporting the User Portability Utilities option.                                        |
| 32841 XSI | <i>NLSPATH</i>     | Determine the location of message catalogs for the processing of <i>LC_MESSAGES</i> .            |
| 32842     | <i>PATH</i>        | Establish a string formatted as described in the Base Definitions volume of                      |
| 32843     |                    | IEEE Std 1003.1-200x, Chapter 8, Environment Variables, used to effect command                   |
| 32844     |                    | interpretation; see Section 2.9.1.1 (on page 2249).                                              |

32845 *PWD* This variable shall represent an absolute pathname of the current working  
32846 directory. Assignments to this variable may be ignored unless the value is an  
32847 absolute pathname of the current working directory and there are no filename  
32848 components of dot or dot-dot.

#### 32849 **ASYNCHRONOUS EVENTS**

32850 Default.

#### 32851 **STDOUT**

32852 See the **STDERR** section.

#### 32853 **STDERR**

32854 Except as otherwise stated (by the descriptions of any invoked utilities or in interactive mode), |  
32855 standard error shall be used only for diagnostic messages. |

#### 32856 **OUTPUT FILES**

32857 None.

#### 32858 **EXTENDED DESCRIPTION**

32859 See Chapter 2. The following additional capabilities are supported on systems supporting the  
32860 User Portability Utilities option.

#### 32861 **Command History List**

32862 When the *sh* utility is being used interactively, it shall maintain a list of commands previously  
32863 entered from the terminal in the file named by the *HISTFILE* environment variable. The type,  
32864 size, and internal format of this file are unspecified. Multiple *sh* processes can share access to the  
32865 file for a user, if file access permissions allow this; see the description of the *HISTFILE*  
32866 environment variable.

#### 32867 **Command Line Editing**

32868 When *sh* is being used interactively from a terminal, the current command and the command  
32869 history (see *fc* (on page 2637)) can be edited using *vi*-mode command line editing. This mode  
32870 uses commands, described below, similar to a subset of those described in the *vi* utility.  
32871 Implementations may offer other command line editing modes corresponding to other editing  
32872 utilities.

32873 The command *set -o vi* shall enable *vi*-mode editing and place *sh* into *vi* insert mode (see  
32874 **Command Line Editing (vi-mode)** (on page 3053)). This command also shall disable any other  
32875 editing mode that the implementation may provide. The command *set +o vi* disables *vi*-mode  
32876 editing.

32877 Certain block-mode terminals may be unable to support shell command line editing. If a  
32878 terminal is unable to provide either edit mode, it need not be possible to *set -o vi* when using the  
32879 shell on this terminal.

32880 In the following sections, the characters *erase*, *interrupt*, *kill*, and *end-of-file* are those set by the  
32881 *stty* utility.

32882 **Command Line Editing (vi-mode)**

32883 In *vi* editing mode, there shall be a distinguished line, the edit line. All the editing operations  
 32884 which modify a line affect the edit line. The edit line is always the newest line in the command  
 32885 history buffer.

32886 With *vi*-mode enabled, *sh* can be switched between insert mode and command mode.

32887 When in insert mode, an entered character shall be inserted into the command line, except as  
 32888 noted in **vi Line Editing Insert Mode**. Upon entering *sh* and after termination of the previous  
 32889 command, *sh* shall be in insert mode.

32890 Typing an escape character shall switch *sh* into command mode (see **vi Line Editing Command**  
 32891 **Mode** (on page 3054)). In command mode, an entered character shall either invoke a defined  
 32892 operation, is used as part of a multi-character operation, or is treated as an error. A character that  
 32893 is not recognized as part of an editing command shall terminate any specific editing command  
 32894 and shall alert the terminal. Typing the *interrupt* character in command mode shall cause *sh* to  
 32895 terminate command line editing on the current command line, reissue the prompt on the next  
 32896 line of the terminal, and reset the command history (see *fc* (on page 2637)) so that the most  
 32897 recently executed command is the previous command (that is, the command that was being  
 32898 edited when it was interrupted is not reentered into the history).

32899 In the following sections, the phrase “move the cursor to the beginning of the word” shall mean  
 32900 “move the cursor to the first character of the current word” and the phrase “move the cursor to  
 32901 the end of the word” shall mean “move the cursor to the last character of the current word”. The  
 32902 phrase “beginning of the command line” indicates the point between the end of the prompt  
 32903 string issued by the shell (or the beginning of the terminal line, if there is no prompt string) and  
 32904 the first character of the command text.

32905 **vi Line Editing Insert Mode**

32906 While in insert mode, any character typed shall be inserted in the current command line, unless  
 32907 it is from the following set.

32908 <newline> Execute the current command line. If the current command line is not empty, this  
 32909 line shall be entered into the command history (see *fc*).

32910 *erase* Delete the character previous to the current cursor position and move the current  
 32911 cursor position back one character. In insert mode, characters shall be erased from  
 32912 both the screen and the buffer when backspacing.

32913 *interrupt* Terminate command line editing with the same effects as described for  
 32914 interrupting command mode; see **Command Line Editing (vi-mode)**.

32915 *kill* Clear all the characters from the input line.

32916 <control>-V Insert the next character input, even if the character is otherwise a special insert  
 32917 mode character.

32918 <control>-W Delete the characters from the one preceding the cursor to the preceding word  
 32919 boundary. The word boundary in this case is the closer to the cursor of either the  
 32920 beginning of the line or a character that is in neither the **blank** nor **punct** character  
 32921 classification of the current locale.

32922 *end-of-file* Interpreted as the end of input in *sh*. This interpretation shall occur only at the  
 32923 beginning of an input line. If *end-of-file* is entered other than at the beginning of the  
 32924 line, the results are unspecified.

- 32925           <ESC>       Place *sh* into command mode.
- 32926           **vi Line Editing Command Mode**
- 32927           In command mode for the command line editing feature, decimal digits not beginning with 0  
32928           that precede a command letter shall be remembered. Some commands use these decimal digits  
32929           as a count number that affects the operation.
- 32930           The term *motion command* represents one of the commands:
- 32931           <space> 0 b F l W ^ \$ ; E f T w | , B e h t
- 32932           If the current line is not the edit line, any command that modifies the current line shall cause the  
32933           content of the current line to replace the content of the edit line, and the current line shall  
32934           become the edit line. This replacement cannot be undone (see the **u** and **U** commands below).  
32935           The modification requested shall then be performed to the edit line. When the current line is the  
32936           edit line, the modification shall be done directly to the edit line.
- 32937           Any command that is preceded by *count* shall take a count (the numeric value of any preceding  
32938           decimal digits). Unless otherwise noted, this count shall cause the specified operation to repeat  
32939           by the number of times specified by the count. Also unless otherwise noted, a *count* that is out of  
32940           range is considered an error condition and shall alert the terminal, but neither the cursor  
32941           position, nor the command line, shall change.
- 32942           The terms *word* and *bigword* are used as defined in the *vi* description. The term *save buffer*  
32943           corresponds to the term *unnamed buffer* in *vi*.
- 32944           The following commands shall be recognized in command mode:
- 32945           <newline>   Execute the current command line. If the current command line is not empty, this  
32946           line shall be entered into the command history (see *fc*).
- 32947           <control>-L   Redraw the current command line. Position the cursor at the same location on the  
32948           redrawn line.
- 32949           #           Insert the character '#' at the beginning of the current command line and treat the  
32950           resulting edit line as a comment. This line shall be entered into the command  
32951           history; see *fc* (on page 2637).
- 32952           =           Display the possible shell word expansions (see Section 2.6 (on page 2238)) of the  
32953           bigword at the current command line position.
- 32954           **Note:**       This does not modify the content of the current line, and therefore does not  
32955           cause the current line to become the edit line.
- 32956           These expansions shall be displayed on subsequent terminal lines. If the bigword  
32957           contains none of the characters '?', '\*', or '[', an asterisk('\*') shall be  
32958           implicitly assumed at the end. If any directories are matched, these expansions  
32959           shall have a '/' character appended. After the expansion, the line shall be  
32960           redrawn, the cursor is repositioned at the current cursor position, and *sh* shall be  
32961           placed in command mode.
- 32962           \           Perform pathname expansion (see Section 2.6.6 (on page 2244)) on the current  
32963           bigword, up to the largest set of characters that can be matched uniquely. If the  
32964           bigword contains none of the characters '?', '\*', or '[', an asterisk('\*') shall  
32965           be implicitly assumed at the end. This maximal expansion then shall replace the  
32966           original bigword in the command line, and the cursor shall be placed after this  
32967           expansion. If the resulting bigword completely and uniquely matches a directory, a  
32968            '/' character shall be inserted directly after the bigword. If some other file is  
32969           completely matched, a single <space> shall be inserted after the bigword. After

|       |                         |                                                                                                        |
|-------|-------------------------|--------------------------------------------------------------------------------------------------------|
| 32970 |                         | this operation, <i>sh</i> shall be placed in insert mode.                                              |
| 32971 | *                       | Perform pathname expansion on the current bigword and insert all expansions                            |
| 32972 |                         | into the command to replace the current bigword, with each expansion separated                         |
| 32973 |                         | by a single <space>. If at the end of the line, the current cursor position shall be                   |
| 32974 |                         | moved to the first column position following the expansions and <i>sh</i> shall be placed              |
| 32975 |                         | in insert mode. Otherwise, the current cursor position shall be the last column                        |
| 32976 |                         | position of the first character after the expansions and <i>sh</i> shall be placed in insert           |
| 32977 |                         | mode. If the current bigword contains none of the characters '?', '*', or '[',                         |
| 32978 |                         | before the operation, an asterisk shall be implicitly assumed at the end.                              |
| 32979 | @ <i>letter</i>         | Insert the value of the alias named <i>_letter</i> . The symbol <i>letter</i> represents a single      |
| 32980 |                         | alphabetic character from the portable character set; implementations may support                      |
| 32981 |                         | additional characters as an extension. If the alias <i>_letter</i> contains other editing              |
| 32982 |                         | commands, these commands shall be performed as part of the insertion. If no alias                      |
| 32983 |                         | <i>_letter</i> is enabled, this command shall have no effect.                                          |
| 32984 | [ <i>count</i> ]~       | Convert, if the current character is a lowercase letter, to the equivalent uppercase                   |
| 32985 |                         | letter and <i>vice versa</i> , as prescribed by the current locale. The current cursor position        |
| 32986 |                         | then shall be advanced by one character. If the cursor was positioned on the last                      |
| 32987 |                         | character of the line, the case conversion shall occur, but the cursor shall not                       |
| 32988 |                         | advance. If the '~' command is preceded by a <i>count</i> , that number of characters                  |
| 32989 |                         | shall be converted, and the cursor shall be advanced to the character position after                   |
| 32990 |                         | the last character converted. If the <i>count</i> is larger than the number of characters              |
| 32991 |                         | after the cursor, this shall not be considered an error; the cursor shall advance to                   |
| 32992 |                         | the last character on the line.                                                                        |
| 32993 | [ <i>count</i> ].       | Repeat the most recent non-motion command, even if it was executed on an earlier                       |
| 32994 |                         | command line. If the previous command was preceded by a <i>count</i> , and no count is                 |
| 32995 |                         | given on the '.' command, the count from the previous command shall be                                 |
| 32996 |                         | included as part of the repeated command. If the '.' command is preceded by a                          |
| 32997 |                         | <i>count</i> , this shall override any <i>count</i> argument to the previous command. The <i>count</i> |
| 32998 |                         | specified in the '.' command shall become the count for subsequent '.'                                 |
| 32999 |                         | commands issued without a count.                                                                       |
| 33000 | [ <i>number</i> ]v      | Invoke the <i>vi</i> editor to edit the current command line in a temporary file. When the             |
| 33001 |                         | editor exits, the commands in the temporary file shall be executed and placed in                       |
| 33002 |                         | the command history. If a <i>number</i> is included, it specifies the command number in                |
| 33003 |                         | the command history to be edited, rather than the current command line.                                |
| 33004 | [ <i>count</i> ]l (ell) |                                                                                                        |
| 33005 | [ <i>count</i> ]<space> |                                                                                                        |
| 33006 |                         | Move the current cursor position to the next character position. If the cursor was                     |
| 33007 |                         | positioned on the last character of the line, the terminal shall be alerted and the                    |
| 33008 |                         | cursor shall not be advanced. If the <i>count</i> is larger than the number of characters              |
| 33009 |                         | after the cursor, this shall not be considered an error; the cursor shall advance to                   |
| 33010 |                         | the last character on the line.                                                                        |
| 33011 | [ <i>count</i> ]h       | Move the current cursor position to the <i>count</i> th (default 1) previous character                 |
| 33012 |                         | position. If the cursor was positioned on the first character of the line, the terminal                |
| 33013 |                         | shall be alerted and the cursor shall not be moved. If the count is larger than the                    |
| 33014 |                         | number of characters before the cursor, this shall not be considered an error; the                     |
| 33015 |                         | cursor shall move to the first character on the line.                                                  |
| 33016 | [ <i>count</i> ]w       | Move to the start of the next word. If the cursor was positioned on the last                           |
| 33017 |                         | character of the line, the terminal shall be alerted and the cursor shall not be                       |

|       |                  |                                                                                               |
|-------|------------------|-----------------------------------------------------------------------------------------------|
| 33018 |                  | advanced. If the <i>count</i> is larger than the number of words after the cursor, this shall |
| 33019 |                  | not be considered an error; the cursor shall advance to the last character on the             |
| 33020 |                  | line.                                                                                         |
| 33021 | <b>[count]W</b>  | Move to the start of the next bigword. If the cursor was positioned on the last               |
| 33022 |                  | character of the line, the terminal shall be alerted and the cursor shall not be              |
| 33023 |                  | advanced. If the <i>count</i> is larger than the number of bigwords after the cursor, this    |
| 33024 |                  | shall not be considered an error; the cursor shall advance to the last character on           |
| 33025 |                  | the line.                                                                                     |
| 33026 | <b>[count]e</b>  | Move to the end of the current word. If at the end of a word, move to the end of the          |
| 33027 |                  | next word. If the cursor was positioned on the last character of the line, the                |
| 33028 |                  | terminal shall be alerted and the cursor shall not be advanced. If the <i>count</i> is larger |
| 33029 |                  | than the number of words after the cursor, this shall not be considered an error; the         |
| 33030 |                  | cursor shall advance to the last character on the line.                                       |
| 33031 | <b>[count]E</b>  | Move to the end of the current bigword. If at the end of a bigword, move to the               |
| 33032 |                  | end of the next bigword. If the cursor was positioned on the last character of the            |
| 33033 |                  | line, the terminal shall be alerted and the cursor shall not be advanced. If the <i>count</i> |
| 33034 |                  | is larger than the number of bigwords after the cursor, this shall not be considered          |
| 33035 |                  | an error; the cursor shall advance to the last character on the line.                         |
| 33036 | <b>[count]b</b>  | Move to the beginning of the current word. If at the beginning of a word, move to             |
| 33037 |                  | the beginning of the previous word. If the cursor was positioned on the first                 |
| 33038 |                  | character of the line, the terminal shall be alerted and the cursor shall not be              |
| 33039 |                  | moved. If the <i>count</i> is larger than the number of words preceding the cursor, this      |
| 33040 |                  | shall not be considered an error; the cursor shall return to the first character on the       |
| 33041 |                  | line.                                                                                         |
| 33042 | <b>[count]B</b>  | Move to the beginning of the current bigword. If at the beginning of a bigword,               |
| 33043 |                  | move to the beginning of the previous bigword. If the cursor was positioned on the            |
| 33044 |                  | first character of the line, the terminal shall be alerted and the cursor shall not be        |
| 33045 |                  | moved. If the <i>count</i> is larger than the number of bigwords preceding the cursor,        |
| 33046 |                  | this shall not be considered an error; the cursor shall return to the first character on      |
| 33047 |                  | the line.                                                                                     |
| 33048 | <b>^</b>         | Move the current cursor position to the first character on the input line that is not a       |
| 33049 |                  | <blank>.                                                                                      |
| 33050 | <b>\$</b>        | Move to the last character position on the current command line.                              |
| 33051 | <b>0</b>         | (Zero.) Move to the first character position on the current command line.                     |
| 33052 | <b>[count]  </b> | Move to the <i>count</i> th character position on the current command line. If no number      |
| 33053 |                  | is specified, move to the first position. The first character position shall be               |
| 33054 |                  | numbered 1. If the count is larger than the number of characters on the line, this            |
| 33055 |                  | shall not be considered an error; the cursor shall be placed on the last character on         |
| 33056 |                  | the line.                                                                                     |
| 33057 | <b>[count]fc</b> | Move to the first occurrence of the character 'c' that occurs after the current               |
| 33058 |                  | cursor position. If the cursor was positioned on the last character of the line, the          |
| 33059 |                  | terminal shall be alerted and the cursor shall not be advanced. If the character 'c'          |
| 33060 |                  | does not occur in the line after the current cursor position, the terminal shall be           |
| 33061 |                  | alerted and the cursor shall not be moved.                                                    |
| 33062 | <b>[count]Fc</b> | Move to the first occurrence of the character 'c' that occurs before the current              |
| 33063 |                  | cursor position. If the cursor was positioned on the first character of the line, the         |
| 33064 |                  | terminal shall be alerted and the cursor shall not be moved. If the character 'c'             |

|       |                       |                                                                                                          |
|-------|-----------------------|----------------------------------------------------------------------------------------------------------|
| 33065 |                       | does not occur in the line before the current cursor position, the terminal shall be                     |
| 33066 |                       | alerted and the cursor shall not be moved.                                                               |
| 33067 | <b>[count]tc</b>      | Move to the character before the first occurrence of the character 'c' that occurs                       |
| 33068 |                       | after the current cursor position. If the cursor was positioned on the last character                    |
| 33069 |                       | of the line, the terminal shall be alerted and the cursor shall not be advanced. If the                  |
| 33070 |                       | character 'c' does not occur in the line after the current cursor position, the                          |
| 33071 |                       | terminal shall be alerted and the cursor shall not be moved.                                             |
| 33072 | <b>[count]Tc</b>      | Move to the character after the first occurrence of the character 'c' that occurs                        |
| 33073 |                       | before the current cursor position. If the cursor was positioned on the first                            |
| 33074 |                       | character of the line, the terminal shall be alerted and the cursor shall not be                         |
| 33075 |                       | moved. If the character 'c' does not occur in the line before the current cursor                         |
| 33076 |                       | position, the terminal shall be alerted and the cursor shall not be moved.                               |
| 33077 | <b>[count];</b>       | Repeat the most recent <b>f</b> , <b>F</b> , <b>t</b> , or <b>T</b> command. Any number argument on that |
| 33078 |                       | previous command shall be ignored. Errors are those described for the repeated                           |
| 33079 |                       | command.                                                                                                 |
| 33080 | <b>[count],</b>       | Repeat the most recent <b>f</b> , <b>F</b> , <b>t</b> , or <b>T</b> command. Any number argument on that |
| 33081 |                       | previous command shall be ignored. However, reverse the direction of that                                |
| 33082 |                       | command.                                                                                                 |
| 33083 | <b>a</b>              | Enter insert mode after the current cursor position. Characters that are entered                         |
| 33084 |                       | shall be inserted before the next character.                                                             |
| 33085 | <b>A</b>              | Enter insert mode after the end of the current command line.                                             |
| 33086 | <b>i</b>              | Enter insert mode at the current cursor position. Characters that are entered shall                      |
| 33087 |                       | be inserted before the current character.                                                                |
| 33088 | <b>I</b>              | Enter insert mode at the beginning of the current command line.                                          |
| 33089 | <b>R</b>              | Enter insert mode, replacing characters from the command line beginning at the                           |
| 33090 |                       | current cursor position.                                                                                 |
| 33091 | <b>[count]cmotion</b> |                                                                                                          |
| 33092 |                       | Delete the characters between the current cursor position and the cursor position                        |
| 33093 |                       | that would result from the specified <i>motion</i> command. Then enter insert mode                       |
| 33094 |                       | before the first character following any deleted characters. If <i>count</i> is specified, it            |
| 33095 |                       | shall be applied to the motion command. A <i>count</i> shall be ignored for the following                |
| 33096 |                       | motion commands:                                                                                         |
| 33097 |                       | 0    ^    \$    c                                                                                        |
| 33098 |                       | If the <i>motion</i> command is the character 'c', the current command line shall be                     |
| 33099 |                       | cleared and insert mode shall be entered. If the <i>motion</i> command would move the                    |
| 33100 |                       | current cursor position toward the beginning of the command line, the character                          |
| 33101 |                       | under the current cursor position shall not be deleted. If the motion command                            |
| 33102 |                       | would move the current cursor position toward the end of the command line, the                           |
| 33103 |                       | character under the current cursor position shall be deleted. If the <i>count</i> is larger              |
| 33104 |                       | than the number of characters between the current cursor position and the end of                         |
| 33105 |                       | the command line toward which the motion command would move the cursor,                                  |
| 33106 |                       | this shall not be considered an error; all of the remaining characters in the                            |
| 33107 |                       | aforementioned range shall be deleted and insert mode shall be entered. If the                           |
| 33108 |                       | motion command is invalid, the terminal shall be alerted, the cursor shall not be                        |
| 33109 |                       | moved, and no text shall be deleted.                                                                     |

|       |                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|-------|-----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 33110 | <b>C</b>              | Delete from the current character to the end of the line and enter insert mode at the new end-of-line.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 33111 |                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 33112 | <b>S</b>              | Clear the entire edit line and enter insert mode.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 33113 | <b>[count]rc</b>      | Replace the current character with the character 'c'. With a number <i>count</i> , replace the current and the following <i>count</i> -1 characters. After this command, the current cursor position shall be on the last character that was changed. If the <i>count</i> is larger than the number of characters after the cursor, this shall not be considered an error; all of the remaining characters shall be changed.                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 33114 |                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 33115 |                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 33116 |                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 33117 |                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 33118 | <b>[count]_</b>       | Append a <space> after the current character position and then append the last bigword in the previous input line after the <space>. Then enter insert mode after the last character just appended. With a number <i>count</i> , append the <i>count</i> th bigword in the previous line.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 33119 |                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 33120 |                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 33121 |                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 33122 | <b>[count]x</b>       | Delete the character at the current cursor position and place the deleted characters in the save buffer. If the cursor was positioned on the last character of the line, the character shall be deleted and the cursor position shall be moved to the previous character (the new last character). If the <i>count</i> is larger than the number of characters after the cursor, this shall not be considered an error; all the characters from the cursor to the end of the line shall be deleted.                                                                                                                                                                                                                                                                                                                                            |
| 33123 |                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 33124 |                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 33125 |                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 33126 |                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 33127 |                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 33128 | <b>[count]X</b>       | Delete the character before the current cursor position and place the deleted characters in the save buffer. The character under the current cursor position shall not change. If the cursor was positioned on the first character of the line, the terminal shall be alerted, and the <b>X</b> command shall have no effect. If the line contained a single character, the <b>X</b> command shall have no effect. If the line contained no characters, the terminal shall be alerted and the cursor shall not be moved. If the <i>count</i> is larger than the number of characters before the cursor, this shall not be considered an error; all the characters from before the cursor to the beginning of the line shall be deleted.                                                                                                        |
| 33129 |                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 33130 |                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 33131 |                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 33132 |                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 33133 |                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 33134 |                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 33135 |                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 33136 |                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 33137 | <b>[count]dmotion</b> |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 33138 |                       | Delete the characters between the current cursor position and the character position that would result from the <i>motion</i> command. A number <i>count</i> repeats the <i>motion</i> command <i>count</i> times. If the <i>motion</i> command would move toward the beginning of the command line, the character under the current cursor position shall not be deleted. If the <i>motion</i> command is <b>d</b> , the entire current command line shall be cleared. If the <i>count</i> is larger than the number of characters between the current cursor position and the end of the command line toward which the motion command would move the cursor, this shall not be considered an error; all of the remaining characters in the aforementioned range shall be deleted. The deleted characters shall be placed in the save buffer. |
| 33139 |                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 33140 |                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 33141 |                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 33142 |                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 33143 |                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 33144 |                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 33145 |                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 33146 |                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 33147 |                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 33148 | <b>D</b>              | Delete all characters from the current cursor position to the end of the line. The deleted characters shall be placed in the save buffer.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 33149 |                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 33150 | <b>[count]ymotion</b> |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 33151 |                       | Yank (that is, copy) the characters from the current cursor position to the position resulting from the <i>motion</i> command into the save buffer. A number <i>count</i> shall be applied to the <i>motion</i> command. If the <i>motion</i> command would move toward the beginning of the command line, the character under the current cursor position shall not be included in the set of yanked characters. If the <i>motion</i> command is <b>y</b> , the entire current command line shall be yanked into the save buffer. The current cursor position shall be unchanged. If the <i>count</i> is larger than the number of                                                                                                                                                                                                            |
| 33152 |                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 33153 |                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 33154 |                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 33155 |                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 33156 |                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 33157 |                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |

|       |                                |                                                                                               |
|-------|--------------------------------|-----------------------------------------------------------------------------------------------|
| 33158 |                                | characters between the current cursor position and the end of the command line                |
| 33159 |                                | toward which the motion command would move the cursor, this shall not be                      |
| 33160 |                                | considered an error; all of the remaining characters in the aforementioned range              |
| 33161 |                                | shall be yanked.                                                                              |
| 33162 | <b>Y</b>                       | Yank the characters from the current cursor position to the end of the line into the          |
| 33163 |                                | save buffer. The current character position shall be unchanged.                               |
| 33164 | <b>[count]p</b>                | Put a copy of the current contents of the save buffer after the current cursor                |
| 33165 |                                | position. The current cursor position shall be advanced to the last character put             |
| 33166 |                                | from the save buffer. A <i>count</i> shall indicate how many copies of the save buffer        |
| 33167 |                                | shall be put.                                                                                 |
| 33168 | <b>[count]P</b>                | Put a copy of the current contents of the save buffer before the current cursor               |
| 33169 |                                | position. The current cursor position shall be moved to the last character put from           |
| 33170 |                                | the save buffer. A <i>count</i> shall indicate how many copies of the save buffer shall be    |
| 33171 |                                | put.                                                                                          |
| 33172 | <b>u</b>                       | Undo the last command that change the edit line. This operation shall not undo the            |
| 33173 |                                | copy of any command line to the edit line.                                                    |
| 33174 | <b>U</b>                       | Undo all changes made to the edit line. This operation shall not undo the copy of             |
| 33175 |                                | any command line to the edit line.                                                            |
| 33176 | <b>[count]k</b>                |                                                                                               |
| 33177 | <b>[count]-</b>                | Set the current command line to be the <i>count</i> th previous command line in the shell     |
| 33178 |                                | command history. If <i>count</i> is not specified, it shall default to 1. The cursor shall be |
| 33179 |                                | positioned on the first character of the new command. If a <b>k</b> or <b>-</b> command would |
| 33180 |                                | retreat past the maximum number of commands in effect for this shell (affected by             |
| 33181 |                                | the <i>HISTSIZE</i> environment variable), the terminal shall be alerted, and the             |
| 33182 |                                | command shall have no effect.                                                                 |
| 33183 | <b>[count]j</b>                |                                                                                               |
| 33184 | <b>[count]+</b>                | Set the current command line to be the <i>count</i> th next command line in the shell         |
| 33185 |                                | command history. If <i>count</i> is not specified, it shall default to 1. The cursor shall be |
| 33186 |                                | positioned on the first character of the new command. If a <b>j</b> or <b>+</b> command       |
| 33187 |                                | advances past the edit line, the current command line shall be restored to the edit           |
| 33188 |                                | line and terminal shall be alerted.                                                           |
| 33189 | <b>[number]G</b>               | Set the current command line to be the oldest command line stored in the shell                |
| 33190 |                                | command history. With a number <i>number</i> , set the current command line to be the         |
| 33191 |                                | command line <i>number</i> in the history. If command line <i>number</i> does not exist, the  |
| 33192 |                                | terminal shall be alerted and the command line shall not be changed.                          |
| 33193 | <b>/pattern&lt;newline&gt;</b> |                                                                                               |
| 33194 |                                | Move backwards through the command history, searching for the specified                       |
| 33195 |                                | pattern, beginning with the previous command line. Patterns use the pattern                   |
| 33196 |                                | matching notation described in Section 2.13 (on page 2264), except that the '^'               |
| 33197 |                                | character shall have special meaning when it appears as the first character of                |
| 33198 |                                | <i>pattern</i> . In this case, the '^' is discarded and the characters after the '^' shall be |
| 33199 |                                | matched only at the beginning of a line. Commands in the command history shall                |
| 33200 |                                | be treated as strings, not as filenames. If the pattern is not found, the current             |
| 33201 |                                | command line shall be unchanged and the terminal is alerted. If it is found in a              |
| 33202 |                                | previous line, the current command line shall be set to that line and the cursor              |
| 33203 |                                | shall be set to the first character of the new command line.                                  |

33204 If *pattern* is empty, the last non-empty pattern provided to / or ? shall be used. If  
 33205 there is no previous non-empty pattern, the terminal shall be alerted and the  
 33206 current command line shall remain unchanged.

33207 **?*pattern*<newline>**

33208 Move forwards through the command history, searching for the specified pattern,  
 33209 beginning with the next command line. Patterns use the pattern matching notation  
 33210 described in Section 2.13 (on page 2264), except that the '^' character shall have |  
 33211 special meaning when it appears as the first character of *pattern*. In this case, the |  
 33212 '^' is discarded and the characters after the '^' shall be matched only at the |  
 33213 beginning of a line. Commands in the command history shall be treated as strings, |  
 33214 not as filenames. If the pattern is not found, the current command line shall be |  
 33215 unchanged and the terminal alerted. If it is found in a following line, the current  
 33216 command line shall be set to that line and the cursor shall be set to the first  
 33217 character of the new command line.

33218 If *pattern* is empty, the last non-empty pattern provided to / or ? shall be used. If  
 33219 there is no previous non-empty pattern, the terminal shall be alerted and the  
 33220 current command line shall remain unchanged.

33221 **n** Repeat the most recent / or ? command. If there is no previous / or ?, the terminal  
 33222 shall be alerted and the current command line shall remain unchanged.

33223 **N** Repeat the most recent / or ? command, reversing the direction of the search. If  
 33224 there is no previous / or ?, the terminal shall be alerted and the current command  
 33225 line shall remain unchanged.

#### 33226 EXIT STATUS

33227 The following exit values shall be returned:

33228 0 The script to be executed consisted solely of zero or more blank lines or comments, or  
 33229 both.

33230 1-125 A non-interactive shell detected a syntax, redirection or variable assignment error.

33231 127 A specified *command\_file* could not be found by a non-interactive shell.

33232 Otherwise, the shell shall return the exit status of the last command it invoked or attempted to  
 33233 invoke (see also the *exit* utility in Section 2.14 (on page 2266)).

#### 33234 CONSEQUENCES OF ERRORS

33235 See Section 2.8.1 (on page 2247).

#### 33236 APPLICATION USAGE

33237 Standard input and standard error are the files that determine whether a shell is interactive  
 33238 when **-i** is not specified. For example:

33239 sh > file

33240 and:

33241 sh 2> file

33242 create interactive and non-interactive shells, respectively. Although both accept terminal input,  
 33243 the results of error conditions are different, as described in Section 2.8.1 (on page 2247); in the  
 33244 second example a redirection error encountered by a special built-in utility aborts the shell.

33245 A conforming application must protect its first operand, if it starts with a plus sign, by preceding |  
 33246 it with the "--" argument that denotes the end of the options.

33247 Applications should note that the standard *PATH* to the shell cannot be assumed to be either  
 33248 **/bin/sh** or **/usr/bin/sh**, and should be determined by interrogation of the *PATH* returned by

33249 *getconf* *PATH*, ensuring that the returned pathname is an absolute pathname and not a shell built  
 33250 in.

33251 For example, to determine the location of the standard *sh* utility:

```
33252 command -v sh
```

33253 On some implementations this might return:

```
33254 /usr/xpg4/bin/sh
```

33255 Furthermore, on systems that support executable scripts (the "#!" construct), it is  
 33256 recommended that applications using executable scripts install them using *getconf -v* to  
 33257 determine the shell pathname and update the "#!" script appropriately as it is being installed  
 33258 (for example, with *sed*). For example:

```
33259 #
33260 # Installation time script to install correct POSIX shell pathname
33261 #
33262 # Get list of paths to check
33263 #
33264 Sifs=$IFS
33265 IFS=:
33266 set $(getconf PATH)
33267 IFS=$Sifs
33268 #
33269 # Check each path for 'sh'
33270 #
33271 for i in $@
33272 do
33273     if [ -f ${i}/sh ];
33274     then
33275         Pshell=${i}/sh
33276     fi
33277 done
33278 #
33279 # This is the list of scripts to update. They should be of the
33280 # form '${name}.source' and will be transformed to '${name}'.
33281 # Each script should begin:
33282 #
33283 # !INSTALLSHELLPATH -p
33284 #
33285 scripts="a b c"
33286 #
33287 # Transform each script
33288 #
33289 for i in ${scripts}
33290 do
33291     sed -e "s|INSTALLSHELLPATH|${Pshell}|" < ${i}.source > ${i}
33292 done
```

### 33293 EXAMPLES

33294 1. Execute a shell command from a string:

```
33295 sh -c "cat myfile"
```

33296 2. Execute a shell script from a file in the current directory:

33297 `sh my_shell_cmds`

### 33298 RATIONALE

33299 The *sh* utility and the *set* special built-in utility share a common set of options.

33300 The KornShell ignores the contents of *IFS* upon entry to the script. A conforming application  
33301 cannot rely on importing *IFS*. One justification for this, beyond security considerations, is to  
33302 assist possible future shell compilers. Allowing *IFS* to be imported from the environment  
33303 prevents many optimizations that might otherwise be performed via dataflow analysis of the  
33304 script itself.

33305 The text in the STDIN section about non-blocking reads concerns an instance of *sh* that has been  
33306 invoked, probably by a C-language program, with standard input that has been opened using  
33307 the `O_NONBLOCK` flag; see *open()* in the System Interfaces volume of IEEE Std 1003.1-200x. If  
33308 the shell did not reset this flag, it would immediately terminate because no input data would be  
33309 available yet and that would be considered the same as end-of-file.

33310 The options associated with a *restricted shell* (command name *rsh* and the `-r` option) were  
33311 excluded because the standard developers considered that the implied level of security could  
33312 not be achieved and they did not want to raise false expectations.

33313 On systems that support set-user-ID scripts, a historical trapdoor has been to link a script to the  
33314 name `-i`. When it is called by a sequence such as

33315 `sh -`

33316 or by:

33317 `#! usr/bin/sh -`

33318 the historical systems have assumed that no option letters follow. Thus, this volume of  
33319 IEEE Std 1003.1-200x allows the single hyphen to mark the end of the options, in addition to the  
33320 use of the regular `---` argument, because it was considered that the older practice was so  
33321 pervasive. An alternative approach is taken by the KornShell, where real and effective  
33322 user/group IDs must match for an interactive shell; this behavior is specifically allowed by this  
33323 volume of IEEE Std 1003.1-200x.

33324 **Note:** There are other problems with set-user-ID scripts that the two approaches described here do  
33325 not resolve.

33326 The initialization process for the history file can be dependent on the system start-up files, in  
33327 that they may contain commands that effectively preempt the user's settings of *HISTFILE* and  
33328 *HISTSIZE*. For example, function definition commands are recorded in the history file, unless  
33329 the *set -o nolog* option is set. If the system administrator includes function definitions in some  
33330 system start-up file called before the *ENV* file, the history file is initialized before the user gets a  
33331 chance to influence its characteristics.) In some historical shells, the history file is initialized just  
33332 after the *ENV* file has been processed. Therefore, it is implementation-defined whether changes  
33333 made to *HISTFILE* after the history file has been initialized are effective.

33334 The default messages for the various *MAIL*-related messages are unspecified because they vary  
33335 across implementations. Typical messages are:

33336 `"you have mail\n"`

33337 or:

33338 `"you have new mail\n"`

33339 It is important that the descriptions of command line editing refer to the same shell as that in  
 33340 IEEE Std 1003.1-200x so that interactive users can also be application programmers without  
 33341 having to deal with programmatic differences in their two environments. It is also essential that  
 33342 the utility name *sh* be specified because this explicit utility name is too firmly rooted in historical  
 33343 practice of application programs for it to change.

33344 Consideration was given to mandating a diagnostic message when attempting to set *vi*-mode on  
 33345 terminals that do not support command line editing. However, it is not historical practice for the  
 33346 shell to be cognizant of all terminal types and thus be able to detect inappropriate terminals in  
 33347 all cases. Implementations are encouraged to supply diagnostics in this case whenever possible,  
 33348 rather than leaving the user in a state where editing commands work incorrectly.

33349 In early proposals, the KornShell-derived *emacs* mode of command line editing was included,  
 33350 even though the *emacs* editor itself was not. The community of *emacs* proponents was adamant  
 33351 that the full *emacs* editor not be included in earlier versions of IEEE Std 1003.1 because they were  
 33352 concerned that an attempt to standardize this very powerful environment would encourage  
 33353 vendors to ship versions conforming strictly to earlier versions of IEEE Std 1003.1, but lacking  
 33354 the extensibility required by the community. The author of the original *emacs* program also  
 33355 expressed his desire to omit the program. Furthermore, there were a number of historical  
 33356 systems that did not include *emacs*, or included it without supporting it, but there were very few  
 33357 that did not include and support *vi*. The shell *emacs* command line editing mode was finally  
 33358 omitted from earlier versions of IEEE Std 1003.1 because it became apparent that the KornShell  
 33359 version and the editor being distributed with the GNU system had diverged in some respects.  
 33360 The author of *emacs* requested that the POSIX *emacs* mode either be deleted or have a significant  
 33361 number of unspecified conditions. Although the KornShell author agreed to consider changes to  
 33362 bring the shell into alignment, the standard developers decided to defer specification at this  
 33363 time, rather than attempting to agree on a specific subset of *emacs* late within the development of  
 33364 earlier versions of IEEE Std 1003.1. At the time, it was assumed that convergence on an  
 33365 acceptable definition would occur for a subsequent draft, but that has not happened, and there  
 33366 appears to be no impetus to do so. In any case, implementations are free to offer additional  
 33367 command line editing modes based on the exact models of editors their users are most  
 33368 comfortable with.

33369 Early proposals had the following list entry in **vi Line Editing Insert Mode** (on page 3053):

33370 \ If followed by the *erase* or *kill* character, that character shall be inserted into the input line.  
 33371 Otherwise, the backslash itself shall be inserted into the input line.

33372 However, this is not actually a feature of *sh* command line editing insert mode, but one of some  
 33373 historical terminal line drivers. Some conforming implementations continue to do this when the  
 33374 *stty ixten* flag is set.

#### 33375 FUTURE DIRECTIONS

33376 None.

#### 33377 SEE ALSO

33378 *cd*, *echo*, *pwd*, *test*, *umask*, the System Interfaces volume of IEEE Std 1003.1-200x, *dup()*, *exec*,  
 33379 *exit()*, *fork()*, *pipe()*, *signal()*, *system()*, *ulimit()*, *umask()*, *wait()*

#### 33380 CHANGE HISTORY

33381 First released in Issue 2.

#### 33382 Issue 5

33383 FUTURE DIRECTIONS section added.

33384 Text is added to the DESCRIPTION for the Large File Summit proposal.

33385 **Issue 6**

- 33386 The Open Group Corrigendum U029/2 is applied, correcting the second SYNOPSIS.
- 33387 The Open Group Corrigendum U027/3 is applied, correcting a typographical error.
- 33388 The following new requirements on POSIX implementations derive from alignment with the  
33389 Single UNIX Specification:
- 33390 • The option letters derived from the set special built-in are also accepted with a leading plus  
33391 sign ('+').
  - 33392 • Large file extensions are added:
    - 33393 — Pathname expansion does not fail due to the size of a file.
    - 33394 — Shell input and output redirections have an implementation-defined offset maximum  
33395 that is established in the open file description.
- 33396 In the ENVIRONMENT VARIABLES section, the text “user’s home directory” is updated to  
33397 “directory referred to by the *HOME* environment variable”.
- 33398 Descriptions for the *ENV* and *PWD* environment variables are included to align with the  
33399 IEEE P1003.2b draft standard.
- 33400 The normative text is reworded to avoid use of the term “must” for application requirements.

33401 **NAME**

33402           sleep — suspend execution for an interval

33403 **SYNOPSIS**33404           sleep *time*33405 **DESCRIPTION**33406           The *sleep* utility shall suspend execution for at least the integral number of seconds specified by  
33407           the *time* operand.33408 **OPTIONS**

33409           None.

33410 **OPERANDS**

33411           The following operand shall be supported:

33412           *time*           A non-negative decimal integer specifying the number of seconds for which to  
33413           suspend execution.33414 **STDIN**

33415           Not used.

33416 **INPUT FILES**

33417           None.

33418 **ENVIRONMENT VARIABLES**33419           The following environment variables shall affect the execution of *sleep*:33420           *LANG*           Provide a default value for the internationalization variables that are unset or null.  
33421           (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
33422           Internationalization Variables for the precedence of internationalization variables  
33423           used to determine the values of locale categories.)33424           *LC\_ALL*       If set to a non-empty string value, override the values of all the other  
33425           internationalization variables.33426           *LC\_CTYPE*   Determine the locale for the interpretation of sequences of bytes of text data as  
33427           characters (for example, single-byte as opposed to multi-byte characters in  
33428           arguments).33429           *LC\_MESSAGES*33430           Determine the locale that should be used to affect the format and contents of  
33431           diagnostic messages written to standard error.33432 **XSI**           *NLSPATH*   Determine the location of message catalogs for the processing of *LC\_MESSAGES*.33433 **ASYNCHRONOUS EVENTS**33434           If the *sleep* utility receives a SIGALRM signal, one of the following actions shall be taken:

- 33435           1. Terminate normally with a zero exit status.
- 
- 33436           2. Effectively ignore the signal.
- 
- 33437           3. Provide the default behavior for signals described in the ASYNCHRONOUS EVENTS
- 
- 33438           section of Section 1.11 (on page 2221). This could include terminating with a non-zero exit
- 
- 33439           status.

33440           The *sleep* utility shall take the standard action for all other signals.

33441 **STDOUT**

33442 Not used.

33443 **STDERR**

33444 The standard error shall be used only for diagnostic messages.

33445 **OUTPUT FILES**

33446 None.

33447 **EXTENDED DESCRIPTION**

33448 None.

33449 **EXIT STATUS**

33450 The following exit values shall be returned:

33451 0 The execution was successfully suspended for at least *time* seconds, or a SIGALRM signal  
33452 was received. See the ASYNCHRONOUS EVENTS section.

33453 &gt;0 An error occurred.

33454 **CONSEQUENCES OF ERRORS**

33455 Default.

33456 **APPLICATION USAGE**

33457 None.

33458 **EXAMPLES**33459 The *sleep* utility can be used to execute a command after a certain amount of time, as in:33460 (*sleep* 105; *command*) &

33461 or to execute a command every so often, as in:

33462 while true  
33463 do  
33464 *command*  
33465 *sleep* 37  
33466 done33467 **RATIONALE**33468 The exit status is allowed to be zero when *sleep* is interrupted by the SIGALRM signal because  
33469 most implementations of this utility rely on the arrival of that signal to notify them that the  
33470 requested finishing time has been successfully attained. Such implementations thus do not  
33471 distinguish this situation from the successful completion case. Other implementations are  
33472 allowed to catch the signal and go back to sleep until the requested time expires or to provide  
33473 the normal signal termination procedures.33474 As with all other utilities that take integral operands and do not specify subranges of allowed  
33475 values, *sleep* is required by this volume of IEEE Std 1003.1-200x to deal with *time* requests of up  
33476 to 2 147 483 647 seconds. This may mean that some implementations have to make multiple calls  
33477 to the delay mechanism of the underlying operating system if its argument range is less than  
33478 this.33479 **FUTURE DIRECTIONS**

33480 None.

33481 **SEE ALSO**33482 *wait*, the System Interfaces volume of IEEE Std 1003.1-200x, *alarm()*, *sleep()*

33483 **CHANGE HISTORY**

33484 First released in Issue 2.

## 33485 NAME

33486 sort — sort, merge, or sequence check text files

## 33487 SYNOPSIS

33488 sort [-m][-o *output*][-bdfinru][-t *char*][-k *keydef*]... [*file*...]

33489 sort -c [-bdfinru][-t *char*][-k *keydef*][*file*]

## 33490 DESCRIPTION

33491 The *sort* utility shall perform one of the following functions:

- 33492 1. Sort lines of all the named files together and write the result to the specified output.
- 33493 2. Merge lines of all the named (presorted) files together and write the result to the specified  
33494 output.
- 33495 3. Check that a single input file is correctly presorted.

33496 Comparisons shall be based on one or more sort keys extracted from each line of input (or, if no  
33497 sort keys are specified, the entire line up to, but not including, the terminating <newline>), and  
33498 shall be performed using the collating sequence of the current locale.

## 33499 OPTIONS

33500 The *sort* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section  
33501 12.2, Utility Syntax Guidelines, and the **-k** *keydef* option should follow the **-b**, **-d**, **-f**, **-i**, **-n**, and  
33502 **-r** options.

33503 The following options shall be supported:

33504 **-c** Check that the single input file is ordered as specified by the arguments and the  
33505 collating sequence of the current locale. No output shall be produced; only the exit  
33506 code shall be affected.

33507 **-m** Merge only; the input file shall be assumed to be already sorted.

33508 **-o** *output* Specify the name of an output file to be used instead of the standard output. This  
33509 file can be the same as one of the input *files*.

33510 **-u** Unique: suppress all but one in each set of lines having equal keys. If used with  
33511 the **-c** option, check that there are no lines with duplicate keys, in addition to  
33512 checking that the input file is sorted.

33513 The following options shall override the default ordering rules. When ordering options appear  
33514 independent of any key field specifications, the requested field ordering rules shall be applied  
33515 globally to all sort keys. When attached to a specific key (see **-k**), the specified ordering options  
33516 shall override all global ordering options for that key.

33517 **-d** Specify that only <blank>s and alphanumeric characters, according to the current  
33518 setting of *LC\_CTYPE*, shall be significant in comparisons. The behavior is  
33519 undefined for a sort key to which **-i** or **-n** also applies.

33520 **-f** Consider all lowercase characters that have uppercase equivalents, according to  
33521 the current setting of *LC\_CTYPE*, to be the uppercase equivalent for the purposes  
33522 of comparison.

33523 **-i** Ignore all characters that are non-printable, according to the current setting of  
33524 *LC\_CTYPE*.

33525 **-n** Restrict the sort key to an initial numeric string, consisting of optional <blank>s,  
33526 optional minus sign, and zero or more digits with an optional radix character and  
33527 thousands separators (as defined in the current locale), which shall be sorted by

- 33528 arithmetic value. An empty digit string shall be treated as zero. Leading zeros and  
33529 signs on zeros shall not affect ordering.
- 33530 **-r** Reverse the sense of comparisons.
- 33531 The treatment of field separators can be altered using the options:
- 33532 **-b** Ignore leading <blank>s when determining the starting and ending positions of a  
33533 restricted sort key. If the **-b** option is specified before the first **-k** option, it shall be  
33534 applied to all **-k** options. Otherwise, the **-b** option can be attached independently  
33535 to each **-k** *field\_start* or *field\_end* option-argument (see below).
- 33536 **-t char** Use *char* as the field separator character; *char* shall not be considered to be part of a  
33537 field (although it can be included in a sort key). Each occurrence of *char* shall be  
33538 significant (for example, <*char*><*char*> delimits an empty field). If **-t** is not  
33539 specified, <blank>s shall be used as default field separators; each maximal non-  
33540 empty sequence of <blank>s that follows a non-<blank> shall be a field separator.
- 33541 Sort keys can be specified using the options:
- 33542 **-k keydef** The *keydef* argument is a restricted sort key field definition. The format of this  
33543 definition is:
- 33544 *field\_start*[*type*][,*field\_end*[*type*]]
- 33545 where *field\_start* and *field\_end* define a key field restricted to a portion of the line  
33546 (see the EXTENDED DESCRIPTION section), and *type* is a modifier from the list of  
33547 characters 'b', 'd', 'f', 'i', 'n', 'r'. The 'b' modifier shall behave like the  
33548 **-b** option, but shall apply only to the *field\_start* or *field\_end* to which it is attached.  
33549 The other modifiers shall behave like the corresponding options, but shall apply  
33550 only to the key field to which they are attached; they shall have this effect if  
33551 specified with *field\_start*, *field\_end*, or both. If any modifier is attached to a  
33552 *field\_start* or to a *field\_end*, no option shall apply to either. Implementations shall  
33553 support at least nine occurrences of the **-k** option, which shall be significant in  
33554 command line order. If no **-k** option is specified, a default sort key of the entire  
33555 line shall be used.
- 33556 When there are multiple key fields, later keys shall be compared only after all  
33557 earlier keys compare equal. Except when the **-u** option is specified, lines that  
33558 otherwise compare equal shall be ordered as if none of the options **-d**, **-f**, **-i**, **-n**, or  
33559 **-k** were present (but with **-r** still in effect, if it was specified) and with all bytes in  
33560 the lines significant to the comparison. The order in which lines that still compare  
33561 equal are written is unspecified.
- 33562 **OPERANDS**
- 33563 The following operand shall be supported:
- 33564 *file* A pathname of a file to be sorted, merged, or checked. If no *file* operands are  
33565 specified, or if a *file* operand is '-', the standard input shall be used.
- 33566 **STDIN**
- 33567 The standard input shall be used only if no *file* operands are specified, or if a *file* operand is '-'.  
33568 See the INPUT FILES section.
- 33569 **INPUT FILES**
- 33570 The input files shall be text files, except that the *sort* utility shall add a <newline> to the end of a  
33571 file ending with an incomplete last line.

## 33572 ENVIRONMENT VARIABLES

33573 The following environment variables shall affect the execution of *sort*:

33574 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 33575 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
 33576 Internationalization Variables for the precedence of internationalization variables  
 33577 used to determine the values of locale categories.)

33578 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 33579 internationalization variables.

33580 *LC\_COLLATE*

33581 Determine the locale for ordering rules.

33582 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 33583 characters (for example, single-byte as opposed to multi-byte characters in  
 33584 arguments and input files) and the behavior of character classification for the *-b*,  
 33585 *-d*, *-f*, *-i*, and *-n* options.

33586 *LC\_MESSAGES*

33587 Determine the locale that should be used to affect the format and contents of  
 33588 diagnostic messages written to standard error.

33589 *LC\_NUMERIC*

33590 Determine the locale for the definition of the radix character and thousands  
 33591 separator for the *-n* option.

33592 XSI *NLS\_PATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

## 33593 ASYNCHRONOUS EVENTS

33594 Default.

33595 *STDOUT*

33596 Unless the *-o* or *-c* options are in effect, the standard output shall contain the sorted input.

33597 *STDERR*

33598 The standard error shall be used for diagnostic messages. A warning message about correcting |  
 33599 an incomplete last line of an input file may be generated, but need not affect the final exit status. |

## 33600 OUTPUT FILES

33601 If the *-o* option is in effect, the sorted input shall be written to the file *output*.

## 33602 EXTENDED DESCRIPTION

33603 The notation:

33604 *-k field\_start[type][,field\_end[type]]*

33605 shall define a key field that begins at *field\_start* and ends at *field\_end* inclusive, unless *field\_start*  
 33606 falls beyond the end of the line or after *field\_end*, in which case the key field is empty. A missing  
 33607 *field\_end* shall mean the last character of the line.

33608 A field comprises a maximal sequence of non-separating characters and, in the absence of option  
 33609 *-t*, any preceding field separator.

33610 The *field\_start* portion of the *keydef* option-argument shall have the form:

33611 *field\_number[.first\_character]*

33612 Fields and characters within fields shall be numbered starting with 1. The *field\_number* and  
 33613 *first\_character* pieces, interpreted as positive decimal integers, shall specify the first character to  
 33614 be used as part of a sort key. If *first\_character* is omitted, it shall refer to the first character of the

33615 field.

33616 The *field\_end* portion of the *keydef* option-argument shall have the form:

33617 *field\_number*[*.last\_character*]

33618 The *field\_number* shall be as described above for *field\_start*. The *last\_character* piece, interpreted  
 33619 as a non-negative decimal integer, shall specify the last character to be used as part of the sort  
 33620 key. If *last\_character* evaluates to zero or *.last\_character* is omitted, it shall refer to the last  
 33621 character of the field specified by *field\_number*.

33622 If the **-b** option or **b** type modifier is in effect, characters within a field shall be counted from the  
 33623 first non-<blank> in the field. (This shall apply separately to *first\_character* and *last\_character*.)

#### 33624 EXIT STATUS

33625 The following exit values shall be returned:

33626 0 All input files were output successfully, or **-c** was specified and the input file was correctly  
 33627 sorted.

33628 1 Under the **-c** option, the file was not ordered as specified, or if the **-c** and **-u** options were  
 33629 both specified, two input lines were found with equal keys.

33630 >1 An error occurred.

#### 33631 CONSEQUENCES OF ERRORS

33632 Default.

#### 33633 APPLICATION USAGE

33634 The default value for **-t**, <blank>, has different properties from, for example, **-t**"<space>". If a  
 33635 line contains:

33636 <space><space>foo

33637 the following treatment would occur with default separation as opposed to specifically selecting  
 33638 a <space>:

| Field | Default           | <b>-t</b> "<space>" |
|-------|-------------------|---------------------|
| 1     | <space><space>foo | <i>empty</i>        |
| 2     | <i>empty</i>      | <i>empty</i>        |
| 3     | <i>empty</i>      | foo                 |

33643 The leading field separator itself is included in a field when **-t** is not used. For example, this  
 33644 command returns an exit status of zero, meaning the input was already sorted:

33645 `sort -c -k 2 <<eof`  
 33646 `y<tab>b`  
 33647 `x<space>a`  
 33648 `eof`

33649 (assuming that a <tab> precedes the <space> in the current collating sequence). The field  
 33650 separator is not included in a field when it is explicitly set via **-t**. This is historical practice and  
 33651 allows usage such as:

33652 `sort -t "|" -k 2n <<eof`  
 33653 `Atlanta|425022|Georgia`  
 33654 `Birmingham|284413|Alabama`  
 33655 `Columbia|100385|South Carolina`  
 33656 `eof`

33657 where the second field can be correctly sorted numerically without regard to the non-numeric  
33658 field separator.

33659 The wording in the OPTIONS section clarifies that the **-b**, **-d**, **-f**, **-i**, **-n**, and **-r** options have to  
33660 come before the first sort key specified if they are intended to apply to all specified keys. The  
33661 way it is described in this volume of IEEE Std 1003.1-200x matches historical practice, not  
33662 historical documentation. The results are unspecified if these options are specified after a **-k**  
33663 option.

33664 The **-f** option might not work as expected in locales where there is not a one-to-one mapping  
33665 between an uppercase and a lowercase letter.

#### 33666 EXAMPLES

33667 1. The following command sorts the contents of **infile** with the second field as the sort key:

```
33668 sort -k 2,2 infile
```

33669 2. The following command sorts, in reverse order, the contents of **infile1** and **infile2**, placing  
33670 the output in **outfile** and using the second character of the second field as the sort key  
33671 (assuming that the first character of the second field is the field separator):

```
33672 sort -r -o outfile -k 2.2,2.2 infile1 infile2
```

33673 3. The following command sorts the contents of **infile1** and **infile2** using the second non-  
33674 <blank> of the second field as the sort key:

```
33675 sort -k 2.2b,2.2b infile1 infile2
```

33676 4. The following command prints the System V password file (user database) sorted by the  
33677 numeric user ID (the third colon-separated field):

```
33678 sort -t : -k 3,3n /etc/passwd
```

33679 5. The following command prints the lines of the already sorted file **infile**, suppressing all  
33680 but one occurrence of lines having the same third field:

```
33681 sort -um -k 3.1,3.0 infile
```

#### 33682 RATIONALE

33683 Examples in some historical documentation state that options **-um** with one input file keep the  
33684 first in each set of lines with equal keys. This behavior was deemed to be an implementation  
33685 artifact and was not standardized.

33686 The **-z** option was omitted; it is not standard practice on most systems and is inconsistent with  
33687 using *sort* to sort several files individually and then merge them together. The text concerning **-z**  
33688 in historical documentation appeared to require implementations to determine the proper buffer  
33689 length during the sort phase of operation, but not during the merge.

33690 The **-y** option was omitted because of non-portability. The **-M** option, present in System V, was  
33691 omitted because of non-portability in international usage.

33692 An undocumented **-T** option exists in some implementations. It is used to specify a directory for  
33693 intermediate files. Implementations are encouraged to support the use of the *TMPDIR*  
33694 environment variable instead of adding an option to support this functionality.

33695 The **-k** option was added to satisfy two objections. First, the zero-based counting used by *sort* is  
33696 not consistent with other utility conventions. Second, it did not meet syntax guideline  
33697 requirements.

33698 Historical documentation indicates that “setting **-n** implies **-b**”. The description of **-n** already  
33699 states that optional leading <blank>s are tolerated in doing the comparison. If **-b** is enabled,

33700 rather than implied, by **-n**, this has unusual side effects. When a character offset is used in a  
33701 column of numbers (for example, to sort modulo 100), that offset is measured relative to the  
33702 most significant digit, not to the column. Based upon a recommendation from the author of the  
33703 original *sort* utility, the **-b** implication has been omitted from this volume of  
33704 IEEE Std 1003.1-200x, and an application wishing to achieve the previously mentioned side  
33705 effects has to code the **-b** flag explicitly.

33706 **FUTURE DIRECTIONS**

33707 None.

33708 **SEE ALSO**

33709 *comm*, *join*, *uniq*, the System Interfaces volume of IEEE Std 1003.1-200x, *toupper()*

33710 **CHANGE HISTORY**

33711 First released in Issue 2.

33712 **Issue 6**

33713 IEEE PASC Interpretation 1003.2 #174 is applied, updating the DESCRIPTION of comparisons.

33714 IEEE PASC Interpretation 1003.2 #168 is applied.

## 33715 NAME

33716 split — split files into pieces

## 33717 SYNOPSIS

33718 UP `split [-l line_count][-a suffix_length][file[name]]`33719 `split -b n[k|m][-a suffix_length][file[name]]`

33720

## 33721 DESCRIPTION

33722 The *split* utility shall read an input file and write one or more output files. The default size of  
 33723 each output file shall be 1 000 lines. The size of the output files can be modified by specification  
 33724 of the **-b** or **-l** options. Each output file shall be created with a unique suffix. The suffix shall  
 33725 consist of exactly *suffix\_length* lowercase letters from the POSIX locale. The letters of the suffix  
 33726 shall be used as if they were a base-26 digit system, with the first suffix to be created consisting  
 33727 of all 'a' characters, the second with a 'b' replacing the last 'a', and so on, until a name of all  
 33728 'z' characters is created. By default, the names of the output files shall be 'x', followed by a  
 33729 two-character suffix from the character set as described above, starting with "aa", "ab", "ac",  
 33730 and so on, and continuing until the suffix "zz", for a maximum of 676 files.

33731 If the number of files required exceeds the maximum allowed by the suffix length provided,  
 33732 such that the last allowable file would be larger than the requested size, the *split* utility shall fail  
 33733 after creating the last file with a valid suffix; *split* shall not delete the files it created with valid  
 33734 suffixes. If the file limit is not exceeded, the last file created shall contain the remainder of the  
 33735 input file, and may be smaller than the requested size.

## 33736 OPTIONS

33737 The *split* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section  
 33738 12.2, Utility Syntax Guidelines.

33739 The following options shall be supported:

33740 **-a *suffix\_length***

33741 Use *suffix\_length* letters to form the suffix portion of the filenames of the split file.  
 33742 If **-a** is not specified, the default suffix length shall be two. If the sum of the *name*  
 33743 operand and the *suffix\_length* option-argument would create a filename exceeding  
 33744 {NAME\_MAX} bytes, an error shall result; *split* shall exit with a diagnostic  
 33745 message and no files shall be created.

33746 **-b *n*** Split a file into pieces *n* bytes in size.

33747 **-b *nk*** Split a file into pieces *n*\*1024 bytes in size.

33748 **-b *nm*** Split a file into pieces *n*\*1 048 576 bytes in size.

33749 **-l *line\_count*** Specify the number of lines in each resulting file piece. The *line\_count* argument is  
 33750 an unsigned decimal integer. The default is 1 000. If the input does not end with a  
 33751 <newline>, the partial line shall be included in the last output file.

## 33752 OPERANDS

33753 The following operands shall be supported:

33754 ***file*** The pathname of the ordinary file to be split. If no input file is given or *file* is '-',  
 33755 the standard input shall be used.

33756 ***name*** The prefix to be used for each of the files resulting from the split operation. If no  
 33757 *name* argument is given, 'x' shall be used as the prefix of the output files. The  
 33758 combined length of the basename of *prefix* and *suffix\_length* cannot exceed  
 33759 {NAME\_MAX} bytes. See the OPTIONS section.

33760 **STDIN**

33761 See the INPUT FILES section.

33762 **INPUT FILES**

33763 Any file can be used as input.

33764 **ENVIRONMENT VARIABLES**33765 The following environment variables shall affect the execution of *split*:

33766 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 33767 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
 33768 Internationalization Variables for the precedence of internationalization variables  
 33769 used to determine the values of locale categories.)

33770 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 33771 internationalization variables.

33772 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 33773 characters (for example, single-byte as opposed to multi-byte characters in  
 33774 arguments and input files).

33775 *LC\_MESSAGES*

33776 Determine the locale that should be used to affect the format and contents of  
 33777 diagnostic messages written to standard error.

33778 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

33779 **ASYNCHRONOUS EVENTS**

33780 Default.

33781 **STDOUT**

33782 Not used.

33783 **STDERR**

33784 The standard error shall be used only for diagnostic messages. |

33785 **OUTPUT FILES**

33786 The output files contain portions of the original input file; otherwise, unchanged.

33787 **EXTENDED DESCRIPTION**

33788 None.

33789 **EXIT STATUS**

33790 The following exit values shall be returned:

33791 0 Successful completion.

33792 &gt;0 An error occurred.

33793 **CONSEQUENCES OF ERRORS**

33794 Default.

33795 **APPLICATION USAGE**

33796 None.

33797 **EXAMPLES**33798 In the following examples **foo** is a text file that contains 5 000 lines.33799 1. Create five files, **xaa**, **xab**, **xac**, **xad**, and **xae**:33800 `split foo`33801 2. Create five files, but the suffixed portion of the created files consists of three letters, **xaaa**,  
33802 **xaab**, **xaac**, **xaad**, and **xaae**:33803 `split -a 3 foo`33804 3. Create three files with four-letter suffixes and a supplied prefix, **bar\_aaaa**, **bar\_aaab**, and  
33805 **bar\_aaac**:33806 `split -a 4 -l 2000 foo bar_`33807 4. Create as many files as are necessary to contain at most 20\*1 024 bytes, each with the  
33808 default prefix of **x** and a five-letter suffix:33809 `split -a 5 -b 20k foo`33810 **RATIONALE**33811 The **-b** option was added to provide a mechanism for splitting files other than by lines. While  
33812 most uses of the **-b** option are for transmitting files over networks, some believed it would have  
33813 additional uses.33814 The **-a** option was added to overcome the limitation of being able to create only 676 files.33815 Consideration was given to deleting this utility, using the rationale that the function provided  
33816 by this utility is available via the *csplit* utility (see *csplit* (on page 2480)). Upon reconsideration of  
33817 the purpose of the User Portability Extension, it was decided to retain both this utility and the  
33818 *csplit* utility because users use both utilities and have historical expectations of their behavior.  
33819 Furthermore, the splitting on byte boundaries in *split* cannot be duplicated with the historical  
33820 *csplit*.33821 The text “*split* shall not delete the files it created with valid suffixes” would normally be  
33822 assumed, but since the related utility, *csplit*, does delete files under some circumstances, the  
33823 historical behavior of *split* is made explicit to avoid misinterpretation.33824 **FUTURE DIRECTIONS**

33825 None.

33826 **SEE ALSO**33827 *csplit*33828 **CHANGE HISTORY**

33829 First released in Issue 2.

33830 **Issue 6**

33831 This utility is now marked as part of the User Portability Utilities option.

33832 The APPLICATION USAGE section is added.

33833 The obsolescent SYNOPSIS is removed.

33834 **NAME**

33835 strings — find printable strings in files

33836 **SYNOPSIS**33837 UP strings [-a][-t *format*][-n *number*][*file...*]

33838

33839 **DESCRIPTION**

33840 The *strings* utility shall look for printable strings in regular files and shall write those strings to  
 33841 standard output. A printable string is any sequence of four (by default) or more printable  
 33842 characters terminated by a <newline> or NUL character. Additional implementation-defined  
 33843 strings may be written; see *localedef*.

33844 **OPTIONS**

33845 The *strings* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section  
 33846 12.2, Utility Syntax Guidelines.

33847 The following options shall be supported:

33848 **-a** Scan files in their entirety. If **-a** is not specified, it is implementation-defined what  
 33849 portion of each file is scanned for strings.

33850 **-n *number*** Specify the minimum string length, where the *number* argument is a positive  
 33851 decimal integer. The default shall be 4.

33852 **-t *format*** Write each string preceded by its byte offset from the start of the file. The format  
 33853 shall be dependent on the single character used as the *format* option-argument:

33854     d The offset shall be written in decimal.

33855     o The offset shall be written in octal.

33856     x The offset shall be written in hexadecimal.

33857 **OPERANDS**

33858 The following operand shall be supported:

33859 ***file*** A pathname of a regular file to be used as input. If no *file* operand is specified, the  
 33860 *strings* utility shall read from the standard input.

33861 **STDIN**

33862 See the INPUT FILES section.

33863 **INPUT FILES**

33864 The input files named by the utility arguments or the standard input shall be regular files of any  
 33865 format.

33866 **ENVIRONMENT VARIABLES**

33867 The following environment variables shall affect the execution of *strings*:

33868 ***LANG*** Provide a default value for the internationalization variables that are unset or null.  
 33869 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
 33870 Internationalization Variables for the precedence of internationalization variables  
 33871 used to determine the values of locale categories.)

33872 ***LC\_ALL*** If set to a non-empty string value, override the values of all the other  
 33873 internationalization variables.

33874 ***LC\_CTYPE*** Determine the locale for the interpretation of sequences of bytes of text data as  
 33875 characters (for example, single-byte as opposed to multi-byte characters in  
 33876 arguments and input files) and to identify printable strings.

- 33877 **LC\_MESSAGES**  
33878 Determine the locale that should be used to affect the format and contents of  
33879 diagnostic messages written to standard error.
- 33880 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 33881 **ASYNCHRONOUS EVENTS**  
33882 Default.
- 33883 **STDOUT**  
33884 Strings found shall be written to the standard output, one per line.  
33885 When the **-t** option is not specified, the format of the output shall be:  
33886 "%s", <string>  
33887 With the **-t o** option, the format of the output shall be:  
33888 "%o %s", <byte offset>, <string>  
33889 With the **-t x** option, the format of the output shall be:  
33890 "%x %s", <byte offset>, <string>  
33891 With the **-t d** option, the format of the output shall be:  
33892 "%d %s", <byte offset>, <string>
- 33893 **STDERR**  
33894 The standard error shall be used only for diagnostic messages.
- 33895 **OUTPUT FILES**  
33896 None.
- 33897 **EXTENDED DESCRIPTION**  
33898 None.
- 33899 **EXIT STATUS**  
33900 The following exit values shall be returned:  
33901 0 Successful completion.  
33902 >0 An error occurred.
- 33903 **CONSEQUENCES OF ERRORS**  
33904 Default.
- 33905 **APPLICATION USAGE**  
33906 By default the data area (as opposed to the text, “bss” or header areas) of a binary executable file  
33907 is scanned. Implementations document which areas are scanned.  
33908 Some historical implementations do not require NUL or <newline> terminators for strings to  
33909 permit those languages that do not use NUL as a string terminator to have their strings written.
- 33910 **EXAMPLES**  
33911 None.
- 33912 **RATIONALE**  
33913 Apart from rationalizing the option syntax and slight difficulties with object and executable  
33914 binary files, *strings* is specified to match historical practice closely. The **-a** and **-n** options were  
33915 introduced to replace the non-conforming **-** and **-number** options.  
33916 The **-o** option historically means different things on different implementations. Some use it to  
33917 mean “*offset* in decimal”, while others use it as “*offset* in octal”. Instead of trying to decide which

- 33918 way would be least objectionable, the `-t` option was added. It was originally named `-O` to mean  
33919 “offset”, but was changed to `-t` to be consistent with `od`.
- 33920 The ISO C standard function `isprint()` is restricted to a domain of **unsigned char**. This volume of  
33921 IEEE Std 1003.1-200x requires implementations to write strings as defined by the current locale.
- 33922 **FUTURE DIRECTIONS**
- 33923 None.
- 33924 **SEE ALSO**
- 33925 *nm*
- 33926 **CHANGE HISTORY**
- 33927 First released in Issue 4.
- 33928 **Issue 6**
- 33929 This utility is now marked as part of the User Portability Utilities option.
- 33930 The obsolescent SYNOPSIS is removed.
- 33931 The normative text is reworded to avoid use of the term “must” for application requirements.

## 33932 NAME

33933 strip — remove unnecessary information from executable files (DEVELOPMENT)

## 33934 SYNOPSIS

33935 SD strip file...

33936

## 33937 DESCRIPTION

33938 The *strip* utility shall remove from executable files named by the *file* operands any information  
 33939 the implementor deems unnecessary for execution of those files. The nature of that information  
 33940 is unspecified. The effect of *strip* shall be similar to the use of the *-s* option to *c99* or *fort77*.

## 33941 OPTIONS

33942 None.

## 33943 OPERANDS

33944 The following operand shall be supported:

33945 *file* A pathname referring to an executable file.

## 33946 STDIN

33947 Not used.

## 33948 INPUT FILES

33949 The input files shall be in the form of executable files successfully produced by any compiler  
 33950 defined by this volume of IEEE Std 1003.1-200x.

## 33951 ENVIRONMENT VARIABLES

33952 The following environment variables shall affect the execution of *strip*:

33953 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 33954 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
 33955 Internationalization Variables for the precedence of internationalization variables  
 33956 used to determine the values of locale categories.)

33957 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 33958 internationalization variables.

33959 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 33960 characters (for example, single-byte as opposed to multi-byte characters in  
 33961 arguments).

33962 *LC\_MESSAGES*

33963 Determine the locale that should be used to affect the format and contents of  
 33964 diagnostic messages written to standard error.

33965 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

## 33966 ASYNCHRONOUS EVENTS

33967 Default.

## 33968 STDOUT

33969 Not used.

## 33970 STDERR

33971 The standard error shall be used only for diagnostic messages. |

33972 **OUTPUT FILES**

33973           The *strip* utility shall produce executable files of unspecified format.

33974 **EXTENDED DESCRIPTION**

33975           None.

33976 **EXIT STATUS**

33977           The following exit values shall be returned:

33978           0   Successful completion.

33979           >0  An error occurred.

33980 **CONSEQUENCES OF ERRORS**

33981           Default.

33982 **APPLICATION USAGE**

33983           None.

33984 **EXAMPLES**

33985           None.

33986 **RATIONALE**

33987           Historically, this utility has been used to remove the symbol table from an executable file. It was included since it is known that the amount of symbolic information can amount to several megabytes; the ability to remove it in a portable manner was deemed important, especially for smaller systems.

33991           The behavior of *strip* is said to be the same as the `-s` option to a compiler. While the end result is essentially the same, it is not required to be identical.

33993 **FUTURE DIRECTIONS**

33994           None.

33995 **SEE ALSO**

33996           *ar*, *c99*, *fort77*

33997 **CHANGE HISTORY**

33998           First released in Issue 2.

33999 **Issue 6**

34000           This utility is now marked as part of the Software Development Utilities option.

## 34001 NAME

34002 stty — set the options for a terminal

## 34003 SYNOPSIS

34004 stty [ -a | -g ]

34005 stty *operands*

## 34006 DESCRIPTION

34007 The *stty* utility shall set or report on terminal I/O characteristics for the device that is its  
 34008 standard input. Without options or operands specified, it shall report the settings of certain  
 34009 characteristics, usually those that differ from implementation-defined defaults. Otherwise, it  
 34010 shall modify the terminal state according to the specified operands. Detailed information about  
 34011 the modes listed in the first five groups below are described in the Base Definitions volume of  
 34012 IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface. Operands in the Combination  
 34013 Modes group (see **Combination Modes** (on page 3087)) are implemented using operands in the  
 34014 previous groups. Some combinations of operands are mutually-exclusive on some terminal  
 34015 types; the results of using such combinations are unspecified.

34016 Typical implementations of this utility require a communications line configured to use the  
 34017 **termios** interface defined in the System Interfaces volume of IEEE Std 1003.1-200x. On systems  
 34018 where none of these lines are available, and on lines not currently configured to support the  
 34019 **termios** interface, some of the operands need not affect terminal characteristics.

## 34020 OPTIONS

34021 The *stty* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section  
 34022 12.2, Utility Syntax Guidelines.

34023 The following options shall be supported:

- 34024 **-a** Write to standard output all the current settings for the terminal.
- 34025 **-g** Write to standard output all the current settings in an unspecified form that can be  
 34026 used as arguments to another invocation of the *stty* utility on the same system. The  
 34027 form used shall not contain any characters that would require quoting to avoid  
 34028 word expansion by the shell; see Section 2.6 (on page 2238).

## 34029 OPERANDS

34030 The following operands shall be supported to set the terminal characteristics.

## 34031 Control Modes

- 34032 **parenb** (**-parenb**) Enable (disable) parity generation and detection. This shall have the effect of |  
 34033 setting (not setting) PARENB in the **termios** *c\_flag* field, as defined in the |  
 34034 Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General  
 34035 Terminal Interface.
- 34036 **parodd** (**-parodd**) Select odd (even) parity. This shall have the effect of setting (not setting)  
 34037 PARODD in the **termios** *c\_flag* field, as defined in the Base Definitions  
 34038 volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface.
- 34039 **cs5 cs6 cs7 cs8** Select character size, if possible. This shall have the effect of setting CS5, CS6,  
 34040 CS7, and CS8, respectively, in the **termios** *c\_flag* field, as defined in the Base  
 34041 Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal  
 34042 Interface.
- 34043 *number* Set terminal baud rate to the number given, if possible. If the baud rate is set  
 34044 to zero, the modem control lines shall not be longer asserted. This shall have  
 34045 the effect of setting the input and output **termios** baud rate values as defined

|       |                                  |                                                                                                        |
|-------|----------------------------------|--------------------------------------------------------------------------------------------------------|
| 34046 |                                  | in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General                            |
| 34047 |                                  | Terminal Interface.                                                                                    |
| 34048 | <b>ispeed</b> <i>number</i>      | Set terminal input baud rate to the number given, if possible. If the input baud                       |
| 34049 |                                  | rate is set to zero, the input baud rate shall be specified by the value of the                        |
| 34050 |                                  | output baud rate. This shall have the effect of setting the input <b>termios</b> baud                  |
| 34051 |                                  | rate values as defined in the Base Definitions volume of IEEE Std 1003.1-200x,                         |
| 34052 |                                  | Chapter 11, General Terminal Interface.                                                                |
| 34053 | <b>ospeed</b> <i>number</i>      | Set terminal output baud rate to the number given, if possible. If the output                          |
| 34054 |                                  | baud rate is set to zero, the modem control lines shall no longer be asserted.                         |
| 34055 |                                  | This shall have the effect of setting the output <b>termios</b> baud rate values as                    |
| 34056 |                                  | defined in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11,                            |
| 34057 |                                  | General Terminal Interface.                                                                            |
| 34058 | <b>hupcl</b> ( <b>-hupcl</b> )   | Stop asserting modem control lines (do not stop asserting modem control                                |
| 34059 |                                  | lines) on last close. This shall have the effect of setting (not setting) HUPCL in                     |
| 34060 |                                  | the <b>termios</b> <i>c_cflag</i> field, as defined in the Base Definitions volume of                  |
| 34061 |                                  | IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface.                                          |
| 34062 | <b>hup</b> ( <b>-hup</b> )       | Equivalent to <b>hupcl</b> ( <b>-hupcl</b> ).                                                          |
| 34063 | <b>cstopb</b> ( <b>-cstopb</b> ) | Use two (one) stop bits per character. This shall have the effect of setting (not                      |
| 34064 |                                  | setting) CSTOPB in the <b>termios</b> <i>c_cflag</i> field, as defined in the Base Definitions         |
| 34065 |                                  | volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface.                                |
| 34066 | <b>cread</b> ( <b>-cread</b> )   | Enable (disable) the receiver. This shall have the effect of setting (not setting)                     |
| 34067 |                                  | CREAD in the <b>termios</b> <i>c_cflag</i> field, as defined in the Base Definitions volume            |
| 34068 |                                  | of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface.                                       |
| 34069 | <b>clocal</b> ( <b>-clocal</b> ) | Assume a line without (with) modem control. This shall have the effect of                              |
| 34070 |                                  | setting (not setting) CLOCAL in the <b>termios</b> <i>c_cflag</i> field, as defined in the             |
| 34071 |                                  | Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General                                   |
| 34072 |                                  | Terminal Interface.                                                                                    |
| 34073 |                                  | It is unspecified whether <i>stty</i> shall report an error if an attempt to set a Control Mode fails. |
| 34074 | <b>Input Modes</b>               |                                                                                                        |
| 34075 | <b>ignbrk</b> ( <b>-ignbrk</b> ) | Ignore (do not ignore) break on input. This shall have the effect of setting (not                      |
| 34076 |                                  | setting) IGNBRK in the <b>termios</b> <i>c_iflag</i> field, as defined in the Base Definitions         |
| 34077 |                                  | volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface.                                |
| 34078 | <b>brkint</b> ( <b>-brkint</b> ) | Signal (do not signal) INTR on break. This shall have the effect of setting (not                       |
| 34079 |                                  | setting) BRKINT in the <b>termios</b> <i>c_iflag</i> field, as defined in the Base Definitions         |
| 34080 |                                  | volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface.                                |
| 34081 | <b>ignpar</b> ( <b>-ignpar</b> ) | Ignore (do not ignore) bytes with parity errors. This shall have the effect of                         |
| 34082 |                                  | setting (not setting) IGNPAR in the <b>termios</b> <i>c_iflag</i> field, as defined in the Base        |
| 34083 |                                  | Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal                               |
| 34084 |                                  | Interface.                                                                                             |
| 34085 | <b>parmrk</b> ( <b>-parmrk</b> ) |                                                                                                        |
| 34086 |                                  | Mark (do not mark) parity errors. This shall have the effect of setting (not                           |
| 34087 |                                  | setting) PARMRK in the <b>termios</b> <i>c_iflag</i> field, as defined in the Base                     |
| 34088 |                                  | Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal                               |
| 34089 |                                  | Interface.                                                                                             |

|           |                                  |                                                                                                                                                                                                                                                                                                                                                                         |
|-----------|----------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 34090     | <b>inpck</b> ( <b>-inpck</b> )   | Enable (disable) input parity checking. This shall have the effect of setting (not setting) INPCK in the <b>termios</b> <i>c_iflag</i> field, as defined in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface.                                                                                                                |
| 34091     |                                  |                                                                                                                                                                                                                                                                                                                                                                         |
| 34092     |                                  |                                                                                                                                                                                                                                                                                                                                                                         |
| 34093     | <b>istrip</b> ( <b>-istrip</b> ) | Strip (do not strip) input characters to seven bits. This shall have the effect of setting (not setting) ISTRIP in the <b>termios</b> <i>c_iflag</i> field, as defined in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface.                                                                                                  |
| 34094     |                                  |                                                                                                                                                                                                                                                                                                                                                                         |
| 34095     |                                  |                                                                                                                                                                                                                                                                                                                                                                         |
| 34096     |                                  |                                                                                                                                                                                                                                                                                                                                                                         |
| 34097     | <b>inlcr</b> ( <b>-inlcr</b> )   | Map (do not map) NL to CR on input. This shall have the effect of setting (not setting) INLCR in the <b>termios</b> <i>c_iflag</i> field, as defined in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface.                                                                                                                    |
| 34098     |                                  |                                                                                                                                                                                                                                                                                                                                                                         |
| 34099     |                                  |                                                                                                                                                                                                                                                                                                                                                                         |
| 34100     | <b>igncr</b> ( <b>-igncr</b> )   | Ignore (do not ignore) CR on input. This shall have the effect of setting (not setting) IGNCR in the <b>termios</b> <i>c_iflag</i> field, as defined in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface.                                                                                                                    |
| 34101     |                                  |                                                                                                                                                                                                                                                                                                                                                                         |
| 34102     |                                  |                                                                                                                                                                                                                                                                                                                                                                         |
| 34103     | <b>icrnl</b> ( <b>-icrnl</b> )   | Map (do not map) CR to NL on input. This shall have the effect of setting (not setting) ICRNL in the <b>termios</b> <i>c_iflag</i> field, as defined in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface.                                                                                                                    |
| 34104     |                                  |                                                                                                                                                                                                                                                                                                                                                                         |
| 34105     |                                  |                                                                                                                                                                                                                                                                                                                                                                         |
| 34106     | <b>ixon</b> ( <b>-ixon</b> )     | Enable (disable) START/STOP output control. Output from the system is stopped when the system receives STOP and started when the system receives START. This shall have the effect of setting (not setting) IXON in the <b>termios</b> <i>c_iflag</i> field, as defined in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface. |
| 34107     |                                  |                                                                                                                                                                                                                                                                                                                                                                         |
| 34108     |                                  |                                                                                                                                                                                                                                                                                                                                                                         |
| 34109     |                                  |                                                                                                                                                                                                                                                                                                                                                                         |
| 34110     |                                  |                                                                                                                                                                                                                                                                                                                                                                         |
| 34111 XSI | <b>ixany</b> ( <b>-ixany</b> )   | Allow any character to restart output. This shall have the effect of setting (not setting) IXANY in the <b>termios</b> <i>c_iflag</i> field, as defined in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface.                                                                                                                 |
| 34112     |                                  |                                                                                                                                                                                                                                                                                                                                                                         |
| 34113     |                                  |                                                                                                                                                                                                                                                                                                                                                                         |
| 34114     | <b>ixoff</b> ( <b>-ixoff</b> )   | Request that the system send (not send) STOP characters when the input queue is nearly full and START characters to resume data transmission. This shall have the effect of setting (not setting) IXOFF in the <b>termios</b> <i>c_iflag</i> field, as defined in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface.          |
| 34115     |                                  |                                                                                                                                                                                                                                                                                                                                                                         |
| 34116     |                                  |                                                                                                                                                                                                                                                                                                                                                                         |
| 34117     |                                  |                                                                                                                                                                                                                                                                                                                                                                         |
| 34118     |                                  |                                                                                                                                                                                                                                                                                                                                                                         |
| 34119     | <b>Output Modes</b>              |                                                                                                                                                                                                                                                                                                                                                                         |
| 34120     | <b>opost</b> ( <b>-opost</b> )   | Post-process output (do not post-process output; ignore all other output modes). This shall have the effect of setting (not setting) OPOST in the <b>termios</b> <i>c_oflag</i> field, as defined in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface.                                                                       |
| 34121     |                                  |                                                                                                                                                                                                                                                                                                                                                                         |
| 34122     |                                  |                                                                                                                                                                                                                                                                                                                                                                         |
| 34123     |                                  |                                                                                                                                                                                                                                                                                                                                                                         |
| 34124 XSI | <b>ocrnl</b> ( <b>-ocrnl</b> )   | Map (do not map) CR to NL on output This shall have the effect of setting (not setting) OCRNL in the <b>termios</b> <i>c_oflag</i> field, as defined in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface.                                                                                                                    |
| 34125     |                                  |                                                                                                                                                                                                                                                                                                                                                                         |
| 34126     |                                  |                                                                                                                                                                                                                                                                                                                                                                         |
| 34127     |                                  |                                                                                                                                                                                                                                                                                                                                                                         |
| 34128     | <b>onocr</b> ( <b>-onocr</b> )   | Do not (do) output CR at column zero. This shall have the effect of setting (not setting) ONOCR in the <b>termios</b> <i>c_oflag</i> field, as defined in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface.                                                                                                                  |
| 34129     |                                  |                                                                                                                                                                                                                                                                                                                                                                         |
| 34130     |                                  |                                                                                                                                                                                                                                                                                                                                                                         |
| 34131     | <b>onlret</b> ( <b>-onlret</b> ) | The terminal newline key performs (does not perform) the CR function. This shall have the effect of setting (not setting) ONLRET in the <b>termios</b> <i>c_oflag</i> field, as defined in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface.                                                                                 |
| 34132     |                                  |                                                                                                                                                                                                                                                                                                                                                                         |
| 34133     |                                  |                                                                                                                                                                                                                                                                                                                                                                         |
| 34134     |                                  |                                                                                                                                                                                                                                                                                                                                                                         |

|       |                                  |                                                                                                                                                                                                                                                                                                                                                               |  |
|-------|----------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|
| 34135 | <b>ofill</b> ( <b>-ofill</b> )   | Use fill characters (use timing) for delays. This shall have the effect of setting (not setting) OFILL in the <b>termios</b> <i>c_oflag</i> field, as defined in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface.                                                                                                 |  |
| 34136 |                                  |                                                                                                                                                                                                                                                                                                                                                               |  |
| 34137 |                                  |                                                                                                                                                                                                                                                                                                                                                               |  |
| 34138 |                                  |                                                                                                                                                                                                                                                                                                                                                               |  |
| 34139 | <b>ofdel</b> ( <b>-ofdel</b> )   | Fill characters are DELs (NULs). This shall have the effect of setting (not setting) OFDEL in the <b>termios</b> <i>c_oflag</i> field, as defined in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface.                                                                                                             |  |
| 34140 |                                  |                                                                                                                                                                                                                                                                                                                                                               |  |
| 34141 |                                  |                                                                                                                                                                                                                                                                                                                                                               |  |
| 34142 | <b>cr0 cr1 cr2 cr3</b>           | Select the style of delay for CRs. This shall have the effect of setting CRDLY to CR0, CR1, CR2, or CR3, respectively, in the <b>termios</b> <i>c_oflag</i> field, as defined in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface.                                                                                 |  |
| 34143 |                                  |                                                                                                                                                                                                                                                                                                                                                               |  |
| 34144 |                                  |                                                                                                                                                                                                                                                                                                                                                               |  |
| 34145 |                                  |                                                                                                                                                                                                                                                                                                                                                               |  |
| 34146 | <b>nl0 nl1</b>                   | Select the style of delay for NL. This has the effect of setting NLDLY to NL0 or NL1, respectively, in the <b>termios</b> <i>c_oflag</i> field, as defined in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface.                                                                                                    |  |
| 34147 |                                  |                                                                                                                                                                                                                                                                                                                                                               |  |
| 34148 |                                  |                                                                                                                                                                                                                                                                                                                                                               |  |
| 34149 |                                  |                                                                                                                                                                                                                                                                                                                                                               |  |
| 34150 | <b>tab0 tab1 tab2 tab3</b>       | Select the style of delay for horizontal tabs. This shall have the effect of setting TABDLY to TAB0, TAB1, TAB2, or TAB3, respectively, in the <b>termios</b> <i>c_oflag</i> field, as defined in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface. Note that TAB3 has the effect of expanding <tab>s to <space>s. |  |
| 34151 |                                  |                                                                                                                                                                                                                                                                                                                                                               |  |
| 34152 |                                  |                                                                                                                                                                                                                                                                                                                                                               |  |
| 34153 |                                  |                                                                                                                                                                                                                                                                                                                                                               |  |
| 34154 |                                  |                                                                                                                                                                                                                                                                                                                                                               |  |
| 34155 |                                  |                                                                                                                                                                                                                                                                                                                                                               |  |
| 34156 | <b>tabs</b> ( <b>-tabs</b> )     | Synonym for <b>tab0</b> ( <b>tab3</b> ).                                                                                                                                                                                                                                                                                                                      |  |
| 34157 |                                  |                                                                                                                                                                                                                                                                                                                                                               |  |
| 34158 | <b>bs0 bs1</b>                   | Select the style of delay for backspaces. This shall have the effect of setting BSDLY to BS0 or BS1, respectively, in the <b>termios</b> <i>c_oflag</i> field, as defined in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface.                                                                                     |  |
| 34159 |                                  |                                                                                                                                                                                                                                                                                                                                                               |  |
| 34160 |                                  |                                                                                                                                                                                                                                                                                                                                                               |  |
| 34161 | <b>ff0 ff1</b>                   | Select the style of delay for form-feeds. This shall have the effect of setting FFDLY to FF0 or FF1, respectively, in the <b>termios</b> <i>c_oflag</i> field, as defined in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface.                                                                                     |  |
| 34162 |                                  |                                                                                                                                                                                                                                                                                                                                                               |  |
| 34163 |                                  |                                                                                                                                                                                                                                                                                                                                                               |  |
| 34164 |                                  |                                                                                                                                                                                                                                                                                                                                                               |  |
| 34165 | <b>vt0 vt1</b>                   | Select the style of delay for vertical-tabs. This shall have the effect of setting VTDLY to VT0 or VT1, respectively, in the <b>termios</b> <i>c_oflag</i> field, as defined in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface.                                                                                  |  |
| 34166 |                                  |                                                                                                                                                                                                                                                                                                                                                               |  |
| 34167 |                                  |                                                                                                                                                                                                                                                                                                                                                               |  |
| 34168 |                                  |                                                                                                                                                                                                                                                                                                                                                               |  |
| 34169 | <b>Local Modes</b>               |                                                                                                                                                                                                                                                                                                                                                               |  |
| 34170 | <b>isig</b> ( <b>-isig</b> )     | Enable (disable) the checking of characters against the special control characters INTR, QUIT, and SUSP. This shall have the effect of setting (not setting) ISIG in the <b>termios</b> <i>c_lflag</i> field, as defined in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface.                                      |  |
| 34171 |                                  |                                                                                                                                                                                                                                                                                                                                                               |  |
| 34172 |                                  |                                                                                                                                                                                                                                                                                                                                                               |  |
| 34173 |                                  |                                                                                                                                                                                                                                                                                                                                                               |  |
| 34174 | <b>icanon</b> ( <b>-icanon</b> ) | Enable (disable) canonical input (ERASE and KILL processing). This shall have the effect of setting (not setting) ICANON in the <b>termios</b> <i>c_lflag</i> field, as defined in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface.                                                                               |  |
| 34175 |                                  |                                                                                                                                                                                                                                                                                                                                                               |  |
| 34176 |                                  |                                                                                                                                                                                                                                                                                                                                                               |  |
| 34177 |                                  |                                                                                                                                                                                                                                                                                                                                                               |  |
| 34178 | <b>ixten</b> ( <b>-ixten</b> )   | Enable (disable) any implementation-defined special control characters not currently controlled by <b>icanon</b> , <b>isig</b> , <b>ixon</b> , or <b>ixoff</b> . This shall have the effect of setting (not setting) IEXTEN in the <b>termios</b> <i>c_lflag</i> field, as defined in the Base                                                                |  |
| 34179 |                                  |                                                                                                                                                                                                                                                                                                                                                               |  |
| 34180 |                                  |                                                                                                                                                                                                                                                                                                                                                               |  |

|       |                                                                                                                                  |                                                                                                  |
|-------|----------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------|
| 34181 |                                                                                                                                  | Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal                         |
| 34182 |                                                                                                                                  | Interface.                                                                                       |
| 34183 | <b>echo</b> ( <del>-echo</del> )                                                                                                 | Echo back (do not echo back) every character typed. This shall have the effect                   |
| 34184 |                                                                                                                                  | of setting (not setting) ECHO in the <b>termios</b> <i>c_lflag</i> field, as defined in the Base |
| 34185 |                                                                                                                                  | Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal                         |
| 34186 |                                                                                                                                  | Interface.                                                                                       |
| 34187 | <b>echoe</b> ( <del>-echoe</del> )                                                                                               | The ERASE character visually erases (does not erase) the last character in the                   |
| 34188 |                                                                                                                                  | current line from the display, if possible. This shall have the effect of setting                |
| 34189 |                                                                                                                                  | (not setting) ECHOE in the <b>termios</b> <i>c_lflag</i> field, as defined in the Base           |
| 34190 |                                                                                                                                  | Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal                         |
| 34191 |                                                                                                                                  | Interface.                                                                                       |
| 34192 | <b>echok</b> ( <del>-echok</del> )                                                                                               | Echo (do not echo) NL after KILL character. This shall have the effect of                        |
| 34193 |                                                                                                                                  | setting (not setting) ECHOK in the <b>termios</b> <i>c_lflag</i> field, as defined in the Base   |
| 34194 |                                                                                                                                  | Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal                         |
| 34195 |                                                                                                                                  | Interface.                                                                                       |
| 34196 | <b>echonl</b> ( <del>-echonl</del> )                                                                                             | Echo (do not echo) NL, even if <b>echo</b> is disabled. This shall have the effect of            |
| 34197 |                                                                                                                                  | setting (not setting) ECHONL in the <b>termios</b> <i>c_lflag</i> field, as defined in the       |
| 34198 |                                                                                                                                  | Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General                             |
| 34199 |                                                                                                                                  | Terminal Interface.                                                                              |
| 34200 | <b>noflsh</b> ( <del>-noflsh</del> )                                                                                             | Disable (enable) flush after INTR, QUIT, SUSP. This shall have the effect of                     |
| 34201 |                                                                                                                                  | setting (not setting) NOFLSH in the <b>termios</b> <i>c_lflag</i> field, as defined in the Base  |
| 34202 |                                                                                                                                  | Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal                         |
| 34203 |                                                                                                                                  | Interface.                                                                                       |
| 34204 | <b>tostop</b> ( <del>-tostop</del> )                                                                                             | Send SIGTTOU for background output. This shall have the effect of setting                        |
| 34205 |                                                                                                                                  | (not setting) TOSTOP in the <b>termios</b> <i>c_lflag</i> field, as defined in the Base          |
| 34206 |                                                                                                                                  | Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal                         |
| 34207 |                                                                                                                                  | Interface.                                                                                       |
| 34208 | <b>Special Control Character Assignments</b>                                                                                     |                                                                                                  |
| 34209 | <i>&lt;control&gt;-character string</i>                                                                                          |                                                                                                  |
| 34210 | Set <i>&lt;control&gt;-character</i> to <i>string</i> . If <i>&lt;control&gt;-character</i> is one of the character sequences in |                                                                                                  |
| 34211 | the first column of the following table, the corresponding Base Definitions volume of                                            |                                                                                                  |
| 34212 | IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface control character from the                                          |                                                                                                  |
| 34213 | second column shall be recognized. This has the effect of setting the corresponding element                                      |                                                                                                  |
| 34214 | of the <b>termios</b> <i>c_cc</i> array (see the Base Definitions volume of IEEE Std 1003.1-200x, Chapter                        |                                                                                                  |
| 34215 | 13, Headers, <b>&lt;termios.h&gt;</b> ).                                                                                         |                                                                                                  |

34216

Table 4-19 Control Character Names in *stty*

34217

34218

34219

34220

34221

34222

34223

34224

34225

34226

| Control Character | c_cc Subscript | Description     |
|-------------------|----------------|-----------------|
| <b>eof</b>        | VEOF           | EOF character   |
| <b>eol</b>        | VEOL           | EOL character   |
| <b>erase</b>      | VERASE         | ERASE character |
| <b>intr</b>       | VINTR          | INTR character  |
| <b>kill</b>       | VKILL          | KILL character  |
| <b>quit</b>       | VQUIT          | QUIT character  |
| <b>susp</b>       | VSUSP          | SUSP character  |
| <b>start</b>      | VSTART         | START character |
| <b>stop</b>       | VSTOP          | STOP character  |

34227

34228

34229

34230

34231

34232

34233

If *string* is a single character, the control character shall be set to that character. If *string* is the two-character sequence "**^\_**" or the string *undef*, the control character shall be set to `_POSIX_VDISABLE`, if it is in effect for the device; if `_POSIX_VDISABLE` is not in effect for the device, it shall be treated as an error. In the POSIX locale, if *string* is a two-character sequence beginning with circumflex (**^**), and the second character is one of those listed in the "**^c**" column of the following table, the control character shall be set to the corresponding character value in the Value column of the table.

34234

Table 4-20 Circumflex Control Characters in *stty*

34235

34236

34237

34238

34239

34240

34241

34242

34243

34244

34245

34246

| <b>^c</b> | Value | <b>^c</b> | Value | <b>^c</b> | Value |
|-----------|-------|-----------|-------|-----------|-------|
| a, A      | <SOH> | l, L      | <FF>  | w, W      | <ETB> |
| b, B      | <STX> | m, M      | <CR>  | x, X      | <CAN> |
| c, C      | <ETX> | n, N      | <SO>  | y, Y      | <EM>  |
| d, D      | <EOT> | o, O      | <SI>  | z, Z      | <SUB> |
| e, E      | <ENQ> | p, P      | <DLE> | [         | <ESC> |
| f, F      | <ACK> | q, Q      | <DC1> | \         | <FS>  |
| g, G      | <BEL> | r, R      | <DC2> | ]         | <GS>  |
| h, H      | <BS>  | s, S      | <DC3> | ^         | <RS>  |
| i, I      | <HT>  | t, T      | <DC4> | _         | <US>  |
| j, J      | <LF>  | u, U      | <NAK> | ?         | <DEL> |
| k, K      | <VT>  | v, V      | <SYN> |           |       |

34247

**min number**

34248

Set the value of MIN to *number*. MIN is used in non-canonical mode input processing (**icanon**).

34249

34250

**time number**

34251

Set the value of TIME to *number*. TIME is used in non-canonical mode input processing (**icanon**).

34252

34253

**Combination Modes**

34254

**saved settings**

34255

Set the current terminal characteristics to the saved settings produced by the **-g** option.

34256

**evenp or parity**

34257

Enable **parenb** and **cs7**; disable **parodd**.

34258

**oddp**

34259

Enable **parenb**, **cs7**, and **parodd**.

- 34260 **-parity, -evenp, or -oddp**  
 34261 Disable **parenb**, and set **cs8**.
- 34262 XSI **raw (-raw or cooked)**  
 34263 Enable (disable) raw input and output. Raw mode shall be equivalent to setting:
- ```
34264 stty cs8 erase ^- kill ^- intr ^- \  

  34265 quit ^- eof ^- eol ^- -post -inpck
```
- 34266 **nl (-nl)**  
 34267 Enable (disable) **icrnl**. In addition, **-nl** unsets **inlcr** and **igncr**.
- 34268 **ek** Reset ERASE and KILL characters back to system defaults.
- 34269 **sane** Reset all modes to some reasonable, unspecified, values.
- 34270 **STDIN**  
 34271 Although no input is read from standard input, standard input shall be used to get the current |  
 34272 terminal I/O characteristics and to set new terminal I/O characteristics. |
- 34273 **INPUT FILES**  
 34274 None.
- 34275 **ENVIRONMENT VARIABLES**  
 34276 The following environment variables shall affect the execution of *stty*:
- 34277 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 34278 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
 34279 Internationalization Variables for the precedence of internationalization variables  
 34280 used to determine the values of locale categories.)
- 34281 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 34282 internationalization variables.
- 34283 **LC\_CTYPE** This variable determines the locale for the interpretation of sequences of bytes of  
 34284 text data as characters (for example, single-byte as opposed to multi-byte  
 34285 characters in arguments) and which characters are in the class **print**.
- 34286 **LC\_MESSAGES**  
 34287 Determine the locale that should be used to affect the format and contents of  
 34288 diagnostic messages written to standard error.
- 34289 XSI **NLSPATH** Determine the location of message catalogs for the processing of **LC\_MESSAGES**.
- 34290 **ASYNCHRONOUS EVENTS**  
 34291 Default.
- 34292 **STDOUT**  
 34293 If operands are specified, no output shall be produced.
- 34294 If the **-g** option is specified, *stty* shall write to standard output the current settings in a form that  
 34295 can be used as arguments to another instance of *stty* on the same system.
- 34296 If the **-a** option is specified, all of the information as described in the OPERANDS section shall  
 34297 be written to standard output. Unless otherwise specified, this information shall be written as  
 34298 <space>-separated tokens in an unspecified format, on one or more lines, with an unspecified  
 34299 number of tokens per line. Additional information may be written.
- 34300 If no options or operands are specified, an unspecified subset of the information written for the  
 34301 **-a** option shall be written.

34302 If speed information is written as part of the default output, or if the `-a` option is specified and if  
 34303 the terminal input speed and output speed are the same, the speed information shall be written  
 34304 as follows:

34305 `"speed %d baud;"`, *<speed>*

34306 Otherwise, speeds shall be written as:

34307 `"ispeed %d baud; ospeed %d baud;"`, *<ispeed>*, *<ospeed>*

34308 In locales other than the POSIX locale, the word **baud** may be changed to something more  
 34309 appropriate in those locales.

34310 If control characters are written as part of the default output, or if the `-a` option is specified,  
 34311 control characters shall be written as:

34312 `"%s = %s;"`, *<control-character name>*, *<value>*

34313 where *<value>* is either the character, or some visual representation of the character if it is non-  
 34314 printable, or the string *undef* if the character is disabled.

#### 34315 **STDERR**

34316 The standard error shall be used only for diagnostic messages.

#### 34317 **OUTPUT FILES**

34318 None.

#### 34319 **EXTENDED DESCRIPTION**

34320 None.

#### 34321 **EXIT STATUS**

34322 The following exit values shall be returned:

34323 0 The terminal options were read or set successfully.

34324 >0 An error occurred.

#### 34325 **CONSEQUENCES OF ERRORS**

34326 Default.

#### 34327 **APPLICATION USAGE**

34328 The `-g` flag is designed to facilitate the saving and restoring of terminal state from the shell level.  
 34329 For example, a program may:

```
34330 saveterm="$ (stty -g)"           # save terminal state
34331 stty (new settings)            # set new state
34332 ...                             # ...
34333 stty $saveterm                 # restore terminal state
```

34334 Since the format is unspecified, the saved value is not portable across systems.

34335 Since the `-a` format is so loosely specified, scripts that save and restore terminal settings should  
 34336 use the `-g` option.

#### 34337 **EXAMPLES**

34338 None.

#### 34339 **RATIONALE**

34340 The original *stty* description was taken directly from System V and reflected the System V  
 34341 terminal driver **termio**. It has been modified to correspond to the terminal driver **termios**.

34342 Output modes are specified only for XSI-conformant systems. All implementations are expected  
 34343 to provide *stty* operands corresponding to all of the output modes they support.

34344 The *stty* utility is primarily used to tailor the user interface of the terminal, such as selecting the  
34345 preferred ERASE and KILL characters. As an application programming utility, *stty* can be used  
34346 within shell scripts to alter the terminal settings for the duration of the script.

34347 The **termios** section states that individual disabling of control characters is possible through the  
34348 option `_POSIX_VDISABLE`. If enabled, two conventions currently exist for specifying this:  
34349 System V uses "`^-`", and BSD uses *undef*. Both are accepted by *stty* in this volume of  
34350 IEEE Std 1003.1-200x. The other BSD convention of using the letter '`u`' was rejected because it  
34351 conflicts with the actual letter '`u`', which is an acceptable value for a control character.

34352 Early proposals did not specify the mapping of "`^c`" to control characters because the control  
34353 characters were not specified in the POSIX locale character set description file requirements. The  
34354 control character set is now specified in the Base Definitions volume of IEEE Std 1003.1-200x,  
34355 Chapter 3, Definitions so the historical mapping is specified. Note that although the mapping  
34356 corresponds to control-character key assignments on many terminals that use the  
34357 ISO/IEC 646:1991 standard (or ASCII) character encodings, the mapping specified here is to the  
34358 control characters, not their keyboard encodings.

34359 Since **termios** supports separate speeds for input and output, two new options were added to  
34360 specify each distinctly.

34361 Some historical implementations use standard input to get and set terminal characteristics;  
34362 others use standard output. Since input from a login TTY is usually restricted to the owner while  
34363 output to a TTY is frequently open to anyone, using standard input provides fewer chances of  
34364 accidentally (or maliciously) altering the terminal settings of other users. Using standard input  
34365 also allows *stty -a* and *stty -g* output to be redirected for later use. Therefore, usage of standard  
34366 input is required by this volume of IEEE Std 1003.1-200x.

#### 34367 **FUTURE DIRECTIONS**

34368 None.

#### 34369 **SEE ALSO**

34370 The Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface

#### 34371 **CHANGE HISTORY**

34372 First released in Issue 2.

#### 34373 **Issue 5**

34374 The description of **tabs** is clarified.

34375 **FUTURE DIRECTIONS** section added.

#### 34376 **Issue 6**

34377 The legacy items **iucl(-iucl)**, **xcase**, **olcuc(-olcuc)**, **lcase(-lcase)**, and **LCASE(-LCASE)**, are  
34378 removed.

## 34379 NAME

34380 tabs — set terminal tabs

## 34381 SYNOPSIS

34382 UP XSI tabs [ -n | -a | -a2 | -c | -c2 | -c3 | -f | -p | -s | -u ][ +m[n] ] [-T type]

34383 tabs [-T type][ +[n] ] n1[,n2,...]

34384

## 34385 DESCRIPTION

34386 The *tabs* utility shall display a series of characters that first clears the hardware terminal tab  
 34387 XSI settings and then initializes the tab stops at the specified positions and optionally adjusts the  
 34388 margin.

34389 The phrase “tab-stop position *N*” shall be taken to mean that, from the start of a line of output,  
 34390 tabbing to position *N* shall cause the next character output to be in the (*N*+1)th column position  
 34391 on that line. The maximum number of tab stops allowed is terminal-dependent.

34392 It need not be possible to implement *tabs* on certain terminals. If the terminal type obtained from  
 34393 the *TERM* environment variable or *-T* option represents such a terminal, an appropriate  
 34394 diagnostic message shall be written to standard error and *tabs* shall exit with a status greater  
 34395 than zero.

## 34396 OPTIONS

34397 The *tabs* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section  
 34398 XSI 12.2, Utility Syntax Guidelines, except for various extensions: the options *-a2*, *-c2*, and *-c3* are  
 34399 multi-character.

34400 The following options shall be supported:

34401 *-n* Specify repetitive tab stops separated by a uniform number of column positions, *n*,  
 34402 where *n* is a single-digit decimal number. The default usage of *tabs* with no  
 34403 arguments shall be equivalent to *tabs-8*. When *-0* is used, the tab stops shall be  
 34404 cleared and no new ones set.

34405 XSI *-a* 1,10,16,36,72  
 34406 Assembler, applicable to some mainframes.

34407 XSI *-a2* 1,10,16,40,72  
 34408 Assembler, applicable to some mainframes.

34409 XSI *-c* 1,8,12,16,20,55  
 34410 COBOL, normal format.

34411 XSI *-c2* 1,6,10,14,49  
 34412 COBOL, compact format (columns 1 to 6 omitted).

34413 XSI *-c3* 1,6,10,14,18,22,26,30,34,38,42,46,50,54,58,62,67  
 34414 COBOL compact format (columns 1 to 6 omitted), with more tabs than *-c2*.

34415 XSI *-f* 1,7,11,15,19,23  
 34416 FORTRAN

34417 XSI *-p* 1,5,9,13,17,21,25,29,33,37,41,45,49,53,57,61  
 34418 PL/1

34419 XSI *-s* 1,10,55  
 34420 SNOBOL

34421 XSI *-u* 1,12,20,44  
 34422 Assembler, applicable to some mainframes.

34423            -T *type*       Indicate the type of terminal. If this option is not supplied and the *TERM* variable  
 34424                                   is unset or null, an unspecified default terminal type shall be used. The setting of  
 34425                                   *type* shall take precedence over the value in *TERM*.

34426 **OPERANDS**

34427            The following operand shall be supported:

34428            *n1[,n2,...]*     A single command line argument that consists of tab-stop values separated using  
 34429                                   either commas or <blank>s. The application shall ensure that the tab-stop values  
 34430                                   are positive decimal integers in strictly ascending order. If any number (except the  
 34431                                   first one) is preceded by a plus sign, it is taken as an increment to be added to the  
 34432                                   previous value. For example, the tab lists 1,10,20,30 and 1,10,+10,+10 are  
 34433                                   considered to be identical.

34434 **STDIN**

34435            Not used.

34436 **INPUT FILES**

34437            None.

34438 **ENVIRONMENT VARIABLES**

34439            The following environment variables shall affect the execution of *tabs*:

34440            *LANG*            Provide a default value for the internationalization variables that are unset or null.  
 34441                                   (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
 34442                                   Internationalization Variables for the precedence of internationalization variables  
 34443                                   used to determine the values of locale categories.)

34444            *LC\_ALL*         If set to a non-empty string value, override the values of all the other  
 34445                                   internationalization variables.

34446            *LC\_CTYPE*     Determine the locale for the interpretation of sequences of bytes of text data as  
 34447                                   characters (for example, single-byte as opposed to multi-byte characters in  
 34448                                   arguments).

34449            *LC\_MESSAGES*

34450                                 Determine the locale that should be used to affect the format and contents of  
 34451                                 diagnostic messages written to standard error.

34452 XSI        *NLSPATH*     Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

34453            *TERM*            Determine the terminal type. If this variable is unset or null, and if the -T option is  
 34454                                   not specified, an unspecified default terminal type shall be used.

34455 **ASYNCHRONOUS EVENTS**

34456            Default.

34457 **STDOUT**

34458            If standard output is a terminal, the appropriate sequence to clear and set the tab stops may be  
 34459                                   written to standard output in an unspecified format. If standard output is not a terminal,  
 34460                                   undefined results occur.

34461 **STDERR**

34462            The standard error shall be used only for diagnostic messages.

34463 **OUTPUT FILES**

34464            None.

34465 **EXTENDED DESCRIPTION**

34466 None.

34467 **EXIT STATUS**

34468 The following exit values shall be returned:

34469 0 Successful completion.

34470 &gt;0 An error occurred.

34471 **CONSEQUENCES OF ERRORS**

34472 Default.

34473 **APPLICATION USAGE**34474 This utility makes use of the terminal's hardware tabs and the *stty tabs* option.

34475 This utility is not recommended for application use.

34476 Some integrated display units might not have escape sequences to set tab stops, but may be set  
34477 by internal system calls. On these terminals, *tabs* works if standard output is directed to the  
34478 terminal; if output is directed to another file, however, *tabs* fails.

34479 **EXAMPLES**

34480 None.

34481 **RATIONALE**

34482 Consideration was given to having the *tput* utility handle all of the functions described in *tabs*.  
34483 However, the separate *tabs* utility was retained because it seems more intuitive to use a  
34484 command named *tabs* than *tput* with a new option. The POSIX Shell and Utilities *tput* does not  
34485 support setting or clearing tabs, and no known historical version of *tabs* supports the capability  
34486 of setting arbitrary tab stops.

34487 The System V *tabs* interface is very complex; the version in this volume of IEEE Std 1003.1-200x  
34488 has a reduced feature list, but many of the features omitted were restored as XSI extensions even  
34489 though the supported languages and coding styles are primarily historical.

34490 There was considerable sentiment for specifying only a means of resetting the tabs back to a  
34491 known state—presumably the “standard” of tabs every eight positions. The following features  
34492 were omitted:

- 34493 • Setting tab stops via the first line in a file, using *--file*. Since even the SVID has no complete  
34494 explanation of this feature, it is doubtful that it is in widespread use.

34495 In an early proposal, a *-t tablist* option was added for consistency with *expand*; this was later  
34496 removed when inconsistencies with the historical list of tabs were identified.

34497 Consideration was given to adding a *-p* option that would output the current tab settings so  
34498 that they could be saved and then later restored. This was not accepted because querying the tab  
34499 stops of the terminal is not a capability in historical *terminfo* or *termcap* facilities and might not be  
34500 supported on a wide range of terminals.

34501 **FUTURE DIRECTIONS**

34502 None.

34503 **SEE ALSO**34504 *expand*, *stty*, *unexpand*

34505 **CHANGE HISTORY**

34506 First released in Issue 2.

34507 **Issue 6**

34508 This utility is now marked as part of the User Portability Utilities option.

34509 The normative text is reworded to avoid use of the term “must” for application requirements.

34510 **NAME**

34511 tail — copy the last part of a file

34512 **SYNOPSIS**34513 tail [-f][ -c *number* | -n *number*][*file*]34514 **DESCRIPTION**34515 The *tail* utility shall copy its input file to the standard output beginning at a designated place.

34516 Copying shall begin at the point in the file indicated by the *-c number* or *-n number* options. The  
 34517 option-argument *number* shall be counted in units of lines or bytes, according to the options *-n*  
 34518 and *-c*. Both line and byte counts start from 1.

34519 Tails relative to the end of the file may be saved in an internal buffer, and thus may be limited in  
 34520 length. Such a buffer, if any, shall be no smaller than {LINE\_MAX}\*10 bytes. |

34521 **OPTIONS**

34522 The *tail* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section  
 34523 12.2, Utility Syntax Guidelines.

34524 The following options shall be supported:

34525 *-c number* The application shall ensure that the *number* option-argument is a decimal integer  
 34526 whose sign affects the location in the file, measured in bytes, to begin the copying:

34527

34528

34529

34530

Sign	Copying Starts
+	Relative to the beginning of the file.
-	Relative to the end of the file.
<i>none</i>	Relative to the end of the file.

34531

34532

The origin for counting shall be 1; that is, *-c +1* represents the first byte of the file,  
*-c -1* the last.

34533 *-f*

34534

34535

34536

34537

34538

If the input file is a regular file or if the *file* operand specifies a FIFO, do not  
 terminate after the last line of the input file has been copied, but read and copy  
 further bytes from the input file when they become available. If no *file* operand is  
 specified and standard input is a pipe, the *-f* option shall be ignored. If the input  
 file is not a FIFO, pipe, or regular file, it is unspecified whether or not the *-f* option  
 shall be ignored.

34539 *-n number*

34540

34541

This option shall be equivalent to *-c number*, except the starting location in the file  
 shall be measured in lines instead of bytes. The origin for counting shall be 1; that  
 is, *-n +1* represents the first line of the file, *-n -1* the last. |

34542 If neither *-c* nor *-n* is specified, *-n 10* shall be assumed.34543 **OPERANDS**

34544 The following operand shall be supported:

34545 *file*

34546

A pathname of an input file. If no *file* operands are specified, the standard input  
 shall be used.

34547 **STDIN**

34548

34549

The standard input shall be used only if no *file* operands are specified. See the INPUT FILES  
 section.

34550 **INPUT FILES**

34551           If the `-c` option is specified, the input file can contain arbitrary data; otherwise, the input file  
34552           shall be a text file.

34553 **ENVIRONMENT VARIABLES**

34554           The following environment variables shall affect the execution of *tail*:

34555           *LANG*       Provide a default value for the internationalization variables that are unset or null.  
34556                       (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
34557                       Internationalization Variables for the precedence of internationalization variables  
34558                       used to determine the values of locale categories.)

34559           *LC\_ALL*     If set to a non-empty string value, override the values of all the other  
34560                       internationalization variables.

34561           *LC\_CTYPE*   Determine the locale for the interpretation of sequences of bytes of text data as  
34562                       characters (for example, single-byte as opposed to multi-byte characters in  
34563                       arguments and input files).

34564           *LC\_MESSAGES*

34565                       Determine the locale that should be used to affect the format and contents of  
34566                       diagnostic messages written to standard error.

34567 *XSI*       *NLSPATH*   Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

34568 **ASYNCHRONOUS EVENTS**

34569           Default.

34570 **STDOUT**

34571           The designated portion of the input file shall be written to standard output.

34572 **STDERR**

34573           The standard error shall be used only for diagnostic messages. |

34574 **OUTPUT FILES**

34575           None.

34576 **EXTENDED DESCRIPTION**

34577           None.

34578 **EXIT STATUS**

34579           The following exit values shall be returned:

34580           0   Successful completion.

34581           >0  An error occurred.

34582 **CONSEQUENCES OF ERRORS**

34583           Default.

34584 **APPLICATION USAGE**

34585           The `-c` option should be used with caution when the input is a text file containing multi-byte  
34586           characters; it may produce output that does not start on a character boundary.

34587           Although the input file to *tail* can be any type, the results might not be what would be expected  
34588           on some character special device files or on file types not described by the System Interfaces  
34589           volume of IEEE Std 1003.1-200x. Since this volume of IEEE Std 1003.1-200x does not specify the  
34590           block size used when doing input, *tail* need not read all of the data from devices that only  
34591           perform block transfers.

34592 **EXAMPLES**

34593       The `-f` option can be used to monitor the growth of a file that is being written by some other  
 34594       process. For example, the command:

34595       `tail -f fred`

34596       prints the last ten lines of the file `fred`, followed by any lines that are appended to `fred` between  
 34597       the time `tail` is initiated and killed. As another example, the command:

34598       `tail -f -c 15 fred`

34599       prints the last 15 bytes of the file `fred`, followed by any bytes that are appended to `fred` between  
 34600       the time `tail` is initiated and killed.

34601 **RATIONALE**

34602       This version of `tail` was created to allow conformance to the Utility Syntax Guidelines. The  
 34603       historical `-b` option was omitted because of the general non-portability of block-sized units of  
 34604       text. The `-c` option historically meant “characters”, but this volume of IEEE Std 1003.1-200x  
 34605       indicates that it means “bytes”. This was selected to allow reasonable implementations when  
 34606       multi-byte characters are possible; it was not named `-b` to avoid confusion with the historical  
 34607       `-b`.

34608       The origin of counting both lines and bytes is 1, matching all widespread historical  
 34609       implementations.

34610       The restriction on the internal buffer is a compromise between the historical System V  
 34611       implementation of 4 096 bytes and the BSD 32 768 bytes.

34612       The `-f` option has been implemented as a loop that sleeps for 1 second and copies any bytes that  
 34613       are available. This is sufficient, but if more efficient methods of determining when new data are  
 34614       available are developed, implementations are encouraged to use them.

34615       Historical documentation indicates that `tail` ignores the `-f` option if the input file is a pipe (pipe  
 34616       and FIFO on systems that support FIFOs). On BSD-based systems, this has been true; on System  
 34617       V-based systems, this was true when input was taken from standard input, but it did not ignore  
 34618       the `-f` flag if a FIFO was named as the `file` operand. Since the `-f` option is not useful on pipes and  
 34619       all historical implementations ignore `-f` if no `file` operand is specified and standard input is a  
 34620       pipe, this volume of IEEE Std 1003.1-200x requires this behavior. However, since the `-f` option is  
 34621       useful on a FIFO, this volume of IEEE Std 1003.1-200x also requires that if standard input is a  
 34622       FIFO or a FIFO is named, the `-f` option shall not be ignored. Although historical behavior does  
 34623       not ignore the `-f` option for other file types, this is unspecified so that implementations are  
 34624       allowed to ignore the `-f` option if it is known that the file cannot be extended.

34625       This was changed to the current form based on comments noting that `-c` was almost never used  
 34626       without specifying a number and that there was no need to specify `-l` if `-n number` was given.

34627 **FUTURE DIRECTIONS**

34628       None.

34629 **SEE ALSO**

34630       `head`

34631 **CHANGE HISTORY**

34632       First released in Issue 2.

34633 **Issue 6**

34634       The obsolescent SYNOPSIS lines and associated text are removed.

34635       The normative text is reworded to avoid use of the term “must” for application requirements.

## 34636 NAME

34637 talk — talk to another user

## 34638 SYNOPSIS

34639 UP `talk address [terminal]`

34640

## 34641 DESCRIPTION

34642 The *talk* utility is a two-way, screen-oriented communication program.34643 When first invoked, *talk* shall send a message similar to:

34644 Message from <unspecified string>  
 34645 talk: connection requested by *your\_address*  
 34646 talk: respond with: talk *your\_address*

34647 to the specified *address*. At this point, the recipient of the message can reply by typing:34648 `talk your_address`34649 Once communication is established, the two parties can type simultaneously, with their output  
 34650 displayed in separate regions of the screen. Characters shall be processed as follows:

- 34651 • Typing the alert character shall alert the recipient's terminal.
- 34652 • Typing <control>-L shall cause the sender's screen regions to be refreshed.
- 34653 • Typing the erase and kill characters shall affect the sender's terminal in the manner described  
 34654 by the **termios** interface in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11,  
 34655 General Terminal Interface.
- 34656 • Typing the interrupt or end-of-file characters shall terminate the local *talk* utility. Once the  
 34657 *talk* session has been terminated on one side, the other side of the *talk* session shall be notified  
 34658 that the *talk* session has been terminated and shall be able to do nothing except exit.
- 34659 • Typing characters from *LC\_CTYPE* classifications **print** or **space** shall cause those characters  
 34660 to be sent to the recipient's terminal.
- 34661 • When and only when the *stty ixten* local mode is enabled, the existence and processing of  
 34662 additional special control characters and multi-byte or single-byte functions shall be  
 34663 implementation-defined.
- 34664 • Typing other non-printable characters shall cause implementation-defined sequences of  
 34665 printable characters to be sent to the recipient's terminal.

34666 Permission to be a recipient of a *talk* message can be denied or granted by use of the *mesg* utility.  
 34667 However, a user's privilege may further constrain the domain of accessibility of other users'  
 34668 terminals. The *talk* utility shall fail when the user lacks the appropriate privileges to perform the  
 34669 requested action.

34670 Certain block-mode terminals do not have all the capabilities necessary to support the  
 34671 simultaneous exchange of messages required for *talk*. When this type of exchange cannot be  
 34672 supported on such terminals, the implementation may support an exchange with reduced levels  
 34673 of simultaneous interaction or it may report an error describing the terminal-related deficiency.

## 34674 OPTIONS

34675 None.

34676 **OPERANDS**

34677 The following operands shall be supported:

34678 *address* The recipient of the *talk* session. One form of *address* is the *<user name>*, as returned  
 34679 by the *who* utility. Other address formats and how they are handled are  
 34680 unspecified.

34681 *terminal* If the recipient is logged in more than once, the *terminal* argument can be used to  
 34682 indicate the appropriate terminal name. If *terminal* is not specified, the *talk* message  
 34683 shall be displayed on one or more accessible terminals in use by the recipient. The  
 34684 format of *terminal* shall be the same as that returned by the *who* utility.

34685 **STDIN**

34686 Characters read from standard input shall be copied to the recipient's terminal in an unspecified  
 34687 manner. If standard input is not a terminal, *talk* shall write a diagnostic message and exit with a  
 34688 non-zero status.

34689 **INPUT FILES**

34690 None.

34691 **ENVIRONMENT VARIABLES**34692 The following environment variables shall affect the execution of *talk*:

34693 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 34694 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
 34695 Internationalization Variables for the precedence of internationalization variables  
 34696 used to determine the values of locale categories.)

34697 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 34698 internationalization variables.

34699 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 34700 characters (for example, single-byte as opposed to multi-byte characters in  
 34701 arguments and input files). If the recipient's locale does not use an *LC\_CTYPE*  
 34702 equivalent to the sender's, the results are undefined.

34703 *LC\_MESSAGES*

34704 Determine the locale that should be used to affect the format and contents of  
 34705 diagnostic messages written to standard error and informative messages written to  
 34706 standard output.

34707 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

34708 *TERM* Determine the name of the invoker's terminal type. If this variable is unset or null,  
 34709 an unspecified default terminal type shall be used.

34710 **ASYNCHRONOUS EVENTS**

34711 When the *talk* utility receives a SIGINT signal, the utility shall terminate and exit with a zero  
 34712 status. It shall take the standard action for all other signals.

34713 **STDOUT**

34714 If standard output is a terminal, characters copied from the recipient's standard input may be  
 34715 written to standard output. Standard output also may be used for diagnostic messages. If  
 34716 standard output is not a terminal, *talk* shall exit with a non-zero status.

34717 **STDERR**

34718 None.

34719 **OUTPUT FILES**

34720 None.

34721 **EXTENDED DESCRIPTION**

34722 None.

34723 **EXIT STATUS**

34724 The following exit values shall be returned:

34725 0 Successful completion.

34726 >0 An error occurred or *talk* was invoked on a terminal incapable of supporting it.34727 **CONSEQUENCES OF ERRORS**

34728 Default.

34729 **APPLICATION USAGE**

34730 Because the handling of non-printable, non-`<space>`s is tied to the *stty* description of **ixten**,  
34731 implementation extensions within the terminal driver can be accessed. For example, some  
34732 implementations provide line editing functions with certain control character sequences.

34733 **EXAMPLES**

34734 None.

34735 **RATIONALE**

34736 The *write* utility was included in this volume of IEEE Std 1003.1-200x since it can be  
34737 implemented on all terminal types. The *talk* utility, which cannot be implemented on certain  
34738 terminals, was considered to be a “better” communications interface. Both of these programs are  
34739 in widespread use on historical implementations. Therefore, both utilities have been specified.

34740 All references to networking abilities (*talking* to a user on another system) were removed as  
34741 being outside the scope of this volume of IEEE Std 1003.1-200x.

34742 Historical BSD and System V versions of *talk* terminate both of the conversations when either  
34743 user breaks out of the session. This can lead to adverse consequences if a user unwittingly  
34744 continues to enter text that is interpreted by the shell when the other terminates the session.  
34745 Therefore, the version of *talk* specified by this volume of IEEE Std 1003.1-200x requires both  
34746 users to terminate their end of the session explicitly.

34747 Only messages sent to the terminal of the invoking user can be internationalized in any way:

34748 • The original “Message from *<unspecified string>* ...” message sent to the terminal of the  
34749 recipient cannot be internationalized because the environment of the recipient is as yet  
34750 inaccessible to the *talk* utility. The environment of the invoking party is irrelevant.

34751 • Subsequent communication between the two parties cannot be internationalized because the  
34752 two parties may specify different languages in their environment (and non-portable  
34753 characters cannot be mapped from one language to another).

34754 • Neither party can be required to communicate in a language other than C and/or the one  
34755 specified by their environment because unavailable terminal hardware support (for example,  
34756 fonts) may be required.

34757 The text in the STDOUT section reflects the usage of the verb “display” in this section; some *talk*  
34758 implementations actually use standard output to write to the terminal, but this volume of  
34759 IEEE Std 1003.1-200x does not require that to be the case.

34760 The format of the terminal name is unspecified, but the descriptions of *ps*, *talk*, *who*, and *write*  
34761 require that they all use or accept the same format.

34762           The handling of non-printable characters is partially implementation-defined because the details  
34763           of mapping them to printable sequences is not needed by the user. Historical implementations,  
34764           for security reasons, disallow the transmission of non-printable characters that may send  
34765           commands to the other terminal.

34766 **FUTURE DIRECTIONS**

34767           None.

34768 **SEE ALSO**

34769           *msg*, *who*, *write*, the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General  
34770           Terminal Interface

34771 **CHANGE HISTORY**

34772           First released in Issue 4.

34773 **Issue 6**

34774           This utility is now marked as part of the User Portability Utilities option.

34775 **NAME**

34776 tee — duplicate standard input

34777 **SYNOPSIS**

34778 tee [-ai][file...]

34779 **DESCRIPTION**

34780 The *tee* utility shall copy standard input to standard output, making a copy in zero or more files.

34781 The *tee* utility shall not buffer output.

34782 If the **-a** option is not specified, output files shall be written (see Section 1.7.1.4 (on page 2204)).

34783 **OPTIONS**

34784 The *tee* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2,  
34785 Utility Syntax Guidelines.

34786 The following options shall be supported:

34787 **-a** Append the output to the files.

34788 **-i** Ignore the SIGINT signal.

34789 **OPERANDS**

34790 The following operands shall be supported:

34791 *file* A pathname of an output file. Processing of at least 13 *file* operands shall be  
34792 supported.

34793 **STDIN**

34794 The standard input can be of any type.

34795 **INPUT FILES**

34796 None.

34797 **ENVIRONMENT VARIABLES**

34798 The following environment variables shall affect the execution of *tee*:

34799 *LANG* Provide a default value for the internationalization variables that are unset or null.  
34800 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
34801 Internationalization Variables for the precedence of internationalization variables  
34802 used to determine the values of locale categories.)

34803 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
34804 internationalization variables.

34805 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
34806 characters (for example, single-byte as opposed to multi-byte characters in  
34807 arguments).

34808 *LC\_MESSAGES*

34809 Determine the locale that should be used to affect the format and contents of  
34810 diagnostic messages written to standard error.

34811 *XSI* *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

34812 **ASYNCHRONOUS EVENTS**

34813 Default, except that if the **-i** option was specified, SIGINT shall be ignored.

34814 **STDOUT**

34815 The standard output shall be a copy of the standard input.

34816 **STDERR**

34817 The standard error shall be used only for diagnostic messages.

34818 **OUTPUT FILES**

34819 If any *file* operands are specified, the standard input shall be copied to each named file.

34820 **EXTENDED DESCRIPTION**

34821 None.

34822 **EXIT STATUS**

34823 The following exit values shall be returned:

34824 0 The standard input was successfully copied to all output files.

34825 >0 An error occurred.

34826 **CONSEQUENCES OF ERRORS**

34827 If a write to any successfully opened *file* operand fails, writes to other successfully opened *file* operands and standard output shall continue, but the exit status shall be non-zero. Otherwise, the default actions specified in Section 1.11 (on page 2221) apply.

34830 **APPLICATION USAGE**

34831 The *tee* utility is usually used in a pipeline, to make a copy of the output of some utility.

34832 The *file* operand is technically optional, but *tee* is no more useful than *cat* when none is specified.

34833 **EXAMPLES**

34834 Save an unsorted intermediate form of the data in a pipeline:

34835 ... | tee unsorted | sort > sorted

34836 **RATIONALE**

34837 The buffering requirement means that *tee* is not allowed to use ISO C standard fully buffered or line-buffered writes. It does not mean that *tee* has to do 1-byte reads followed by 1-byte writes.

34839 It should be noted that early versions of BSD ignore any invalid options and accept a single '-' as an alternative to -i. They also print a message if unable to open a file:

34841 "tee: cannot access %s\n", <pathname>

34842 Historical implementations ignore write errors. This is explicitly not permitted by this volume of IEEE Std 1003.1-200x.

34844 Some historical implementations use O\_APPEND when providing append mode; others use the *lseek()* function to seek to the end of file after opening the file without O\_APPEND. This volume of IEEE Std 1003.1-200x requires functionality equivalent to using O\_APPEND; see Section 1.7.1.4 (on page 2204).

34848 **FUTURE DIRECTIONS**

34849 None.

34850 **SEE ALSO**

34851 *cat*

34852 **CHANGE HISTORY**

34853 First released in Issue 2.

34854 **Issue 6**

34855 IEEE PASC Interpretation 1003.2 #168 is applied.

34856 **NAME**

34857           test — evaluate expression

34858 **SYNOPSIS**34859           test [*expression*]34860           [ [*expression*] ]34861 **DESCRIPTION**

34862           The *test* utility shall evaluate the *expression* and indicates the result of the evaluation by its exit  
 34863           status. An exit status of zero indicates that the expression evaluated as true and an exit status of  
 34864           1 indicates that the expression evaluated as false.

34865           In the second form of the utility, which uses "[ ]" rather than *test*, the application shall ensure  
 34866           that the square brackets are separate arguments.

34867 **OPTIONS**

34868           The *test* utility shall not recognize the "--" argument in the manner specified by guideline 10 in  
 34869           the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2, Utility Syntax Guidelines.

34870           No options shall be supported.

34871 **OPERANDS**

34872           The application shall ensure that all operators and elements of primaries are presented as  
 34873           separate arguments to the *test* utility.

34874           The following primaries can be used to construct *expression*:

34875           **-b file**       True if *file* exists and is a block special file.

34876           **-c file**       True if *file* exists and is a character special file.

34877           **-d file**       True if *file* exists and is a directory.

34878           **-e file**       True if *file* exists.

34879           **-f file**       True if *file* exists and is a regular file.

34880           **-g file**       True if *file* exists and its set-group-ID flag is set.

34881           **-h file**       True if *file* exists and is a symbolic link.

34882           **-L file**       True if *file* exists and is a symbolic link.

34883           **-n string**    True if the length of *string* is non-zero.

34884           **-p file**       True if *file* is a FIFO.

34885           **-r file**       True if *file* exists and is readable. True shall indicate that permission to read from  
 34886           *file* will be granted, as defined in Section 1.7.1.4 (on page 2204).

34887           **-S file**       True if *file* exists and is a socket.

34888           **-s file**       True if *file* exists and has a size greater than zero.

34889           **-t file\_descriptor**

34890           True if the file whose file descriptor number is *file\_descriptor* is open and is  
 34891           associated with a terminal.

34892           **-u file**       True if *file* exists and its set-user-ID flag is set.

34893           **-w file**       True if *file* exists and is writable. True shall indicate that permission to write from  
 34894           *file* will be granted, as defined in Section 1.7.1.4 (on page 2204).

34895	<b>-x</b> <i>file</i>	True if <i>file</i> exists and is executable. True shall indicate that permission to execute <i>file</i> will be granted, as defined in Section 1.7.1.4 (on page 2204). If <i>file</i> is a directory, true shall indicate that permission to search <i>file</i> will be granted.
34896		
34897		
34898	<b>-z</b> <i>string</i>	True if the length of string <i>string</i> is zero.
34899	<i>string</i>	True if the string <i>string</i> is not the null string.
34900	<i>s1</i> = <i>s2</i>	True if the strings <i>s1</i> and <i>s2</i> are identical.
34901	<i>s1</i> != <i>s2</i>	True if the strings <i>s1</i> and <i>s2</i> are not identical.
34902	<i>n1</i> <b>-eq</b> <i>n2</i>	True if the integers <i>n1</i> and <i>n2</i> are algebraically equal.
34903	<i>n1</i> <b>-ne</b> <i>n2</i>	True if the integers <i>n1</i> and <i>n2</i> are not algebraically equal.
34904	<i>n1</i> <b>-gt</b> <i>n2</i>	True if the integer <i>n1</i> is algebraically greater than the integer <i>n2</i> .
34905	<i>n1</i> <b>-ge</b> <i>n2</i>	True if the integer <i>n1</i> is algebraically greater than or equal to the integer <i>n2</i> .
34906	<i>n1</i> <b>-lt</b> <i>n2</i>	True if the integer <i>n1</i> is algebraically less than the integer <i>n2</i> .
34907	<i>n1</i> <b>-le</b> <i>n2</i>	True if the integer <i>n1</i> is algebraically less than or equal to the integer <i>n2</i> .
34908 XSI	<i>expression1</i> <b>-a</b> <i>expression2</i>	True if both <i>expression1</i> and <i>expression2</i> are true. The <b>-a</b> binary primary is left associative. It has a higher precedence than <b>-o</b> .
34909		
34910		
34911 XSI	<i>expression1</i> <b>-o</b> <i>expression2</i>	True if either <i>expression1</i> or <i>expression2</i> is true. The <b>-o</b> binary primary is left associative.
34912		
34913		
34914		With the exception of the <b>-h</b> <i>file</i> primary, if a <i>file</i> argument is a symbolic link, <i>test</i> shall evaluate the expression by resolving the symbolic link and using the file referenced by the link.
34915		
34916		These primaries can be combined with the following operators:
34917	<b>!</b> <i>expression</i>	True if <i>expression</i> is false.
34918 XSI	<b>(</b> <i>expression</i> <b>)</b>	True if <i>expression</i> is true. The parentheses can be used to alter the normal precedence and associativity.
34919		
34920		The primaries with two elements of the form:
34921	<b>-</b> <i>primary_operator</i> <i>primary_operand</i>	
34922		are known as <i>unary primaries</i> . The primaries with three elements in either of the two forms:
34923	<i>primary_operand</i> <b>-</b> <i>primary_operator</i> <i>primary_operand</i>	
34924	<i>primary_operand</i> <b>-</b> <i>primary_operator</i> <i>primary_operand</i>	
34925		are known as <i>binary primaries</i> . Additional implementation-defined operators and <i>primary_operators</i> may be provided by implementations. They shall be of the form <b>-operator</b> where the first character of <i>operator</i> is not a digit.
34926		
34927		
34928		The algorithm for determining the precedence of the operators and the return value that shall be generated is based on the number of arguments presented to <i>test</i> . (However, when using the "[...]" form, the right-bracket final argument shall not be counted in this algorithm.)
34929		
34930		
34931		In the following list, \$1, \$2, \$3, and \$4 represent the arguments presented to <i>test</i> :
34932	0 arguments:	Exit false (1).

- 34933 1 argument: Exit true (0) if \$1 is not null; otherwise, exit false.
- 34934 2 arguments:
  - If \$1 is '!', exit true if \$2 is null, false if \$2 is not null.
  - If \$1 is a unary primary, exit true if the unary test is true, false if the unary test is false.
  - Otherwise, produce unspecified results.
- 34935
- 34936
- 34937
- 34938 3 arguments:
  - If \$2 is a binary primary, perform the binary test of \$1 and \$3.
  - If \$1 is '!', negate the two-argument test of \$2 and \$3.
  - If \$1 is '( ' and \$3 is ') ', perform the unary test of \$2.
  - Otherwise, produce unspecified results.
- 34939
- 34940
- 34941
- 34942 4 arguments:
  - If \$1 is '!', negate the three-argument test of \$2, \$3, and \$4.
  - If \$1 is '( ' and \$4 is ') ', perform the two-argument test of \$2 and \$3.
  - Otherwise, the results are unspecified.
- 34943 XSI
- 34944
- 34945 >4 arguments: The results are unspecified.
- 34946 XSI On XSI-conformant systems, combinations of primaries and operators shall be  
34947 evaluated using the precedence and associativity rules described previously.  
34948 In addition, the string comparison binary primaries '=' and '!=' shall have  
34949 a higher precedence than any unary primary.
- 34950 **STDIN**
- 34951 Not used.
- 34952 **INPUT FILES**
- 34953 None.
- 34954 **ENVIRONMENT VARIABLES**
- 34955 The following environment variables shall affect the execution of *test*:
- 34956 *LANG* Provide a default value for the internationalization variables that are unset or null.  
34957 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
34958 Internationalization Variables for the precedence of internationalization variables  
34959 used to determine the values of locale categories.)
- 34960 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
34961 internationalization variables.
- 34962 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
34963 characters (for example, single-byte as opposed to multi-byte characters in  
34964 arguments).
- 34965 *LC\_MESSAGES*
- 34966 Determine the locale that should be used to affect the format and contents of  
34967 diagnostic messages written to standard error.
- 34968 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 34969 **ASYNCHRONOUS EVENTS**
- 34970 Default.

34971 **STDOUT**

34972 Not used.

34973 **STDERR**

34974 The standard error shall be used only for diagnostic messages. |

34975 **OUTPUT FILES**

34976 None.

34977 **EXTENDED DESCRIPTION**

34978 None.

34979 **EXIT STATUS**

34980 The following exit values shall be returned:

34981 0 *expression* evaluated to true.34982 1 *expression* evaluated to false or *expression* was missing.

34983 &gt;1 An error occurred.

34984 **CONSEQUENCES OF ERRORS**

34985 Default.

34986 **APPLICATION USAGE**34987 Scripts should be careful when dealing with user-supplied input that could be confused with  
34988 primaries and operators. Unless the application writer knows all the cases that produce input to  
34989 the script, invocations like:34990 `test "$1" -a "$2"`

34991 should be written as:

34992 `test "$1" && test "$2"`34993 to avoid problems if a user supplied values such as \$1 set to '!' and \$2 set to the null string.  
34994 That is, in cases where maximal portability is of concern, replace:34995 `test expr1 -a expr2`

34996 with:

34997 `test expr1 && test expr2`

34998 and replace:

34999 `test expr1 -o expr2`

35000 with:

35001 `test expr1 || test expr2`35002 but note that, in *test*, `-a` has higher precedence than `-o` while `&&` and `||` have equal  
35003 precedence in the shell.

35004 Parentheses or braces can be used in the shell command language to effect grouping.

35005 Parentheses must be escaped when using *sh*; for example:35006 `test \( expr1 -a expr2 \) -o expr3`35007 This command is not always portable outside XSI-conformant systems. The following form can  
35008 be used instead:

```
35009      ( test expr1 && test expr2 ) || test expr3
```

35010 The two commands:

```
35011      test "$1"
```

```
35012      test ! "$1"
```

35013 could not be used reliably on some historical systems. Unexpected results would occur if such a *string* expression were used and \$1 expanded to '!', '( ', or a known unary primary. Better constructs are:

```
35016      test -n "$1"
```

```
35017      test -z "$1"
```

35018 respectively.

35019 Historical systems have also been unreliable given the common construct:

```
35020      test "$response" = "expected string"
```

35021 One of the following is a more reliable form:

```
35022      test "X$response" = "Xexpected string"
```

```
35023      test "expected string" = "$response"
```

35024 Note that the second form assumes that *expected string* could not be confused with any unary primary. If *expected string* starts with '-', '( ', '! ', or even '= ', the first form should be used instead. Using the preceding rules without the XSI marked extensions, any of the three comparison forms is reliable, given any input. (However, note that the strings are quoted in all cases.)

35029 Because the string comparison binary primaries, '=' and '!=', have a higher precedence than any unary primary in the greater than 4 argument case, unexpected results can occur if arguments are not properly prepared. For example, in:

```
35032      test -d $1 -o -d $2
```

35033 If \$1 evaluates to a possible directory name of '=', the first three arguments are considered a string comparison, which shall cause a syntax error when the second -d is encountered. One of the following forms prevents this; the second is preferred:

```
35036      test \( -d "$1" \) -o \( -d "$2" \)
```

```
35037      test -d "$1" || test -d "$2"
```

35038 Also in the greater than 4 argument case:

```
35039      test "$1" = "bat" -a "$2" = "ball"
```

35040 Syntax errors occur if \$1 evaluates to '( ' or '! '. One of the following forms prevents this; the third is preferred:

```
35042      test "X$1" = "Xbat" -a "X$2" = "Xball"
```

```
35043      test "$1" = "bat" && test "$2" = "ball"
```

```
35044      test "X$1" = "Xbat" && test "X$2" = "Xball"
```

#### 35045 EXAMPLES

35046 1. Exit if there are not two or three arguments (two variations):

```
35047      if [ $# -ne 2 -a $# -ne 3 ]; then exit 1; fi
```

```
35048      if [ $# -lt 2 -o $# -gt 3 ]; then exit 1; fi
```

```

35049      2. Perform a mkdir if a directory does not exist:
35050          test ! -d tempdir && mkdir tempdir
35051      3. Wait for a file to become non-readable:
35052          while test -r thefile
35053          do
35054              sleep 30
35055          done
35056          echo '"thefile" is no longer readable'
35057      4. Perform a command if the argument is one of three strings (two variations):
35058          if [ "$1" = "pear" ] || [ "$1" = "grape" ] || [ "$1" = "apple" ]
35059          then
35060              command
35061          fi
35062          case "$1" in
35063              pear|grape|apple) command ;;
35064          esac

```

#### 35065 RATIONALE

35066 The KornShell-derived conditional command (double bracket `[[ ]]`) was removed from the shell  
 35067 command language description in an early proposal. Objections were raised that the real  
 35068 problem is misuse of the *test* command (`()`), and putting it into the shell is the wrong way to fix  
 35069 the problem. Instead, proper documentation and a new shell reserved word (`!`) are sufficient.

35070 Tests that require multiple *test* operations can be done at the shell level using individual  
 35071 invocations of the *test* command and shell logicals, rather than using the error-prone `-o` flag of  
 35072 *test*.

35073 XSI-conformant systems support more than four arguments.

35074 XSI-conformant systems support the combining of primaries with the following constructs:

35075 *expression1* **-a** *expression2*

35076 True if both *expression1* and *expression2* are true.

35077 *expression1* **-o** *expression2*

35078 True if at least one of *expression1* and *expression2* are true.

35079 (*expression*)

35080 True if *expression* is true.

35081 In evaluating these more complex combined expressions, the following precedence rules are  
 35082 used:

- 35083 • The unary primaries have higher precedence than the algebraic binary primaries.
- 35084 • The unary primaries have lower precedence than the string binary primaries.
- 35085 • The unary and binary primaries have higher precedence than the unary *string* primary.
- 35086 • The `!` operator has higher precedence than the `-a` operator, and the `-a` operator has higher  
 35087 precedence than the `-o` operator.
- 35088 • The `-a` and `-o` operators are left associative.
- 35089 • The parentheses can be used to alter the normal precedence and associativity.

35090 The BSD and System V versions of `-f` are not the same. The BSD definition was:

35091 `-f file` True if *file* exists and is not a directory.

35092 The SVID version (true if the file exists and is a regular file) was chosen for this volume of  
35093 IEEE Std 1003.1-200x because its use is consistent with the `-b`, `-c`, `-d`, and `-p` operands (*file* exists  
35094 and is a specific file type).

35095 The `-e` primary, possessing similar functionality to that provided by the C shell, was added  
35096 because it provides the only way for a shell script to find out if a file exists without trying to  
35097 open the file. Since implementations are allowed to add additional file types, a portable script  
35098 cannot use:

35099 `test -b foo -o -c foo -o -d foo -o -f foo -o -p foo`

35100 to find out if `foo` is an existing file.) On historical BSD systems, the existence of a file could be  
35101 determined by:

35102 `test -f foo -o -d foo`

35103 but there was no easy way to determine that an existing file was a regular file. An early proposal  
35104 used the KornShell `-a` primary (with the same meaning), but this was changed to `-e` because  
35105 there were concerns about the high probability of humans confusing the `-a` primary with the `-a`  
35106 binary operator.

35107 The following options were not included in this volume of IEEE Std 1003.1-200x, although they  
35108 are provided by some implementations. These operands should not be used by new  
35109 implementations for other purposes:

35110 `-k file` True if *file* exists and its sticky bit is set.

35111 `-C file` True if *file* is a contiguous file.

35112 `-V file` True if *file* is a version file.

35113 The following option was not included because it was undocumented in most implementations,  
35114 has been removed from some implementations (including System V), and the functionality is  
35115 provided by the shell (see Section 2.6.2 (on page 2239)).

35116 `-l string` The length of the string *string*.

35117 The `-b`, `-c`, `-g`, `-p`, `-u`, and `-x` operands are derived from the SVID; historical BSD does not  
35118 provide them. The `-k` operand is derived from System V; historical BSD does not provide it.

35119 On historical BSD systems, `test -w directory` always returned false because `test` tried to open the  
35120 directory for writing, which always fails.

35121 Some additional primaries newly invented or from the KornShell appeared in an early proposal  
35122 as part of the conditional command (`[[]]`): `s1 > s2`, `s1 < s2`, `str = pattern`, `str != pattern`, `f1 -nt f2`, `f1`  
35123 `-ot f2`, and `f1 -ef f2`. They were not carried forward into the `test` utility when the conditional  
35124 command was removed from the shell because they have not been included in the `test` utility  
35125 built into historical implementations of the `sh` utility.

35126 The `-t file_descriptor` primary is shown with a mandatory argument because the grammar is  
35127 ambiguous if it can be omitted. Historical implementations have allowed it to be omitted,  
35128 providing a default of 1.

35129 **FUTURE DIRECTIONS**

35130 None.

35131 **SEE ALSO**35132 *find*35133 **CHANGE HISTORY**

35134 First released in Issue 2.

35135 **Issue 5**

35136 FUTURE DIRECTIONS section added.

35137 **Issue 6**35138 The **-h** operand is added for symbolic links, and access permission requirements are clarified for  
35139 the **-r**, **-w**, and **-x** operands to align with the IEEE P1003.2b draft standard.

35140 The normative text is reworded to avoid use of the term “must” for application requirements.

35141 The **-L** and **-S** operands are added for symbolic links and sockets.

35142 **NAME**

35143           time — time a simple command

35144 **SYNOPSIS**35145 UP       time [-p] *utility* [*argument...*]

35146

35147 **DESCRIPTION**35148       The *time* utility shall invoke the utility named by the *utility* operand with arguments supplied as the *argument* operands and write a message to standard error that lists timing statistics for the utility. The message shall include the following information:

- 35151           • The elapsed (real) time between invocation of *utility* and its termination.
- 35152           • The User CPU time, equivalent to the sum of the *tms\_utime* and *tms\_cutime* fields returned by the *times()* function defined in the System Interfaces volume of IEEE Std 1003.1-200x for the process in which *utility* is executed.
- 35153           • The System CPU time, equivalent to the sum of the *tms\_stime* and *tms\_cstime* fields returned by the *times()* function for the process in which *utility* is executed.

35157       The precision of the timing shall be no less than the granularity defined for the size of the clock tick unit on the system, but the results shall be reported in terms of standard time units (for example, 0.02 seconds, 00:00:00.02, 1m33.75s, 365.21 seconds), not numbers of clock ticks.

35160       When *time* is used as part of a pipeline, the times reported are unspecified, except when it is the sole command within a grouping command (see Section 2.9.4.1 (on page 2253)) in that pipeline. For example, the commands on the left are unspecified; those on the right report on utilities **a** and **c**, respectively:

```
35164       time a | b | c       { time a } | b | c
35165       a | b | time c       a | b | (time c)
```

35166 **OPTIONS**35167       The *time* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2, Utility Syntax Guidelines.

35169       The following option shall be supported:

35170       **-p**           Write the timing output to standard error in the format shown in the **STDERR**

35171                      section.

35172 **OPERANDS**

35173       The following operands shall be supported:

35174       *utility*       The name of a utility that is to be invoked. If the *utility* operand names any of the special built-in utilities in Section 2.14 (on page 2266), the results are undefined.35176       *argument*     Any string to be supplied as an argument when invoking the utility named by the *utility* operand.35178 **STDIN**

35179       Not used.

35180 **INPUT FILES**

35181       None.

## 35182 ENVIRONMENT VARIABLES

35183 The following environment variables shall affect the execution of *time*:

35184 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 35185 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
 35186 Internationalization Variables for the precedence of internationalization variables  
 35187 used to determine the values of locale categories.)

35188 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 35189 internationalization variables.

35190 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 35191 characters (for example, single-byte as opposed to multi-byte characters in  
 35192 arguments).

35193 *LC\_MESSAGES*

35194 Determine the locale that should be used to affect the format and contents of  
 35195 diagnostic and informative messages written to standard error.

35196 *LC\_NUMERIC*

35197 Determine the locale for numeric formatting.

35198 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

35199 *PATH* Determine the search path that shall be used to locate the utility to be invoked; see  
 35200 the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 8, Environment  
 35201 Variables.

## 35202 ASYNCHRONOUS EVENTS

35203 Default.

35204 *STDOUT*

35205 Not used.

35206 *STDERR*

35207 The standard error shall be used to write the timing statistics. If *-p* is specified, the following  
 35208 format shall be used in the POSIX locale:

35209 "real %f\nuser %f\nsys %f\n", <real seconds>, <user seconds>,  
 35210 <system seconds>

35211 where each floating-point number shall be expressed in seconds. The precision used may be less  
 35212 than the default six digits of %f, but shall be sufficiently precise to accommodate the size of the  
 35213 clock tick on the system (for example, if there were 60 clock ticks per second, at least two digits  
 35214 shall follow the radix character). The number of digits following the radix character shall be no  
 35215 less than one, even if this always results in a trailing zero. The implementation may append  
 35216 white space and additional information following the format shown here.

## 35217 OUTPUT FILES

35218 None.

## 35219 EXTENDED DESCRIPTION

35220 None.

## 35221 EXIT STATUS

35222 If the *utility* utility is invoked, the exit status of *time* shall be the exit status of *utility*; otherwise,  
 35223 the *time* utility shall exit with one of the following values:

35224 1-125 An error occurred in the *time* utility.

35225 126 The utility specified by *utility* was found but could not be invoked.

35226 127 The utility specified by *utility* could not be found.

### 35227 CONSEQUENCES OF ERRORS

35228 Default.

### 35229 APPLICATION USAGE

35230 The *command*, *env*, *nice*, *nohup*, *time*, and *xargs* utilities have been specified to use exit code 127 if  
 35231 an error occurs so that applications can distinguish “failure to find a utility” from “invoked  
 35232 utility exited with an error indication”. The value 127 was chosen because it is not commonly  
 35233 used for other meanings; most utilities use small values for “normal error conditions” and the  
 35234 values above 128 can be confused with termination due to receipt of a signal. The value 126 was  
 35235 chosen in a similar manner to indicate that the utility could be found, but not invoked. Some  
 35236 scripts produce meaningful error messages differentiating the 126 and 127 cases. The distinction  
 35237 between exit codes 126 and 127 is based on KornShell practice that uses 127 when all attempts to  
 35238 *exec* the utility fail with [ENOENT], and uses 126 when any attempt to *exec* the utility fails for  
 35239 any other reason.

### 35240 EXAMPLES

35241 It is frequently desirable to apply *time* to pipelines or lists of commands. This can be done by  
 35242 placing pipelines and command lists in a single file; this file can then be invoked as a utility, and  
 35243 the *time* applies to everything in the file.

35244 Alternatively, the following command can be used to apply *time* to a complex command:

```
35245 time sh -c 'complex-command-line'
```

### 35246 RATIONALE

35247 When the *time* utility was originally proposed to be included in the earlier version of |  
 35248 IEEE Std 1003.1, questions were raised about its suitability for inclusion on the grounds that it |  
 35249 was not useful for portable applications, specifically: |

- 35250 • The underlying CPU definitions from the System Interfaces volume of IEEE Std 1003.1-200x  
 35251 are vague, so the numeric output could not be compared accurately between systems or even  
 35252 between invocations.
- 35253 • The creation of portable benchmark programs was outside the scope this volume of  
 35254 IEEE Std 1003.1-200x.

35255 However, *time* does fit in the scope of user portability. Human judgement can be applied to the  
 35256 analysis of the output, and it could be very useful in hands-on debugging of applications or in  
 35257 providing subjective measures of system performance. Hence it has been included in this  
 35258 volume of IEEE Std 1003.1-200x.

35259 The default output format has been left unspecified because historical implementations differ  
 35260 greatly in their style of depicting this numeric output. The **-p** option was invented to provide  
 35261 scripts a common means of obtaining this information.

35262 In the KornShell, *time* is a shell reserved word that can be used to time an entire pipeline, rather  
 35263 than just a simple command. The POSIX definition has been worded to allow this  
 35264 implementation. Consideration was given to invalidating this approach because of the historical  
 35265 model from the C shell and System V shell. However, since the System V *time* utility historically  
 35266 has not produced accurate results in pipeline timing (because the constituent processes are not  
 35267 all owned by the same parent process, as allowed by POSIX), it did not seem worthwhile to  
 35268 break historical KornShell usage.

35269 The term *utility* is used, rather than *command*, to highlight the fact that shell compound  
 35270 commands, pipelines, special built-ins, and so on, cannot be used directly. However, *utility*

35271 includes user application programs and shell scripts, not just the standard utilities.

35272 **FUTURE DIRECTIONS**

35273 None.

35274 **SEE ALSO**

35275 *sh*, the System Interfaces volume of IEEE Std 1003.1-200x, *times()*

35276 **CHANGE HISTORY**

35277 First released in Issue 2.

35278 **Issue 6**

35279 This utility is now marked as part of the User Portability Utilities option.

## 35280 NAME

35281 touch — change file access and modification times

## 35282 SYNOPSIS

35283 touch [-acm][ -r *ref\_file* | -t *time*] *file*...

## 35284 DESCRIPTION

35285 The *touch* utility shall change the modification times, access times, or both of files. The  
 35286 modification time shall be equivalent to the value of the *st\_mtime* member of the **stat** structure  
 35287 for a file, as described in the System Interfaces volume of IEEE Std 1003.1-200x; the access time  
 35288 shall be equivalent to the value of *st\_atime*.

35289 The time used can be specified by the **-t** *time* option-argument, the corresponding time fields of  
 35290 the file referenced by the **-r** *ref\_file* option-argument, or the *date\_time* operand, as specified in the  
 35291 following sections. If none of these are specified, *touch* shall use the current time (the value  
 35292 returned by the equivalent of the *time()* function defined in the System Interfaces volume of  
 35293 IEEE Std 1003.1-200x).

35294 For each *file* operand, *touch* shall perform actions equivalent to the following functions defined  
 35295 in the System Interfaces volume of IEEE Std 1003.1-200x:

- 35296 1. If *file* does not exist, a *creat()* function call is made with the *file* operand used as the *path*  
 35297 argument and the value of the bitwise-inclusive OR of S\_IRUSR, S\_IWUSR, S\_IRGRP,  
 35298 S\_IWGRP, S\_IROTH, and S\_IWOTH used as the *mode* argument.
- 35299 2. The *utime()* function is called with the following arguments:
  - 35300 a. The *file* operand is used as the *path* argument.
  - 35301 b. The **utimbuf** structure members *actime* and *modtime* are determined as described in  
 35302 the OPTIONS section.

## 35303 OPTIONS

35304 The *touch* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section  
 35305 12.2, Utility Syntax Guidelines.

35306 The following options shall be supported:

- |                |                           |   |  |
|----------------|---------------------------|---|--|
| 35307<br>35308 | <b>-a</b>                 | Change the access time of <i>file</i> . Do not change the modification time unless <b>-m</b> is also specified.             |  |
| 35309<br>35310 | <b>-c</b>                 | Do not create a specified <i>file</i> if it does not exist. Do not write any diagnostic messages concerning this condition. |  |
| 35311<br>35312 | <b>-m</b>                 | Change the modification time of <i>file</i> . Do not change the access time unless <b>-a</b> is also specified.             |  |
| 35313<br>35314 | <b>-r</b> <i>ref_file</i> | Use the corresponding time of the file named by the pathname <i>ref_file</i> instead of the current time.                   |  |
| 35315<br>35316 | <b>-t</b> <i>time</i>     | Use the specified <i>time</i> instead of the current time. The option-argument shall be a decimal number of the form:       |  |
| 35317          |                           | [[ <i>CC</i> ][ <i>YY</i> ] <i>MMDDhmm</i> [. <i>SS</i> ]   |  |
| 35318          |                           | where each two digits represents the following:   |  |
| 35319          | <i>MM</i>                 | The month of the year [01,12].  |  |
| 35320          | <i>DD</i>                 | The day of the month [01,31].   |  |

35321	<i>hh</i>	The hour of the day [00,23].	
35322	<i>mm</i>	The minute of the hour [00,59].	
35323	<i>CC</i>	The first two digits of the year (the century).	
35324	<i>YY</i>	The second two digits of the year.	
35325	<i>SS</i>	The second of the minute [00,60].	
35326		Both <i>CC</i> and <i>YY</i> shall be optional. If neither is given, the current year shall be	
35327		assumed. If <i>YY</i> is specified, but <i>CC</i> is not, <i>CC</i> shall be derived as follows:	
35328			
35329			
35330			
35331	<b>Note:</b>	It is expected that in a future version of IEEE Std 1003.1-200x the default	
35332		century inferred from a 2-digit year will change. (This would apply to all	
35333		commands accepting a 2-digit year as input.)	
35334		The resulting time shall be affected by the value of the <i>TZ</i> environment variable. If	
35335		the resulting time value precedes the Epoch, <i>touch</i> shall exit immediately with an	
35336		error status. The range of valid times past the Epoch is implementation-defined,	
35337		but it shall extend to at least the time 0 hours, 0 minutes, 0 seconds, January 1,	
35338		2038, Coordinated Universal Time. Some implementations may not be able to	
35339		represent dates beyond the January 18, 2038, because they use <b>signed int</b> as a time	
35340		holder.	
35341		The range for <i>SS</i> is [00,60] rather than [00,59] because of leap seconds. If <i>SS</i> is 60,	
35342		and the resulting time, as affected by the <i>TZ</i> environment variable, does not refer	
35343		to a leap second, the resulting time shall be one second after a time where <i>SS</i> is 59.	
35344		If <i>SS</i> is not given a value, it is assumed to be zero.	
35345		If neither the <b>-a</b> nor <b>-m</b> options were specified, <i>touch</i> shall behave as if both the <b>-a</b> and <b>-m</b>	
35346		options were specified.	
35347	<b>OPERANDS</b>		
35348		The following operands shall be supported:	
35349	<i>file</i>	A pathname of a file whose times shall be modified.	
35350	<b>STDIN</b>		
35351		Not used.	
35352	<b>INPUT FILES</b>		
35353		None.	
35354	<b>ENVIRONMENT VARIABLES</b>		
35355		The following environment variables shall affect the execution of <i>touch</i> :	
35356	<i>LANG</i>	Provide a default value for the internationalization variables that are unset or null.	
35357		(See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,	
35358		Internationalization Variables for the precedence of internationalization variables	
35359		used to determine the values of locale categories.)	
35360	<i>LC_ALL</i>	If set to a non-empty string value, override the values of all the other	
35361		internationalization variables.	
35362	<i>LC_CTYPE</i>	Determine the locale for the interpretation of sequences of bytes of text data as	
35363		characters (for example, single-byte as opposed to multi-byte characters in	

- 35364 arguments).
- 35365 **LC\_MESSAGES**
- 35366 Determine the locale that should be used to affect the format and contents of  
35367 diagnostic messages written to standard error.
- 35368 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 35369 **TZ** Determine the timezone to be used for interpreting the *time* option-argument. If *TZ*  
35370 is unset or null, an unspecified default timezone shall be used.
- 35371 **ASYNCHRONOUS EVENTS**
- 35372 Default.
- 35373 **STDOUT**
- 35374 Not used.
- 35375 **STDERR**
- 35376 The standard error shall be used only for diagnostic messages. |
- 35377 **OUTPUT FILES**
- 35378 None.
- 35379 **EXTENDED DESCRIPTION**
- 35380 None.
- 35381 **EXIT STATUS**
- 35382 The following exit values shall be returned:
- 35383 0 The utility executed successfully and all requested changes were made.
- 35384 >0 An error occurred.
- 35385 **CONSEQUENCES OF ERRORS**
- 35386 Default.
- 35387 **APPLICATION USAGE**
- 35388 The interpretation of time is taken to be *seconds since the Epoch* (see the Base Definitions volume |  
35389 of IEEE Std 1003.1-200x, Section 4.14, Seconds Since the Epoch). It should be noted that |  
35390 implementations conforming to the System Interfaces volume of IEEE Std 1003.1-200x do not |  
35391 take leap seconds into account when computing seconds since the Epoch. When *SS=60* is used,  
35392 the resulting time always refers to 1 plus *seconds since the Epoch* for a time when *SS=59*.
- 35393 Although the *-t time* option-argument specifies values in 1969, the access time and modification  
35394 time fields are defined in terms of seconds since the Epoch (00:00:00 on 1 January 1970 UTC). |  
35395 Therefore, depending on the value of *TZ* when *touch* is run, there is never more than a few valid |  
35396 hours in 1969 and there need not be any valid times in 1969.
- 35397 One ambiguous situation occurs if *-t time* is not specified, *-r ref\_file* is not specified, and the first  
35398 operand is an eight or ten-digit decimal number. A portable script can avoid this problem by  
35399 using:
- 35400 `touch -- file`
- 35401 or:
- 35402 `touch ./file`
- 35403 in this case.

35404 **EXAMPLES**

35405 None.

35406 **RATIONALE**

35407 The functionality of *touch* is described almost entirely through references to functions in the  
 35408 System Interfaces volume of IEEE Std 1003.1-200x. In this way, there is no duplication of effort  
 35409 required for describing such side effects as the relationship of user IDs to the user database,  
 35410 permissions, and so on.

35411 There are some significant differences between the *touch* utility in this volume of  
 35412 IEEE Std 1003.1-200x and those in System V and BSD systems. They are upward-compatible for  
 35413 historical applications from both implementations:

35414 1. In System V, an ambiguity exists when a pathname that is a decimal number leads the  
 35415 operands; it is treated as a time value. In BSD, no *time* value is allowed; files may only be  
 35416 *touched* to the current time. The `-t time` construct solves these problems for future  
 35417 conforming applications (note that the `-t` option is not historical practice).

35418 2. The inclusion of the century digits, *CC*, is also new. Note that a ten-digit *time* value is  
 35419 treated as if *YY*, and not *CC*, were specified. The caveat about the range of dates following  
 35420 the Epoch was included as recognition that some implementations are not able to  
 35421 represent dates beyond 18 January 2038 because they use **signed int** as a time holder.

35422 The `-r` option was added because several comments requested this capability. This option was  
 35423 named `-f` in an early proposal, but was changed because the `-f` option is used in the BSD version  
 35424 of *touch* with a different meaning.

35425 At least one historical implementation of *touch* incremented the exit code if `-c` was specified and  
 35426 the file did not exist. This volume of IEEE Std 1003.1-200x requires exit status zero if no errors  
 35427 occur.

35428 **FUTURE DIRECTIONS**35429 Applications should use the `-r` or `-t` options.35430 **SEE ALSO**35431 *date*, the System Interfaces volume of IEEE Std 1003.1-200x, *creat()*, *time()*, `<sys/stat.h>`35432 **CHANGE HISTORY**

35433 First released in Issue 2.

35434 **Issue 6**35435 The obsolescent *date\_time* operand is removed.

35436 The Open Group Corrigendum U027/1 is applied. This extends the range of valid time past the  
 35437 Epoch to at least the time 0 hours, 0 minutes, 0 seconds, January 1, 2038, Coordinated Universal  
 35438 Time. This is a new requirement on POSIX implementations.

35439 The range for double leap seconds is changed from [00,61] to [00,60] to align with the  
 35440 ISO/IEC 9899:1999 standard.

35441 **NAME**

35442 tput — change terminal characteristics

35443 **SYNOPSIS**35444 UP tput [-T *type*] *operand...*

35445

35446 **DESCRIPTION**

35447 The *tput* utility shall display terminal-dependent information. The manner in which this  
 35448 information is retrieved is unspecified. The information displayed shall clear the terminal screen,  
 35449 initialize the user's terminal, or reset the user's terminal, depending on the operand given. The  
 35450 exact consequences of displaying this information are unspecified.

35451 **OPTIONS**

35452 The *tput* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section  
 35453 12.2, Utility Syntax Guidelines.

35454 The following option shall be supported:

35455 **-T *type*** Indicate the type of terminal. If this option is not supplied and the *TERM* variable  
 35456 is unset or null, an unspecified default terminal type shall be used. The setting of  
 35457 *type* shall take precedence over the value in *TERM*.

35458 **OPERANDS**

35459 The following strings shall be supported as operands by the implementation in the POSIX locale:

35460 **clear** Display the clear-screen sequence.

35461 **init** Display the sequence that initializes the user's terminal in an implementation-  
 35462 defined manner.

35463 **reset** Display the sequence that resets the user's terminal in an implementation-defined  
 35464 manner.

35465 If a terminal does not support any of the operations described by these operands, this shall not  
 35466 be considered an error condition.

35467 **STDIN**

35468 Not used.

35469 **INPUT FILES**

35470 None.

35471 **ENVIRONMENT VARIABLES**

35472 The following environment variables shall affect the execution of *tput*:

35473 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 35474 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
 35475 Internationalization Variables for the precedence of internationalization variables  
 35476 used to determine the values of locale categories.)

35477 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 35478 internationalization variables.

35479 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 35480 characters (for example, single-byte as opposed to multi-byte characters in  
 35481 arguments).

35482 **LC\_MESSAGES**

35483 Determine the locale that should be used to affect the format and contents of  
 35484 diagnostic messages written to standard error.

- 35485 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 35486 **TERM** Determine the terminal type. If this variable is unset or null, and if the **-T** option is  
35487 not specified, an unspecified default terminal type shall be used.
- 35488 **ASYNCHRONOUS EVENTS**
- 35489 Default.
- 35490 **STDOUT**
- 35491 If standard output is a terminal device, it may be used for writing the appropriate sequence to  
35492 clear the screen or reset or initialize the terminal. If standard output is not a terminal device,  
35493 undefined results occur.
- 35494 **STDERR**
- 35495 The standard error shall be used only for diagnostic messages.
- 35496 **OUTPUT FILES**
- 35497 None.
- 35498 **EXTENDED DESCRIPTION**
- 35499 None.
- 35500 **EXIT STATUS**
- 35501 The following exit values shall be returned:
- 35502 0 The requested string was written successfully.
- 35503 1 Unspecified.
- 35504 2 Usage error.
- 35505 3 No information is available about the specified terminal type.
- 35506 4 The specified operand is invalid.
- 35507 >4 An error occurred.
- 35508 **CONSEQUENCES OF ERRORS**
- 35509 If one of the operands is not available for the terminal, *tput* continues processing the remaining  
35510 operands.
- 35511 **APPLICATION USAGE**
- 35512 The difference between resetting and initializing a terminal is left unspecified, as they vary  
35513 greatly based on hardware types. In general, resetting is a more severe action.
- 35514 Some terminals use control characters to perform the stated functions, and on such terminals it  
35515 might make sense to use *tput* to store the initialization strings in a file or environment variable  
35516 for later use. However, because other terminals might rely on system calls to do this work, the  
35517 standard output cannot be used in a portable manner, such as the following non-portable  
35518 constructs:
- 35519 ClearVar=`tput clear`  
35520 tput reset | mailx -s "Wake Up" ddg
- 35521 **EXAMPLES**
- 35522 1. Initialize the terminal according to the type of terminal in the environmental variable  
35523 *TERM*. This command can be included in a **.profile** file.
- 35524 tput init
- 35525 2. Reset a 450 terminal.

35526           tput -T 450 reset

35527 **RATIONALE**

35528           The list of operands was reduced to a minimum for the following reasons:

35529           • The only features chosen were those that were likely to be used by human users interacting  
35530           with a terminal.

35531           • Specifying the full *terminfo* set was not considered desirable, but the standard developers did  
35532           not want to select among operands.

35533           • This volume of IEEE Std 1003.1-200x does not attempt to provide applications with  
35534           sophisticated terminal handling capabilities, as that falls outside of its assigned scope and  
35535           intersects with the responsibilities of other standards bodies.

35536           The difference between resetting and initializing a terminal is left unspecified as this varies  
35537           greatly based on hardware types. In general, resetting is a more severe action.

35538           The exit status of 1 is historically reserved for finding out if a Boolean operand is not set.  
35539           Although the operands were reduced to a minimum, the exit status of 1 should still be reserved  
35540           for the Boolean operands, for those sites that wish to support them.

35541 **FUTURE DIRECTIONS**

35542           None.

35543 **SEE ALSO**

35544           *stty*, *tabs*

35545 **CHANGE HISTORY**

35546           First released in Issue 4.

35547 **Issue 6**

35548           This utility is now marked as part of the User Portability Utilities option.

35549 **NAME**

35550 tr — translate characters

35551 **SYNOPSIS**35552 tr [-c | -C][-s] *string1 string2*35553 tr -s [-c | -C] *string1*35554 tr -d [-c | -C] *string1*35555 tr -ds [-c | -C] *string1 string2*35556 **DESCRIPTION**

35557 The *tr* utility shall copy the standard input to the standard output with substitution or deletion  
 35558 of selected characters. The options specified and the *string1* and *string2* operands shall control  
 35559 translations that occur while copying characters and single-character collating elements.

35560 **OPTIONS**

35561 The *tr* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2,  
 35562 Utility Syntax Guidelines.

35563 The following options shall be supported:

35564 **-c** Complement the set of values specified by *string1*. See the EXTENDED  
 35565 DESCRIPTION section.

35566 **-C** Complement the set of characters specified by *string1*. See the EXTENDED  
 35567 DESCRIPTION section.

35568 **-d** Delete all occurrences of input characters that are specified by *string1*.

35569 **-s** Replace instances of repeated characters with a single character, as described in the  
 35570 EXTENDED DESCRIPTION section.

35571 **OPERANDS**

35572 The following operands shall be supported:

35573 *string1, string2*

35574 Translation control strings. Each string shall represent a set of characters to be  
 35575 converted into an array of characters used for the translation. For a detailed  
 35576 description of how the strings are interpreted, see the EXTENDED DESCRIPTION  
 35577 section.

35578 **STDIN**

35579 The standard input can be any type of file.

35580 **INPUT FILES**

35581 None.

35582 **ENVIRONMENT VARIABLES**

35583 The following environment variables shall affect the execution of *tr*:

35584 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 35585 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
 35586 Internationalization Variables for the precedence of internationalization variables  
 35587 used to determine the values of locale categories.)

35588 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 35589 internationalization variables.

35590 **LC\_COLLATE**

35591 Determine the locale for the behavior of range expressions and equivalence classes.

- 35592 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 35593 characters (for example, single-byte as opposed to multi-byte characters in  
 35594 arguments) and the behavior of character classes.
- 35595 **LC\_MESSAGES**  
 35596 Determine the locale that should be used to affect the format and contents of  
 35597 diagnostic messages written to standard error.
- 35598 XSI **NLSPATH** Determine the location of message catalogs for the processing of **LC\_MESSAGES**.
- 35599 **ASYNCHRONOUS EVENTS**  
 35600 Default.
- 35601 **STDOUT**  
 35602 The *tr* output shall be identical to the input, with the exception of the specified transformations.
- 35603 **STDERR**  
 35604 The standard error shall be used only for diagnostic messages.
- 35605 **OUTPUT FILES**  
 35606 None.
- 35607 **EXTENDED DESCRIPTION**  
 35608 The operands *string1* and *string2* (if specified) define two arrays of characters. The constructs in  
 35609 the following list can be used to specify characters or single-character collating elements. If any  
 35610 of the constructs result in multi-character collating elements, *tr* shall exclude, without a  
 35611 diagnostic, those multi-character elements from the resulting array.
- 35612 *character* Any character not described by one of the conventions below shall represent itself.
- 35613 *\octal* Octal sequences can be used to represent characters with specific coded values. An  
 35614 octal sequence shall consist of a backslash followed by the longest sequence of one,  
 35615 two, or three-octal-digit characters (01234567). The sequence shall cause the value  
 35616 whose encoding is represented by the one, two, or three-digit octal integer to be  
 35617 placed into the array. If the size of a byte on the system is greater than nine bits, the  
 35618 valid escape sequence used to represent a byte is implementation-defined. Multi-  
 35619 byte characters require multiple, concatenated escape sequences of this type,  
 35620 including the leading '\ ' for each byte.
- 35621 *\character* The backslash-escape sequences in the Base Definitions volume of  
 35622 IEEE Std 1003.1-200x, Table 5-1, Escape Sequences and Associated Actions ('\ ',  
 35623 '\a', '\b', '\f', '\n', '\r', '\t', '\v') shall be supported. The results of  
 35624 using any other character, other than an octal digit, following the backslash are  
 35625 unspecified.
- 35626 *c-c* In the POSIX locale, this construct shall represent the range of collating elements  
 35627 between the range endpoints (as long as neither endpoint is an octal sequence of  
 35628 the form *\octal*), inclusive, as defined by the collation sequence. The characters or  
 35629 collating elements in the range shall be placed in the array in ascending collation  
 35630 sequence. If the second endpoint precedes the starting endpoint in the collation  
 35631 sequence, it is unspecified whether the range of collating elements is empty, or this  
 35632 construct is treated as invalid. In locales other than the POSIX locale, this construct  
 35633 has unspecified behavior.
- 35634 If either or both of the range endpoints are octal sequences of the form *\octal*, this  
 35635 shall represent the range of specific coded values between the two range  
 35636 endpoints, inclusive.

35637	<code>[:class:]</code>	Represents all characters belonging to the defined character class, as defined by the current setting of the <code>LC_CTYPE</code> locale category. The following character class names shall be accepted when specified in <code>string1</code> :
35638		
35639		
35640		<b>alnum</b> <b>blank</b> <b>digit</b> <b>lower</b> <b>punct</b> <b>upper</b>
35641		<b>alpha</b> <b>cntrl</b> <b>graph</b> <b>print</b> <b>space</b> <b>xdigit</b>
35642 XSI		In addition, character class expressions of the form <code>[:name:]</code> shall be recognized in those locales where the <code>name</code> keyword has been given a <b>charclass</b> definition in the <code>LC_CTYPE</code> category.
35643		
35644		
35645		When both the <code>-d</code> and <code>-s</code> options are specified, any of the character class names shall be accepted in <code>string2</code> . Otherwise, only character class names <b>lower</b> or <b>upper</b> are valid in <code>string2</code> and then only if the corresponding character class ( <b>upper</b> and <b>lower</b> , respectively) is specified in the same relative position in <code>string1</code> . Such a specification shall be interpreted as a request for case conversion. When <code>[:lower:]</code> appears in <code>string1</code> and <code>[:upper:]</code> appears in <code>string2</code> , the arrays shall contain the characters from the <b>toupper</b> mapping in the <code>LC_CTYPE</code> category of the current locale. When <code>[:upper:]</code> appears in <code>string1</code> and <code>[:lower:]</code> appears in <code>string2</code> , the arrays shall contain the characters from the <b>tolower</b> mapping in the <code>LC_CTYPE</code> category of the current locale. The first character from each mapping pair shall be in the array for <code>string1</code> and the second character from each mapping pair shall be in the array for <code>string2</code> in the same relative position.
35646		
35647		
35648		
35649		
35650		
35651		
35652		
35653		
35654		
35655		
35656		
35657		Except for case conversion, the characters specified by a character class expression shall be placed in the array in an unspecified order.
35658		
35659		If the name specified for <code>class</code> does not define a valid character class in the current locale, the behavior is undefined.
35660		
35661	<code>[=equiv=]</code>	Represents all characters or collating elements belonging to the same equivalence class as <code>equiv</code> , as defined by the current setting of the <code>LC_COLLATE</code> locale category. An equivalence class expression shall be allowed only in <code>string1</code> , or in <code>string2</code> when it is being used by the combined <code>-d</code> and <code>-s</code> options. The characters belonging to the equivalence class shall be placed in the array in an unspecified order.
35662		
35663		
35664		
35665		
35666		
35667	<code>[x*n]</code>	Represents <code>n</code> repeated occurrences of the character <code>x</code> . Because this expression is used to map multiple characters to one, it is only valid when it occurs in <code>string2</code> . If <code>n</code> is omitted or is zero, it shall be interpreted as large enough to extend the <code>string2</code> -based sequence to the length of the <code>string1</code> -based sequence. If <code>n</code> has a leading zero, it shall be interpreted as an octal value. Otherwise, it shall be interpreted as a decimal value.
35668		
35669		
35670		
35671		
35672		
35673		When the <code>-d</code> option is not specified:
35674		• Each input character found in the array specified by <code>string1</code> shall be replaced by the character in the same relative position in the array specified by <code>string2</code> . When the array specified by <code>string2</code> is shorter than the one specified by <code>string1</code> , the results are unspecified.
35675		
35676		
35677		• If the <code>-C</code> option is specified, the complements of the characters specified by <code>string1</code> (the set of all characters in the current character set, as defined by the current setting of <code>LC_CTYPE</code> , except for those actually specified in the <code>string1</code> operand) shall be placed in the array in ascending collation sequence, as defined by the current setting of <code>LC_COLLATE</code> .
35678		
35679		
35680		
35681		• If the <code>-c</code> option is specified, the complement of the values specified by <code>string1</code> shall be placed in the array in ascending order by binary value.
35682		

35683 • Because the order in which characters specified by character class expressions or equivalence  
 35684 class expressions is undefined, such expressions should only be used if the intent is to map  
 35685 several characters into one. An exception is case conversion, as described previously.

35686 When the `-d` option is specified:

- 35687 • Input characters found in the array specified by *string1* shall be deleted.
- 35688 • When the `-C` option is specified with `-d`, all characters except those specified by *string1* shall  
 35689 be deleted. The contents of *string2* are ignored, unless the `-s` option is also specified.
- 35690 • When the `-c` option is specified with `-d`, all values except those specified by *string1* shall be  
 35691 deleted. The contents of *string2* shall be ignored, unless the `-s` option is also specified.
- 35692 • The same string cannot be used for both the `-d` and the `-s` option; when both options are  
 35693 specified, both *string1* (used for deletion) and *string2* (used for squeezing) shall be required.

35694 When the `-s` option is specified, after any deletions or translations have taken place, repeated  
 35695 sequences of the same character shall be replaced by one occurrence of the same character, if the  
 35696 character is found in the array specified by the last operand. If the last operand contains a  
 35697 character class, such as the following example:

```
35698 tr -s '[:space:]'
```

35699 the last operand's array shall contain all of the characters in that character class. However, in a  
 35700 case conversion, as described previously, such as:

```
35701 tr -s '[:upper:]' '[:lower:]'
```

35702 the last operand's array shall contain only those characters defined as the second characters in  
 35703 each of the **toupper** or **tolower** character pairs, as appropriate.

35704 An empty string used for *string1* or *string2* produces undefined results.

#### 35705 EXIT STATUS

35706 The following exit values shall be returned:

35707 0 All input was processed successfully.

35708 >0 An error occurred.

#### 35709 CONSEQUENCES OF ERRORS

35710 Default.

#### 35711 APPLICATION USAGE

35712 If necessary, *string1* and *string2* can be quoted to avoid pattern matching by the shell.

35713 If an ordinary digit (representing itself) is to follow an octal sequence, the octal sequence must  
 35714 use the full three digits to avoid ambiguity.

35715 When *string2* is shorter than *string1*, a difference results between historical System V and BSD  
 35716 systems. A BSD system pads *string2* with the last character found in *string2*. Thus, it is possible  
 35717 to do the following:

```
35718 tr 0123456789 d
```

35719 which would translate all digits to the letter 'd'. Since this area is specifically unspecified in  
 35720 this volume of IEEE Std 1003.1-200x, both the BSD and System V behaviors are allowed, but a  
 35721 conforming application cannot rely on the BSD behavior. It would have to code the example in  
 35722 the following way:

```
35723 tr 0123456789 '[d*]'
```

35724 It should be noted that, despite similarities in appearance, the string operands used by *tr* are not  
35725 regular expressions.

35726 Unlike some historical implementations, this definition of the *tr* utility correctly processes NUL  
35727 characters in its input stream. NUL characters can be stripped by using:

```
35728 tr -d '\000'
```

### 35729 EXAMPLES

35730 1. The following example creates a list of all words in **file1** one per line in **file2**, where a word  
35731 is taken to be a maximal string of letters.

```
35732 tr -cs "[:alpha:]" "[\n*]" <file1 >file2
```

35733 2. The next example translates all lowercase characters in **file1** to uppercase and writes the  
35734 results to standard output.

```
35735 tr "[:lower:]" "[:upper:]" <file1
```

35736 3. This example uses an equivalence class to identify accented variants of the base character  
35737 'e' in **file1**, which are stripped of diacritical marks and written to **file2**.

```
35738 tr "[=e=]" e <file1 >file2
```

### 35739 RATIONALE

35740 In some early proposals, an explicit option **-n** was added to disable the historical behavior of  
35741 stripping NUL characters from the input. It was considered that automatically stripping NUL  
35742 characters from the input was not correct functionality. However, the removal of **-n** in a later  
35743 proposal does not remove the requirement that *tr* correctly process NUL characters in its input  
35744 stream. NUL characters can be stripped by using *tr -d '\000'*.

35745 Historical implementations of *tr* differ widely in syntax and behavior. For example, the BSD  
35746 version has not needed the bracket characters for the repetition sequence. The POSIX Shell and  
35747 Utilities *tr* syntax is based more closely on the System V and XPG3 model while attempting to  
35748 accommodate historical BSD implementations. In the case of the short *string2* padding, the  
35749 decision was to unspecified the behavior and preserve System V and XPG3 scripts, which might  
35750 find difficulty with the BSD method. The assumption was made that BSD users of *tr* have to  
35751 make accommodations to meet the POSIX Shell and Utilities syntax. Since it is possible to use  
35752 the repetition sequence to duplicate the desired behavior, whereas there is no simple way to  
35753 achieve the System V method, this was the correct, if not desirable, approach.

35754 The use of octal values to specify control characters, while having historical precedents, is not  
35755 portable. The introduction of escape sequences for control characters should provide the  
35756 necessary portability. It is recognized that this may cause some historical scripts to break.

35757 An early proposal included support for multi-character collating elements. It was pointed out  
35758 that, while *tr* does employ some syntactical elements from REs, the aim of *tr* is quite different;  
35759 ranges, for example, do not have a similar meaning (“any of the chars in the range matches”,  
35760 *versus* “translate each character in the range to the output counterpart”). As a result, the  
35761 previously included support for multi-character collating elements has been removed. What  
35762 remains are ranges in current collation order (to support, for example, accented characters),  
35763 character classes, and equivalence classes.

35764 In XPG3 the `[:class:]` and `[=equiv=]` conventions are shown with double brackets, as in RE syntax.  
35765 However, *tr* does not implement RE principles; it just borrows part of the syntax. Consequently,  
35766 `[:class:]` and `[=equiv=]` should be regarded as syntactical elements on a par with `[x*n]`, which is  
35767 not an RE bracket expression.

- 35768 The standard developers will consider changes to *tr* that allow it to translate characters between  
 35769 different character encodings, or they will consider providing a new utility to accomplish this.
- 35770 On historical System V systems, a range expression requires enclosing square-brackets, such as:  
 35771 `tr '[a-z]' '[A-Z]'`
- 35772 However, BSD-based systems did not require the brackets, and this convention is used by POSIX  
 35773 Shell and Utilities to avoid breaking large numbers of BSD scripts:
- 35774 `tr a-z A-Z`
- 35775 The preceding System V script will continue to work because the brackets, treated as regular  
 35776 characters, are translated to themselves. However, any System V script that relied on *a-z*  
 35777 representing the three characters '-', ',' and 'z' have to be rewritten as *az-*.
- 35778 A prior version of IEEE Std 1003.1-200x had a *-c* option that behaved similarly to the *-C* option,  
 35779 but did not supply functionality equivalent to the *-c* option specified in IEEE Std 1003.1-200x.  
 35780 This meant that historical practice of being able to specify `tr -d\200-\377` (which would delete  
 35781 all bytes with the top bit set) would have no effect because, in the C locale, bytes with the values  
 35782 octal 200 to octal 377 are not characters.
- 35783 The earlier version also said that octal sequences referred to collating elements and could be  
 35784 placed adjacent to each other to specify multi-byte characters. However, it was noted that this  
 35785 caused ambiguities because *tr* would not be able to tell whether adjacent octal sequences were  
 35786 intending to specify multi-byte characters or multiple single byte characters.  
 35787 IEEE Std 1003.1-200x specifies that octal sequences always refer to single byte binary values.
- 35788 **FUTURE DIRECTIONS**
- 35789 None.
- 35790 **SEE ALSO**
- 35791 *sed*
- 35792 **CHANGE HISTORY**
- 35793 First released in Issue 2.
- 35794 **Issue 6**
- 35795 The *-C* operand is added, and the description of the *-c* operand is changed to align with the  
 35796 IEEE P1003.2b draft standard.
- 35797 The normative text is reworded to avoid use of the term “must” for application requirements.

35798 **NAME**

35799 true — return true value

35800 **SYNOPSIS**

35801 true

35802 **DESCRIPTION**35803 The *true* utility shall return with exit code zero.35804 **OPTIONS**

35805 None.

35806 **OPERANDS**

35807 None.

35808 **STDIN**

35809 Not used.

35810 **INPUT FILES**

35811 None.

35812 **ENVIRONMENT VARIABLES**

35813 None.

35814 **ASYNCHRONOUS EVENTS**

35815 Default.

35816 **STDOUT**

35817 Not used.

35818 **STDERR**

35819 None.

35820 **OUTPUT FILES**

35821 None.

35822 **EXTENDED DESCRIPTION**

35823 None.

35824 **EXIT STATUS**

35825 Default.

35826 **CONSEQUENCES OF ERRORS**

35827 None.

35828 **APPLICATION USAGE**35829 This utility is typically used in shell scripts, as shown in the **EXAMPLES** section. The special  
35830 built-in utility `:` is sometimes more efficient than *true*.35831 **EXAMPLES**

35832 This command is executed forever:

35833 while true  
35834 do  
35835     command  
35836 done

35837 **RATIONALE**

35838           The *true* utility has been retained in this volume of IEEE Std 1003.1-200x, even though the shell  
35839           special built-in : provides similar functionality, because *true* is widely used in historical scripts  
35840           and is less cryptic to novice script readers.

35841 **FUTURE DIRECTIONS**

35842           None.

35843 **SEE ALSO**

35844           *false*, Section 2.9 (on page 2248)

35845 **CHANGE HISTORY**

35846           First released in Issue 2.



35888 **STDERR**

35889 The standard error shall be used only for diagnostic messages.

35890 **OUTPUT FILES**

35891 None.

35892 **EXTENDED DESCRIPTION**

35893 None.

35894 **EXIT STATUS**

35895 The following exit values shall be returned:

35896 0 Successful completion.

35897 >0 An error occurred.

35898 **CONSEQUENCES OF ERRORS**

35899 Default.

35900 **APPLICATION USAGE**

35901 The *LC\_COLLATE* variable need not affect the actions of *tsort*. The output ordering is not  
35902 lexicographic, but depends on the pairs of items given as input.

35903 **EXAMPLES**

35904 The command:

35905 `tsort <<EOF`

35906 `a b c c d e`

35907 `g g`

35908 `f g e f`

35909 `h h`

35910 `EOF`

35911 produces the output:

35912 **a**

35913 **b**

35914 **c**

35915 **d**

35916 **e**

35917 **f**

35918 **g**

35919 **h**

35920 **RATIONALE**

35921 None.

35922 **FUTURE DIRECTIONS**

35923 None.

35924 **SEE ALSO**

35925 None.

35926 **CHANGE HISTORY**

35927 First released in Issue 2.

35928 **Issue 6**

35929 The normative text is reworded to avoid use of the term “must” for application requirements.

35930 **NAME**

35931 tty — return user's terminal name

35932 **SYNOPSIS**

35933 tty

35934 **DESCRIPTION**

35935 The *tty* utility shall write to the standard output the name of the terminal that is open as  
 35936 standard input. The name that is used shall be equivalent to the string that would be returned by  
 35937 the *ttyname()* function defined in the System Interfaces volume of IEEE Std 1003.1-200x.

35938 **OPTIONS**

35939 The *tty* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2,  
 35940 Utility Syntax Guidelines.

35941 **OPERANDS**

35942 None.

35943 **STDIN**

35944 While no input is read from standard input, standard input shall be examined to determine  
 35945 whether or not it is a terminal, and, if so, to determine the name of the terminal.

35946 **INPUT FILES**

35947 None.

35948 **ENVIRONMENT VARIABLES**35949 The following environment variables shall affect the execution of *tty*:

35950 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 35951 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
 35952 Internationalization Variables for the precedence of internationalization variables  
 35953 used to determine the values of locale categories.)

35954 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 35955 internationalization variables.

35956 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 35957 characters (for example, single-byte as opposed to multi-byte characters in  
 35958 arguments).

35959 *LC\_MESSAGES*

35960 Determine the locale that should be used to affect the format and contents of  
 35961 diagnostic messages written to standard error and informative messages written to  
 35962 standard output.

35963 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

35964 **ASYNCHRONOUS EVENTS**

35965 Default.

35966 **STDOUT**

35967 If standard input is a terminal device, a pathname of the terminal as specified by the *ttyname()*  
 35968 function defined in the System Interfaces volume of IEEE Std 1003.1-200x shall be written in the  
 35969 following format:

35970 "%s\n", &lt;terminal name&gt;

35971 Otherwise, a message shall be written indicating that standard input is not connected to a  
 35972 terminal. In the POSIX locale, the *tty* utility shall use the format:

35973 "not a tty\n"

35974 **STDERR**

35975 The standard error shall be used only for diagnostic messages. |

35976 **OUTPUT FILES**

35977 None.

35978 **EXTENDED DESCRIPTION**

35979 None.

35980 **EXIT STATUS**

35981 The following exit values shall be returned:

35982 0 Standard input is a terminal.

35983 1 Standard input is not a terminal.

35984 >1 An error occurred.

35985 **CONSEQUENCES OF ERRORS**

35986 Default.

35987 **APPLICATION USAGE**

35988 This utility checks the status of the file open as standard input against that of a implementation- |  
35989 defined set of files. It is possible that no match can be found, or that the match found need not be |  
35990 the same file as that which was opened for standard input (although they are the same device). |

35991 **EXAMPLES**

35992 None.

35993 **RATIONALE**

35994 None.

35995 **FUTURE DIRECTIONS**

35996 None.

35997 **SEE ALSO**

35998 The System Interfaces volume of IEEE Std 1003.1-200x, *isatty()*, *ttyname()*

35999 **CHANGE HISTORY**

36000 First released in Issue 2.

36001 **Issue 5**

36002 The SYNOPSIS is changed to indicate two forms of the command, with the second form marked  
36003 as obsolete. This is a clarification and does not change the functionality published in previous  
36004 issues.

36005 **Issue 6**

36006 The obsolescent **-s** option is removed.

36007 **NAME**

36008            type — write a description of command type

36009 **SYNOPSIS**

36010 xSI        type name . . .

36011

36012 **DESCRIPTION**

36013            The *type* utility shall indicate how each argument would be interpreted if used as a command  
36014            name.

36015 **OPTIONS**

36016            None.

36017 **OPERANDS**

36018            The following operand shall be supported:

36019            *name*            A name to be interpreted.

36020 **STDIN**

36021            Not used.

36022 **INPUT FILES**

36023            None.

36024 **ENVIRONMENT VARIABLES**

36025            The following environment variables shall affect the execution of *type*:

36026            *LANG*            Provide a default value for the internationalization variables that are unset or null.  
36027                                (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
36028                                Internationalization Variables for the precedence of internationalization variables  
36029                                used to determine the values of locale categories.)

36030            *LC\_ALL*        If set to a non-empty string value, override the values of all the other  
36031                                internationalization variables.

36032            *LC\_CTYPE*       Determine the locale for the interpretation of sequences of bytes of text data as  
36033                                characters (for example, single-byte as opposed to multi-byte characters in  
36034                                arguments).

36035            *LC\_MESSAGES*

36036                                Determine the locale that should be used to affect the format and contents of  
36037                                diagnostic messages written to standard error.

36038            *NLSPATH*        Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

36039            *PATH*            Determine the location of *name*, as described in the Base Definitions volume of  
36040                                IEEE Std 1003.1-200x, Chapter 8, Environment Variables.

36041 **ASYNCHRONOUS EVENTS**

36042            Default.

36043 **STDOUT**

36044            The standard output of *type* contains information about each operand in an unspecified format.  
36045            The information provided typically identifies the operand as a shell built-in, function, alias, or  
36046            keyword, and where applicable, may display the operand's pathname.

36047 **STDERR**

36048           The standard error shall be used only for diagnostic messages.

36049 **OUTPUT FILES**

36050           None.

36051 **EXTENDED DESCRIPTION**

36052           None.

36053 **EXIT STATUS**

36054           The following exit values shall be returned:

36055           0   Successful completion.

36056           >0  An error occurred.

36057 **CONSEQUENCES OF ERRORS**

36058           Default.

36059 **APPLICATION USAGE**

36060           Since *type* must be aware of the contents of the current shell execution environment (such as the lists of commands, functions, and built-ins processed by *hash*), it is always provided as a shell regular built-in. If it is called in a separate utility execution environment, such as one of the following:

36064           nohup type writer

36065           find . -type f | xargs type

36066           it might not produce accurate results.

36067 **EXAMPLES**

36068           None.

36069 **RATIONALE**

36070           None.

36071 **FUTURE DIRECTIONS**

36072           None.

36073 **SEE ALSO**

36074           *command*

36075 **CHANGE HISTORY**

36076           First released in Issue 2.

36077 **NAME**

36078           ulimit — set or report file size limit

36079 **SYNOPSIS**

36080 XSI        ulimit [-f][*blocks*]

36081

36082 **DESCRIPTION**

36083           The *ulimit* utility shall set or report the file-size writing limit imposed on files written by the  
36084 shell and its child processes (files of any size may be read). Only a process with appropriate  
36085 privileges can increase the limit.

36086 **OPTIONS**

36087           The *ulimit* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section  
36088 12.2, Utility Syntax Guidelines.

36089           The following option shall be supported:

36090        -f           Set (or report, if no *blocks* operand is present), the file size limit in blocks. The -f  
36091 option shall also be the default case.

36092 **OPERANDS**

36093           The following operand shall be supported:

36094        *blocks*        The number of 512-byte blocks to use as the new file size limit.

36095 **STDIN**

36096           Not used.

36097 **INPUT FILES**

36098           None.

36099 **ENVIRONMENT VARIABLES**

36100           The following environment variables shall affect the execution of *ulimit*:

36101        *LANG*        Provide a default value for the internationalization variables that are unset or null.  
36102 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
36103 Internationalization Variables for the precedence of internationalization variables  
36104 used to determine the values of locale categories.)

36105        *LC\_ALL*       If set to a non-empty string value, override the values of all the other  
36106 internationalization variables.

36107        *LC\_CTYPE*     Determine the locale for the interpretation of sequences of bytes of text data as  
36108 characters (for example, single-byte as opposed to multi-byte characters in  
36109 arguments).

36110        *LC\_MESSAGES*

36111           Determine the locale that should be used to affect the format and contents of  
36112 diagnostic messages written to standard error.

36113        *NLSPATH*     Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

36114 **ASYNCHRONOUS EVENTS**

36115           Default.

36116 **STDOUT**

36117           The standard output shall be used when no *blocks* operand is present. If the current number of  
36118 blocks is limited, the number of blocks in the current limit shall be written in the following  
36119 format:

- 36120 "%d\n", <number of 512-byte blocks>
- 36121 If there is no current limit on the number of blocks, in the POSIX locale the following format  
36122 shall be used:
- 36123 "unlimited\n"
- 36124 **STDERR**
- 36125 The standard error shall be used only for diagnostic messages.
- 36126 **OUTPUT FILES**
- 36127 None.
- 36128 **EXTENDED DESCRIPTION**
- 36129 None.
- 36130 **EXIT STATUS**
- 36131 The following exit values shall be returned:
- 36132 0 Successful completion.
- 36133 >0 A request for a higher limit was rejected or an error occurred.
- 36134 **CONSEQUENCES OF ERRORS**
- 36135 Default.
- 36136 **APPLICATION USAGE**
- 36137 Since *ulimit* affects the current shell execution environment, it is always provided as a shell  
36138 regular built-in. If it is called in separate utility execution environment, such as one of the  
36139 following:
- 36140 nohup ulimit -f 10000  
36141 env ulimit 10000
- 36142 it does not affect the file size limit of the caller's environment.
- 36143 Once a limit has been decreased by a process, it cannot be increased (unless appropriate  
36144 privileges are involved), even back to the original system limit.
- 36145 **EXAMPLES**
- 36146 Set the file size limit to 51 200 bytes:
- 36147 ulimit -f 100
- 36148 **RATIONALE**
- 36149 None.
- 36150 **FUTURE DIRECTIONS**
- 36151 None.
- 36152 **SEE ALSO**
- 36153 The System Interfaces volume of IEEE Std 1003.1-200x, *ulimit()*
- 36154 **CHANGE HISTORY**
- 36155 First released in Issue 2.

36156 **NAME**

36157           umask — get or set the file mode creation mask

36158 **SYNOPSIS**36159           umask [-S][*mask*]36160 **DESCRIPTION**

36161           The *umask* utility shall set the file mode creation mask of the current shell execution environment (see Section 2.12 (on page 2263)) to the value specified by the *mask* operand. This mask shall affect the initial value of the file permission bits of subsequently created files. If *umask* is called in a subshell or separate utility execution environment, such as one of the following:

```
36165           (umask 002)
36166           nohup umask ...
36167           find . -exec umask ... \;
```

36168           it shall not affect the file mode creation mask of the caller's environment.

36169           If the *mask* operand is not specified, the *umask* utility shall write to standard output the value of the invoking process's file mode creation mask.

36171 **OPTIONS**

36172           The *umask* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2, Utility Syntax Guidelines.

36173           The following option shall be supported:

36174           **-S**           Produce symbolic output.

36175           The default output style is unspecified, but shall be recognized on a subsequent invocation of *umask* on the same system as a *mask* operand to restore the previous file mode creation mask.

36178 **OPERANDS**

36179           The following operand shall be supported:

36180           ***mask***           A string specifying the new file mode creation mask. The string is treated in the same way as the *mode* operand described in the EXTENDED DESCRIPTION section for *chmod*.

36181           For a *symbolic\_mode* value, the new value of the file mode creation mask shall be the logical complement of the file permission bits portion of the file mode specified by the *symbolic\_mode* string.

36182           In a *symbolic\_mode* value, the permissions *op* characters '+' and '-' shall be interpreted relative to the current file mode creation mask; '+' shall cause the bits for the indicated permissions to be cleared in the mask; '-' shall cause the bits for the indicated permissions to be set in the mask.

36183           The interpretation of *mode* values that specify file mode bits other than the file permission bits is unspecified.

36184           In the octal integer form of *mode*, the specified bits are set in the file mode creation mask.

36185           The file mode creation mask shall be set to the resulting numeric value.

36186           The default output of a prior invocation of *umask* on the same system with no operand also shall be recognized as a *mask* operand.

36197 **STDIN**

36198 Not used.

36199 **INPUT FILES**

36200 None.

36201 **ENVIRONMENT VARIABLES**36202 The following environment variables shall affect the execution of *umask*:

36203 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 36204 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
 36205 Internationalization Variables for the precedence of internationalization variables  
 36206 used to determine the values of locale categories.)

36207 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 36208 internationalization variables.

36209 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 36210 characters (for example, single-byte as opposed to multi-byte characters in  
 36211 arguments).

36212 *LC\_MESSAGES*

36213 Determine the locale that should be used to affect the format and contents of  
 36214 diagnostic messages written to standard error.

36215 *NSLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

36216 **ASYNCHRONOUS EVENTS**

36217 Default.

36218 **STDOUT**

36219 When the *mask* operand is not specified, the *umask* utility shall write a message to standard  
 36220 output that can later be used as a *umask mask* operand.

36221 If *-S* is specified, the message shall be in the following format:

36222 "u=%s,g=%s,o=%s\n", <owner permissions>, <group permissions>,  
 36223 <other permissions>

36224 where the three values shall be combinations of letters from the set {r, w, x}; the presence of a  
 36225 letter shall indicate that the corresponding bit is clear in the file mode creation mask.

36226 If a *mask* operand is specified, there shall be no output written to standard output.

36227 **STDERR**

36228 The standard error shall be used only for diagnostic messages.

36229 **OUTPUT FILES**

36230 None.

36231 **EXTENDED DESCRIPTION**

36232 None.

36233 **EXIT STATUS**

36234 The following exit values shall be returned:

36235 0 The file mode creation mask was successfully changed, or no *mask* operand was supplied.

36236 >0 An error occurred.

## 36237 CONSEQUENCES OF ERRORS

36238 Default.

## 36239 APPLICATION USAGE

36240 Since *umask* affects the current shell execution environment, it is generally provided as a shell  
 36241 regular built-in.

36242 In contrast to the negative permission logic provided by the file mode creation mask and the  
 36243 octal number form of the *mask* argument, the symbolic form of the *mask* argument specifies those  
 36244 permissions that are left alone.

## 36245 EXAMPLES

36246 Either of the commands:

36247 `umask a=rx,ug+w`36248 `umask 002`36249 sets the mode mask so that subsequently created files have their `S_IWOTH` bit cleared.

36250 After setting the mode mask with either of the above commands, the *umask* command can be  
 36251 used to write out the current value of the mode mask:

36252 `$ umask`36253 `0002`

36254 (The output format is unspecified, but historical implementations use the octal integer mode  
 36255 format.)

36256 `$ umask -S`36257 `u=rwx,g=rwx,o=rx`

36258 Either of these outputs can be used as the mask operand to a subsequent invocation of the *umask*  
 36259 utility.

36260 Assuming the mode mask is set as above, the command:

36261 `umask g-w`

36262 sets the mode mask so that subsequently created files have their `S_IWGRP` and `S_IWOTH` bits  
 36263 cleared.

36264 The command:

36265 `umask -- -w`

36266 sets the mode mask so that subsequently created files have all their write bits cleared. Note that  
 36267 *mask* operands `-r`, `-w`, `-x` or anything beginning with a hyphen, must be preceded by `--` to  
 36268 keep it from being interpreted as an option.

## 36269 RATIONALE

36270 Since *umask* affects the current shell execution environment, it is generally provided as a shell  
 36271 regular built-in. If it is called in a subshell or separate utility execution environment, such as one  
 36272 of the following:

36273 `(umask 002)`36274 `nohup umask ...`36275 `find . -exec umask ... \;`

36276 it does not affect the file mode creation mask of the environment of the caller.

36277 The description of the historical utility was modified to allow it to use the symbolic modes of  
 36278 *chmod*. The `-s` option used in early proposals was changed to `-S` because `-s` could be confused

- 36279 with a *symbolic\_mode* form of mask referring to the S\_ISUID and S\_ISGID bits.
- 36280 The default output style is implementation-defined to permit implementors to provide  
36281 migration to the new symbolic style at the time most appropriate to their users. An `-o` flag to  
36282 force octal mode output was omitted because the octal mode may not be sufficient to specify all  
36283 of the information that may be present in the file mode creation mask when more secure file  
36284 access permission checks are implemented.
- 36285 It has been suggested that trusted systems developers might appreciate ameliorating the  
36286 requirement that the mode mask “affects” the file access permissions, since it seems access  
36287 control lists might replace the mode mask to some degree. The wording has been changed to say  
36288 that it affects the file permission bits, and it leaves the details of the behavior of how they affect  
36289 the file access permissions to the description in the System Interfaces volume of  
36290 IEEE Std 1003.1-200x.
- 36291 **FUTURE DIRECTIONS**
- 36292 None.
- 36293 **SEE ALSO**
- 36294 *chmod*, the System Interfaces volume of IEEE Std 1003.1-200x, *umask()*
- 36295 **CHANGE HISTORY**
- 36296 First released in Issue 2.
- 36297 **Issue 6**
- 36298 The following new requirements on POSIX implementations derive from alignment with the  
36299 Single UNIX Specification:
- 36300
- The octal mode is supported.

36301 **NAME**

36302 unalias — remove alias definitions

36303 **SYNOPSIS**36304 UP unalias *alias-name*...

36305 unalias -a

36306

36307 **DESCRIPTION**

36308 The *unalias* utility shall remove the definition for each alias name specified. See Section 2.3.1 (on  
 36309 page 2234). The aliases shall be removed from the current shell execution environment; see  
 36310 Section 2.12 (on page 2263).

36311 **OPTIONS**

36312 The *unalias* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section  
 36313 12.2, Utility Syntax Guidelines.

36314 The following option shall be supported:

36315 -a Remove all alias definitions from the current shell execution environment.

36316 **OPERANDS**

36317 The following operand shall be supported:

36318 *alias-name* The name of an alias to be removed.36319 **STDIN**

36320 Not used.

36321 **INPUT FILES**

36322 None.

36323 **ENVIRONMENT VARIABLES**36324 The following environment variables shall affect the execution of *unalias*:

36325 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 36326 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
 36327 Internationalization Variables for the precedence of internationalization variables  
 36328 used to determine the values of locale categories.)

36329 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 36330 internationalization variables.

36331 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 36332 characters (for example, single-byte as opposed to multi-byte characters in  
 36333 arguments).

36334 *LC\_MESSAGES*

36335 Determine the locale that should be used to affect the format and contents of  
 36336 diagnostic messages written to standard error.

36337 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.36338 **ASYNCHRONOUS EVENTS**

36339 Default.

36340 **STDOUT**

36341 Not used.

36342 **STDERR**

36343 The standard error shall be used only for diagnostic messages.

36344 **OUTPUT FILES**

36345 None.

36346 **EXTENDED DESCRIPTION**

36347 None.

36348 **EXIT STATUS**

36349 The following exit values shall be returned:

36350 0 Successful completion.

36351 >0 One of the *alias-name* operands specified did not represent a valid alias definition, or an  
36352 error occurred.

36353 **CONSEQUENCES OF ERRORS**

36354 Default.

36355 **APPLICATION USAGE**

36356 Since *unalias* affects the current shell execution environment, it is generally provided as a shell  
36357 regular built-in.

36358 **EXAMPLES**

36359 None.

36360 **RATIONALE**

36361 The *unalias* description is based on that from historical KornShell implementations. Known  
36362 differences exist between that and the C shell. The KornShell version was adopted to be  
36363 consistent with all the other KornShell features in this volume of IEEE Std 1003.1-200x, such as  
36364 command line editing.

36365 The *-a* option is the equivalent of the *unalias \** form of the C shell and is provided to address  
36366 security concerns about unknown aliases entering the environment of a user (or application)  
36367 through the allowable implementation-defined predefined alias route or as a result of an *ENV*  
36368 file. (Although *unalias* could be used to simplify the “secure” shell script shown in the *command*  
36369 rationale, it does not obviate the need to quote all command names. An initial call to *unalias -a*  
36370 would have to be quoted in case there was an alias for *unalias*.)

36371 **FUTURE DIRECTIONS**

36372 None.

36373 **SEE ALSO**

36374 *alias*

36375 **CHANGE HISTORY**

36376 First released in Issue 4.

36377 **Issue 6**

36378 This utility is now marked as part of the User Portability Utilities option.

36379 **NAME**

36380            **uname** — return system name

36381 **SYNOPSIS**

36382            **uname** [-snrvma]

36383 **DESCRIPTION**

36384            By default, the *uname* utility shall write the operating system name to standard output. When  
36385            options are specified, symbols representing one or more system characteristics shall be written  
36386            to the standard output. The format and contents of the symbols are implementation-defined. On  
36387            systems conforming to the System Interfaces volume of IEEE Std 1003.1-200x, the symbols  
36388            written shall be those supported by the *uname()* function as defined in the System Interfaces  
36389            volume of IEEE Std 1003.1-200x.

36390 **OPTIONS**

36391            The *uname* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section  
36392            12.2, Utility Syntax Guidelines.

36393            The following options shall be supported:

- 36394            **-a**            Behave as though all of the options **-mnrsv** were specified.
- 36395            **-m**            Write the name of the hardware type on which the system is running to standard  
36396            output.
- 36397            **-n**            Write the name of this node within an implementation-defined communications  
36398            network.
- 36399            **-r**            Write the current release level of the operating system implementation.
- 36400            **-s**            Write the name of the implementation of the operating system.
- 36401            **-v**            Write the current version level of this release of the operating system  
36402            implementation.

36403            If no options are specified, the *uname* utility shall write the operating system name, as if the **-s**  
36404            option had been specified.

36405 **OPERANDS**

36406            None.

36407 **STDIN**

36408            Not used.

36409 **INPUT FILES**

36410            None.

36411 **ENVIRONMENT VARIABLES**

36412            The following environment variables shall affect the execution of *uname*:

- 36413            **LANG**        Provide a default value for the internationalization variables that are unset or null.  
36414            (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
36415            Internationalization Variables for the precedence of internationalization variables  
36416            used to determine the values of locale categories.)
- 36417            **LC\_ALL**       If set to a non-empty string value, override the values of all the other  
36418            internationalization variables.
- 36419            **LC\_CTYPE**     Determine the locale for the interpretation of sequences of bytes of text data as  
36420            characters (for example, single-byte as opposed to multi-byte characters in  
36421            arguments).

- 36422 **LC\_MESSAGES**  
36423 Determine the locale that should be used to affect the format and contents of  
36424 diagnostic messages written to standard error.
- 36425 XSI **NLSPATH** Determine the location of message catalogs for the processing of **LC\_MESSAGES**.
- 36426 **ASYNCHRONOUS EVENTS**  
36427 Default.
- 36428 **STDOUT**  
36429 By default, the output shall be a single line of the following form:  
36430 "%s\n", <sysname>  
36431 If the **-a** option is specified, the output shall be a single line of the following form:  
36432 "%s %s %s %s %s\n", <sysname>, <nodename>, <release>,  
36433 <version>, <machine>  
36434 Additional implementation-defined symbols may be written; all such symbols shall be written at  
36435 the end of the line of output before the <newline>.  
36436 If options are specified to select different combinations of the symbols, only those symbols shall  
36437 be written, in the order shown above for the **-a** option. If a symbol is not selected for writing, its  
36438 corresponding trailing <blank>s also shall not be written.
- 36439 **STDERR**  
36440 The standard error shall be used only for diagnostic messages.
- 36441 **OUTPUT FILES**  
36442 None.
- 36443 **EXTENDED DESCRIPTION**  
36444 None.
- 36445 **EXIT STATUS**  
36446 The following exit values shall be returned:  
36447 0 The requested information was successfully written.  
36448 >0 An error occurred.
- 36449 **CONSEQUENCES OF ERRORS**  
36450 Default.
- 36451 **APPLICATION USAGE**  
36452 Note that any of the symbols could include embedded <space>s, which may affect parsing  
36453 algorithms if multiple options are selected for output.  
36454 The node name is typically a name that the system uses to identify itself for intersystem  
36455 communication addressing.
- 36456 **EXAMPLES**  
36457 The following command:  
36458 `uname -sr`  
36459 writes the operating system name and release level, separated by one or more <blank>s.

**36460 RATIONALE**

36461 It was suggested that this utility cannot be used portably since the format of the symbols is  
36462 implementation-defined. The POSIX.1 working group could not achieve consensus on defining  
36463 these formats in the underlying *uname()* function, and there was no expectation that this volume  
36464 of IEEE Std 1003.1-200x would be any more successful. Some applications may still find this  
36465 historical utility of value. For example, the symbols could be used for system log entries or for  
36466 comparison with operator or user input.

**36467 FUTURE DIRECTIONS**

36468 None.

**36469 SEE ALSO**

36470 The System Interfaces volume of IEEE Std 1003.1-200x, *uname()*

**36471 CHANGE HISTORY**

36472 First released in Issue 2.

36473 **NAME**

36474 uncompress — expand compressed data

36475 **SYNOPSIS**36476 xSI uncompress [-cfv][*file...*]

36477

36478 **DESCRIPTION**

36479 The *uncompress* utility shall restore files to their original state after they have been compressed  
 36480 using the *compress* utility. If no files are specified, the standard input shall be uncompressed to  
 36481 the standard output. If the invoking process has appropriate privileges, the ownership, modes,  
 36482 access time, and modification time of the original file shall be preserved.

36483 This utility shall support the uncompressing of any files produced by the *compress* utility on the  
 36484 same implementation. For files produced by *compress* on other systems, *uncompress* supports 9 to  
 36485 14-bit compression (see *compress* (on page 2465), **-b**); it is implementation-defined whether  
 36486 values of **-b** greater than 14 are supported.

36487 **OPTIONS**

36488 The *uncompress* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x,  
 36489 Section 12.2, Utility Syntax Guidelines.

36490 The following options shall be supported:

- 36491 **-c** Write to standard output; no files are changed.
- 36492 **-f** Do not prompt for overwriting files. Except when run in the background, if **-f** is  
 36493 not given the user shall be prompted as to whether an existing file should be  
 36494 overwritten. If the standard input is not a terminal and **-f** is not given, *uncompress*  
 36495 shall write a diagnostic message to standard error and exit with a status greater  
 36496 than zero.
- 36497 **-v** Write messages to standard error concerning the expansion of each file.

36498 **OPERANDS**

36499 The following operand shall be supported:

- 36500 *file* A pathname of a file. If *file* already has the **.Z** suffix specified, it shall be used as the  
 36501 input file and the output file shall be named **file** with the **.Z** suffix removed.  
 36502 Otherwise, *file* shall be used as the name of the output file and **file** with the **.Z**  
 36503 suffix appended shall be used as the input file.

36504 **STDIN**

36505 The standard input shall be used only if no *file* operands are specified, or if a *file* operand is '-'.

36506 **INPUT FILES**

36507 Input files shall be in the format produced by the *compress* utility.

36508 **ENVIRONMENT VARIABLES**

36509 The following environment variables shall affect the execution of *uncompress*:

- 36510 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 36511 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
 36512 Internationalization Variables for the precedence of internationalization variables  
 36513 used to determine the values of locale categories.)
- 36514 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 36515 internationalization variables.

- 36516            *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
36517 characters (for example, single-byte as opposed to multi-byte characters in  
36518 arguments).
- 36519            *LC\_MESSAGES*  
36520                    Determine the locale that should be used to affect the format and contents of  
36521 diagnostic messages written to standard error.
- 36522            *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 36523 **ASYNCHRONOUS EVENTS**
- 36524            Default.
- 36525 **STDOUT**
- 36526            When there are no *file* operands or the *-c* option is specified, the uncompressed output is written  
36527 to standard output.
- 36528 **STDERR**
- 36529            Prompts shall be written to the standard error output under the conditions specified in the  
36530 DESCRIPTION and OPTIONS sections. The prompts shall contain the *file* pathname, but their  
36531 format is otherwise unspecified. Otherwise, the standard error output shall be used only for  
36532 diagnostic messages.
- 36533 **OUTPUT FILES**
- 36534            Output files are the same as the respective input files to *compress*.
- 36535 **EXTENDED DESCRIPTION**
- 36536            None.
- 36537 **EXIT STATUS**
- 36538            The following exit values shall be returned:
- 36539            0 Successful completion.
- 36540            >0 An error occurred.
- 36541 **CONSEQUENCES OF ERRORS**
- 36542            The input file remains unmodified.
- 36543 **APPLICATION USAGE**
- 36544            The limit of 14 on the *compress -b bits* argument is to achieve portability to all systems (within  
36545 the restrictions imposed by the lack of an explicit published file format). Some implementations |  
36546 based on 16-bit architectures cannot support 15 or 16-bit uncompression. |
- 36547 **EXAMPLES**
- 36548            None.
- 36549 **RATIONALE**
- 36550            None.
- 36551 **FUTURE DIRECTIONS**
- 36552            None.
- 36553 **SEE ALSO**
- 36554            *compress, zcat*
- 36555 **CHANGE HISTORY**
- 36556            First released in Issue 4.

36557 **Issue 6**

36558

The normative text is reworded to avoid use of the term “must” for application requirements.

36559 **NAME**

36560 unexpand — convert spaces to tabs

36561 **SYNOPSIS**36562 UP unexpand [ *-a* | *-t* *tablist* ][*file...*]

36563

36564 **DESCRIPTION**

36565 The *unexpand* utility shall copy files or standard input to standard output, converting <blank>s  
 36566 at the beginning of each line into the maximum number of <tab>s followed by the minimum  
 36567 number of <space>s needed to fill the same column positions originally filled by the translated  
 36568 <blank>s. By default, tabstops shall be set at every eighth column position. Each <backspace>  
 36569 shall be copied to the output, and shall cause the column position count for tab calculations to be  
 36570 decremented; the count shall never be decremented to a value less than one.

36571 **OPTIONS**

36572 The *unexpand* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x,  
 36573 Section 12.2, Utility Syntax Guidelines.

36574 The following options shall be supported:

36575 **-a** In addition to translating <blank>s at the beginning of each line, translate all  
 36576 sequences of two or more <blank>s immediately preceding a tab stop to the  
 36577 maximum number of <tab>s followed by the minimum number of <space>s  
 36578 needed to fill the same column positions originally filled by the translated  
 36579 <blank>s.

36580 **-t *tablist*** Specify the tab stops. The application shall ensure that the *tablist* option-argument  
 36581 is a single argument consisting of a single positive decimal integer or multiple  
 36582 positive decimal integers, separated by <blank>s or commas, in ascending order. If  
 36583 a single number is given, tabs shall be set *tablist* column positions apart instead of  
 36584 the default 8. If multiple numbers are given, the tabs shall be set at those specific  
 36585 column positions.

36586 The application shall ensure that each tab-stop position *N* is an integer value  
 36587 greater than zero, and the list shall be in strictly ascending order. This is taken to  
 36588 mean that, from the start of a line of output, tabbing to position *N* shall cause the  
 36589 next character output to be in the (*N*+1)th column position on that line. When the  
 36590 **-t** option is not specified, the default shall be the equivalent of specifying **-t 8**  
 36591 (except for the interaction with **-a**, described below).

36592 No <space>-to-<tab> conversions shall occur for characters at positions beyond  
 36593 the last of those specified in a multiple tab-stop list.

36594 When **-t** is specified, the presence or absence of the **-a** option shall be ignored;  
 36595 conversion shall not be limited to the processing of leading <blank>s.

36596 **OPERANDS**

36597 The following operand shall be supported:

36598 *file* A pathname of a text file to be used as input.

36599 **STDIN**

36600 See the INPUT FILES section.

36601 **INPUT FILES**

36602 The input files shall be text files.

36603 **ENVIRONMENT VARIABLES**36604 The following environment variables shall affect the execution of *unexpand*:

36605 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 36606 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
 36607 Internationalization Variables for the precedence of internationalization variables  
 36608 used to determine the values of locale categories.)

36609 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 36610 internationalization variables.

36611 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 36612 characters (for example, single-byte as opposed to multi-byte characters in  
 36613 arguments and input files), the processing of <tab>s and <space>s and for the  
 36614 determination of the width in column positions each character would occupy on  
 36615 an output device.

36616 *LC\_MESSAGES*

36617 Determine the locale that should be used to affect the format and contents of  
 36618 diagnostic messages written to standard error.

36619 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

36620 **ASYNCHRONOUS EVENTS**

36621 Default.

36622 **STDOUT**

36623 The standard output shall be equivalent to the input files with the specified <space>-to-<tab> |  
 36624 conversions.

36625 **STDERR**

36626 The standard error shall be used only for diagnostic messages. |

36627 **OUTPUT FILES**

36628 None.

36629 **EXTENDED DESCRIPTION**

36630 None.

36631 **EXIT STATUS**

36632 The following exit values shall be returned:

36633 0 Successful completion.

36634 &gt;0 An error occurred.

36635 **CONSEQUENCES OF ERRORS**

36636 Default.

**36637 APPLICATION USAGE**

36638 One non-intuitive aspect of *unexpand* is its restriction to leading spaces when neither **-a** nor **-t** is  
36639 specified. Users who desire to always convert all spaces in a file can easily alias *unexpand* to use  
36640 the **-a** or **-t 8** option.

**36641 EXAMPLES**

36642 None.

**36643 RATIONALE**

36644 On several occasions, consideration was given to adding a **-t** option to the *unexpand* utility to  
36645 complement the **-t** in *expand* (see *expand* (on page 2627)). The historical intent of *unexpand* was  
36646 to translate multiple <blank>s into tab stops, where tab stops were a multiple of eight column  
36647 positions on most UNIX systems. An early proposal omitted **-t** because it seemed outside the  
36648 scope of the UPE; it was not described in any of the base documents. However, hard-coding tab  
36649 stops every eight columns was not suitable for the international community and broke historical  
36650 precedents for some vendors in the FORTRAN community, so **-t** was restored in conjunction  
36651 with the list of valid extension categories considered by the standard developers. Thus, *unexpand*  
36652 is now the logical converse of *expand*.

**36653 FUTURE DIRECTIONS**

36654 None.

**36655 SEE ALSO**

36656 *expand*, *tabs*

**36657 CHANGE HISTORY**

36658 First released in Issue 4.

**36659 Issue 6**

36660 This utility is now marked as part of the User Portability Utilities option.

36661 The definition of the *LC\_CTYPE* environment variable is changed to align with the  
36662 IEEE P1003.2b draft standard.

36663 The normative text is reworded to avoid use of the term “must” for application requirements.

36664 **NAME**36665 unget — undo a previous get of an SCCS file (**DEVELOPMENT**)36666 **SYNOPSIS**36667 xSI unget [-ns][-r *SID*] *file...*

36668

36669 **DESCRIPTION**36670 The *unget* utility shall reverse the effect of a *get -e* done prior to creating the intended new delta.36671 **OPTIONS**36672 The *unget* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section  
36673 12.2, Utility Syntax Guidelines.

36674 The following options shall be supported:

36675 **-r** *SID* Uniquely identify which delta is no longer intended. (This would have been  
36676 specified by *get* as the new delta.) The use of this option is necessary only if two or  
36677 more outstanding *get* commands for editing on the same SCCS file were done by  
36678 the same person (login name).36679 **-s** Suppress the writing to standard output of the intended delta's SID.36680 **-n** Retain the file that was obtained by *get*, which would normally be removed from  
36681 the current directory.36682 **OPERANDS**

36683 The following operands shall be supported:

36684 **file** A pathname of an existing SCCS file or a directory. If *file* is a directory, the *unget*  
36685 utility shall behave as though each file in the directory were specified as a named  
36686 file, except that non-SCCS files (last component of the pathname does not begin  
36687 with *s.*) and unreadable files shall be silently ignored.36688 If exactly one *file* operand appears, and it is '-', the standard input shall be read;  
36689 each line of the standard input shall be taken to be the name of an SCCS file to be  
36690 processed. Non-SCCS files and unreadable files shall be silently ignored.36691 **STDIN**36692 The standard input shall be a text file used only when the *file* operand is specified as '-'. Each  
36693 line of the text file shall be interpreted as an SCCS pathname.36694 **INPUT FILES**

36695 Any SCCS files processed shall be files of an unspecified format.

36696 **ENVIRONMENT VARIABLES**36697 The following environment variables shall affect the execution of *unget*:36698 **LANG** Provide a default value for the internationalization variables that are unset or null.  
36699 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
36700 Internationalization Variables for the precedence of internationalization variables  
36701 used to determine the values of locale categories.)36702 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
36703 internationalization variables.36704 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
36705 characters (for example, single-byte as opposed to multi-byte characters in  
36706 arguments and input files).

- 36707 *LC\_MESSAGES*
- 36708 Determine the locale that should be used to affect the format and contents of  
36709 diagnostic messages written to standard error.
- 36710 *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 36711 **ASYNCHRONOUS EVENTS**
- 36712 Default.
- 36713 **STDOUT**
- 36714 The standard output shall consist of a line for each file, in the following format:
- 36715 "%s\n", <*SID removed from file*>
- 36716 If there is more than one named file or if a directory or standard input is named, each pathname  
36717 shall be written before each of the preceding lines:
- 36718 "\n%s:\n", <*pathname*>
- 36719 **STDERR**
- 36720 The standard error shall be used only for diagnostic messages.
- 36721 **OUTPUT FILES**
- 36722 Any SCCS files updated shall be files of an unspecified format. During processing of a *file*, a  
36723 locking *z-file*, as described in *get*, and a *q-file* (a working copy of the *p-file*), may be created and  
36724 deleted. The *p-file* and *g-file*, as described in *get*, shall be deleted.
- 36725 **EXTENDED DESCRIPTION**
- 36726 None.
- 36727 **EXIT STATUS**
- 36728 The following exit values shall be returned:
- 36729 0 Successful completion.
- 36730 >0 An error occurred.
- 36731 **CONSEQUENCES OF ERRORS**
- 36732 Default.
- 36733 **APPLICATION USAGE**
- 36734 None.
- 36735 **EXAMPLES**
- 36736 None.
- 36737 **RATIONALE**
- 36738 None.
- 36739 **FUTURE DIRECTIONS**
- 36740 None.
- 36741 **SEE ALSO**
- 36742 *delta, get, sact*
- 36743 **CHANGE HISTORY**
- 36744 First released in Issue 2.
- 36745 **Issue 6**
- 36746 The normative text is reworded to avoid use of the term “must” for application requirements.
- 36747 The normative text is reworded to emphasize the term “shall” for implementation requirements.

36748 **NAME**36749            **uniq** — report or filter out repeated lines in a file36750 **SYNOPSIS**36751            **uniq** [-c|-d|-u][-f *fields*][-s *char*][*input\_file* [*output\_file*]]36752 **DESCRIPTION**36753            The *uniq* utility shall read an input file comparing adjacent lines, and writes one copy of each  
36754            input line on the output. The second and succeeding copies of repeated adjacent input lines shall  
36755            not be written.

36756            Repeated lines in the input shall not be detected if they are not adjacent.

36757 **OPTIONS**36758            The *uniq* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section  
36759            12.2, Utility Syntax Guidelines.

36760            The following options shall be supported:

36761            **-c**            Precede each output line with a count of the number of times the line occurred in  
36762            the input.36763            **-d**            Suppress the writing of lines that are not repeated in the input.36764            **-f *fields***    Ignore the first *fields* fields on each input line when doing comparisons, where  
36765            *fields* is a positive decimal integer. A field is the maximal string matched by the  
36766            basic regular expression:36767            [[*:blank:*]]\*[^*:blank:*]]\*36768            If the *fields* option-argument specifies more fields than appear on an input line, a  
36769            null string shall be used for comparison.36770            **-s *chars***    Ignore the first *chars* characters when doing comparisons, where *chars* shall be a  
36771            positive decimal integer. If specified in conjunction with the **-f** option, the first  
36772            *chars* characters after the first *fields* fields shall be ignored. If the *chars* option-  
36773            argument specifies more characters than remain on an input line, a null string shall  
36774            be used for comparison.36775            **-u**            Suppress the writing of lines that are repeated in the input.36776 **OPERANDS**

36777            The following operands shall be supported:

36778            *input\_file*    A pathname of the input file. If the *input\_file* operand is not specified, or if the  
36779            *input\_file* is '-', the standard input shall be used.36780            *output\_file*   A pathname of the output file. If the *output\_file* operand is not specified, the  
36781            standard output shall be used. The results are unspecified if the file named by  
36782            *output\_file* is the file named by *input\_file*.36783 **STDIN**36784            The standard input shall be used only if no *input\_file* operand is specified or if *input\_file* is '-'.  
36785            See the INPUT FILES section.36786 **INPUT FILES**

36787            The input file shall be a text file.

36788 **ENVIRONMENT VARIABLES**

36789 The following environment variables shall affect the execution of *uniq*:

36790 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 36791 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
 36792 Internationalization Variables for the precedence of internationalization variables  
 36793 used to determine the values of locale categories.)

36794 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 36795 internationalization variables.

36796 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 36797 characters (for example, single-byte as opposed to multi-byte characters in  
 36798 arguments and input files) which characters constitute a <blank> in the current  
 36799 locale.

36800 *LC\_MESSAGES*

36801 Determine the locale that should be used to affect the format and contents of  
 36802 diagnostic messages written to standard error.

36803 *NSI* *NLS\_PATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

36804 **ASYNCHRONOUS EVENTS**

36805 Default.

36806 **STDOUT**

36807 The standard output shall be used only if no *output\_file* operand is specified. See the OUTPUT  
 36808 FILES section.

36809 **STDERR**

36810 The standard error shall be used only for diagnostic messages.

36811 **OUTPUT FILES**

36812 If the *-c* option is specified, the application shall ensure that the output file is empty or each line  
 36813 shall be of the form:

36814 "%d %s", <number of duplicates>, <line>

36815 otherwise, the application shall ensure that the output file is empty or each line shall be of the  
 36816 form:

36817 "%s", <line>

36818 **EXTENDED DESCRIPTION**

36819 None.

36820 **EXIT STATUS**

36821 The following exit values shall be returned:

36822 0 The utility executed successfully.

36823 >0 An error occurred.

36824 **CONSEQUENCES OF ERRORS**

36825 Default.

36826 **APPLICATION USAGE**

36827 The *sort* utility can be used to cause repeated lines to be adjacent in the input file.

36828 **EXAMPLES**

36829 The following input file data (but flushed left) was used for a test series on *uniq*:

```
36830 #01 foo0 bar0 fool bar1
36831 #02 bar0 fool bar1 fool
36832 #03 foo0 bar0 fool bar1
36833 #04
36834 #05 foo0 bar0 fool bar1
36835 #06 foo0 bar0 fool bar1
36836 #07 bar0 fool bar1 foo0
```

36837 What follows is a series of test invocations of the *uniq* utility that use a mixture of *uniq* options  
 36838 against the input file data. These tests verify the meaning of *adjacent*. The *uniq* utility views the  
 36839 input data as a sequence of strings delimited by '\n'. Accordingly, for the *fieldsth* member of  
 36840 the sequence, *uniq* interprets unique or repeated adjacent lines strictly relative to the *fields+1*th  
 36841 member.

36842 1. This first example tests the line counting option, comparing each line of the input file data  
 36843 starting from the second field:

```
36844 uniq -c -f 1 uniq_0I.t
36845     1 #01 foo0 bar0 fool bar1
36846     1 #02 bar0 fool bar1 foo0
36847     1 #03 foo0 bar0 fool bar1
36848     1 #04
36849     2 #05 foo0 bar0 fool bar1
36850     1 #07 bar0 fool bar1 foo0
```

36851 The number '2', prefixing the fifth line of output, signifies that the *uniq* utility detected a  
 36852 pair of repeated lines. Given the input data, this can only be true when *uniq* is run using  
 36853 the *-f 1* option (which shall cause *uniq* to ignore the first field on each input line).

36854 2. The second example tests the option to suppress unique lines, comparing each line of the  
 36855 input file data starting from the second field:

```
36856 uniq -d -f 1 uniq_0I.t
36857 #05 foo0 bar0 fool bar1
```

36858 3. This test suppresses repeated lines, comparing each line of the input file data starting from  
 36859 the second field:

```
36860 uniq -u -f 1 uniq_0I.t
36861 #01 foo0 bar0 fool bar1
36862 #02 bar0 fool bar1 fool
36863 #03 foo0 bar0 fool bar1
36864 #04
36865 #07 bar0 fool bar1 foo0
```

36866 4. This suppresses unique lines, comparing each line of the input file data starting from the  
 36867 third character:

```
36868 uniq -d -s 2 uniq_0I.t
```

36869 In the last example, the *uniq* utility found no input matching the above criteria.

36870 **RATIONALE**

36871           Some historical implementations have limited lines to be 1 080 bytes in length, which does not  
36872           meet the implied {LINE\_MAX} limit.

36873 **FUTURE DIRECTIONS**

36874           None.

36875 **SEE ALSO**

36876           *comm, sort*

36877 **CHANGE HISTORY**

36878           First released in Issue 2.

36879 **Issue 6**

36880           The obsolescent SYNOPSIS and associated text are removed.

36881           The normative text is reworded to avoid use of the term “must” for application requirements.

36882 **NAME**36883 unlink — call the *unlink()* function36884 **SYNOPSIS**36885 XSI unlink *file*

36886

36887 **DESCRIPTION**36888 The *unlink* utility shall perform the function call:36889 unlink(*file*);36890 A user may need appropriate privilege to invoke the *unlink* utility.36891 **OPTIONS**

36892 None.

36893 **OPERANDS**

36894 The following operands shall be supported:

36895 *file* The pathname of an existing file.36896 **STDIN**

36897 Not used.

36898 **INPUT FILES**

36899 Not used.

36900 **ENVIRONMENT VARIABLES**36901 The following environment variables shall affect the execution of *unlink*:

36902 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 36903 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
 36904 Internationalization Variables for the precedence of internationalization variables  
 36905 used to determine the values of locale categories.)

36906 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 36907 internationalization variables.

36908 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 36909 characters (for example, single-byte as opposed to multi-byte characters in  
 36910 arguments).

36911 *LC\_MESSAGES*

36912 Determine the locale that should be used to affect the format and contents of  
 36913 diagnostic messages written to standard error.

36914 *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

36915 **ASYNCHRONOUS EVENTS**

36916 Default.

36917 **STDOUT**

36918 None.

36919 **STDERR**

36920 The standard error shall be used only for diagnostic messages. |

36921 **OUTPUT FILES**

36922 None.

36923 **EXTENDED DESCRIPTION**

36924 None.

36925 **EXIT STATUS**

36926 The following exit values shall be returned:

36927 0 Successful completion.

36928 &gt;0 An error occurred.

36929 **CONSEQUENCES OF ERRORS**

36930 Default.

36931 **APPLICATION USAGE**

36932 None.

36933 **EXAMPLES**

36934 None.

36935 **RATIONALE**

36936 None.

36937 **FUTURE DIRECTIONS**

36938 None.

36939 **SEE ALSO**36940 *link*, *rm*, the System Interfaces volume of IEEE Std 1003.1-200x, *unlink()*36941 **CHANGE HISTORY**

36942 First released in Issue 5.

36943 **NAME**

36944 uucp — system-to-system copy

36945 **SYNOPSIS**36946 xSI uucp [-cCdfjmr][-n user] *source-file*... *destination-file* |

36947

36948 **DESCRIPTION**36949 The *uucp* utility shall copy files named by the *source-file* arguments to the *destination-file*  
36950 argument. The files named can be on local or remote systems.36951 The *uucp* utility cannot guarantee support for all character encodings in all circumstances. For  
36952 example, transmission data may be restricted to 7 bits by the underlying network, 8-bit data and  
36953 filenames need not be portable to non-internationalized systems, and so on. Under these  
36954 circumstances, it is recommended that only characters defined in the ISO/IEC 646:1991  
36955 standard International Reference Version (equivalent to ASCII) 7-bit range of characters be used,  
36956 and that only characters defined in the portable filename character set be used for naming files. |  
36957 The protocol for transfer of files is unspecified by IEEE Std 1003.1-200x. |36958 Typical implementations of this utility require a communications line configured to use the Base  
36959 Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface, but other  
36960 communications means may be used. On systems where there are no available communications  
36961 means (either temporarily or permanently), this utility shall write an error message describing  
36962 the problem and exit with a non-zero exit status.36963 **OPTIONS**36964 The *uucp* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section  
36965 12.2, Utility Syntax Guidelines.

36966 The following options shall be supported:

36967 **-c** Do not copy local file to the spool directory for transfer to the remote machine  
36968 (default). |36969 **-C** Force the copy of local files to the spool directory for transfer. |36970 **-d** Make all necessary directories for the file copy (default). |36971 **-f** Do not make intermediate directories for the file copy. |36972 **-j** Write the job identification string to standard output. This job identification can be |  
36973 used by *uustat* to obtain the status or terminate a job. |36974 **-m** Send mail to the requester when the copy is completed. |36975 **-n user** Notify *user* on the remote system that a file was sent. |36976 **-r** Do not start the file transfer; just queue the job. |36977 **OPERANDS**

36978 The following operands shall be supported:

36979 *destination-file*, *source-file*36980 A pathname of a file to be copied to, or from, respectively. Either name can be a  
36981 pathname on the local machine, or can have the form:36982 *system-name*!*pathname*36983 where *system-name* is taken from a list of system names that *uucp* knows about.  
36984 The destination *system-name* can also be a list of names such as:

36985 *system-name!system-name!...!system-name!pathname*

36986 in which case, an attempt is made to send the file via the specified route to the  
36987 destination. Care should be taken to ensure that intermediate nodes in the route  
36988 are willing to forward information.

36989 The shell pattern matching notation characters '?', '\*', and "[...]" appearing  
36990 in *pathname* shall be expanded on the appropriate system.

36991 Pathnames can be one of:

36992 1. An absolute pathname.

36993 2. A pathname preceded by *~user* where *user* is a login name on the specified  
36994 system and is replaced by that user's login directory. Note that if an invalid  
36995 login is specified, the default is to the public directory (called *PUBDIR*; the  
36996 actual location of *PUBDIR* is implementation-defined).

36997 3. A pathname preceded by *~/destination* where *destination* is appended to  
36998 *PUBDIR*.

36999 **Note:** This destination is treated as a filename unless more than one file is being  
37000 transferred by this request or the destination is already a directory. To  
37001 ensure that it is a directory, follow the destination with a '/'. For  
37002 example, *~/dan/* as the destination makes the directory **PUBDIR/dan** if it  
37003 does not exist and put the requested files in that directory.

37004 4. Anything else shall be prefixed by the current directory.

37005 If the result is an erroneous pathname for the remote system, the copy shall fail. If  
37006 the *destination-file* is a directory, the last part of the *source-file* name shall be used.

37007 The read, write, and execute permissions given by *uucp* are implementation-  
37008 defined.

37009 **STDIN**

37010 Not used.

37011 **INPUT FILES**

37012 The files to be copied are regular files.

37013 **ENVIRONMENT VARIABLES**

37014 The following environment variables shall affect the execution of *uucp*:

37015 *LANG* Provide a default value for the internationalization variables that are unset or null.  
37016 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
37017 Internationalization Variables for the precedence of internationalization variables  
37018 used to determine the values of locale categories.)

37019 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
37020 internationalization variables.

37021 *LC\_COLLATE*

37022 Determine the locale for the behavior of ranges, equivalence classes and multi-  
37023 character collating elements within bracketed filename patterns.

37024 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
37025 characters (for example, single-byte as opposed to multi-byte characters in  
37026 arguments and input files) and the behavior of character classes within bracketed  
37027 filename patterns (for example, "[[:lower:]]\*").

- 37028 *LC\_MESSAGES*  
37029 Determine the locale that should be used to affect the format and contents of  
37030 diagnostic messages written to standard error, and informative messages written  
37031 to standard output.
- 37032 *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 37033 **ASYNCHRONOUS EVENTS**  
37034 Default.
- 37035 **STDOUT**  
37036 Not used.
- 37037 **STDERR**  
37038 The standard error shall be used only for diagnostic messages.
- 37039 **OUTPUT FILES**  
37040 The output files (which may be on other systems) are copies of the input files.  
37041 If the *-m* is used, mail files are modified.
- 37042 **EXTENDED DESCRIPTION**  
37043 None.
- 37044 **EXIT STATUS**  
37045 The following exit values shall be returned:  
37046 0 Successful completion.  
37047 >0 An error occurred.
- 37048 **CONSEQUENCES OF ERRORS**  
37049 Default.
- 37050 **APPLICATION USAGE**  
37051 The domain of remotely accessible files can (and for obvious security reasons usually should) be  
37052 severely restricted.  
37053 Note that the *'!*' character in addresses has to be escaped when using *cs**h* as a command  
37054 interpreter because of its history substitution syntax. For *ksh* and *sh* the escape is not necessary,  
37055 but may be used.  
37056 As noted above, shell metacharacters appearing in pathnames are expanded on the appropriate  
37057 system. On an internationalized system, this is done under the control of local settings of  
37058 *LC\_COLLATE* and *LC\_CTYPE*. Thus, care should be taken when using bracketed filename  
37059 patterns, as collation and typing rules may vary from one system to another. Also be aware that  
37060 certain types of expression (that is, equivalence classes, character classes, and collating symbols)  
37061 need not be supported on non-internationalized systems.
- 37062 **EXAMPLES**  
37063 None.
- 37064 **RATIONALE**  
37065 None.
- 37066 **FUTURE DIRECTIONS**  
37067 None.

37068 **SEE ALSO**37069 *mailx, uuencode, uustat, uux*37070 **CHANGE HISTORY**

37071 First released in Issue 2.

37072 **Issue 6**37073 The *LC\_TIME* and *TZ* entries are removed from the ENVIRONMENT VARIABLES section. |37074 The UN margin codes and associated shading are removed from the **-C**, **-f**, **-j**, **-n**, and **-r** |  
37075 options in response to The Open Group Base Resolution bwg2001-003. |

37076 **NAME**

37077 uudecode — decode a binary file

37078 **SYNOPSIS**37079 UP uudecode [-o *outfile*][*file*]

37080

37081 **DESCRIPTION**

37082 The *uudecode* utility shall read a file, or standard input if no file is specified, that includes data  
 37083 created by the *uuencode* utility. The *uudecode* utility shall scan the input file, searching for data  
 37084 compatible with one of the formats specified in *uuencode* and attempt to create or overwrite the  
 37085 file described by the data (or overridden by the **-o** option). The pathname shall be contained in  
 37086 the data or specified by the **-o** option. The file access permission bits and contents for the file to  
 37087 be produced shall be contained in that data. The mode bits of the created file (other than  
 37088 standard output) shall be set from the file access permission bits contained in the data; that is,  
 37089 other attributes of the mode, including the file mode creation mask (see *umask*), shall not affect  
 37090 the file being produced.

37091 If the pathname of the file to be produced exists, and the user does not have write permission on  
 37092 that file, *uudecode* shall terminate with an error. If the pathname of the file to be produced exists,  
 37093 and the user has write permission on that file, the existing file shall be overwritten.

37094 If the input data was produced by *uuencode* on a system with a different number of bits per byte  
 37095 than on the target system, the results of *uudecode* are unspecified.

37096 **OPTIONS**

37097 The *uudecode* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x,  
 37098 Section 12.2, Utility Syntax Guidelines.

37099 The following option shall be supported by the implementation:

37100 **-o *outfile*** A pathname of a file that shall be used instead of any pathname contained in the  
 37101 input data. Specifying an *outfile* option-argument of **/dev/stdout** shall indicate  
 37102 standard output.

37103 **OPERANDS**

37104 The following operand shall be supported:

37105 *file* The pathname of a file containing the output of *uuencode*.

37106 **STDIN**

37107 See the INPUT FILES section.

37108 **INPUT FILES**

37109 The input files shall be files containing the output of *uuencode*.

37110 **ENVIRONMENT VARIABLES**

37111 The following environment variables shall affect the execution of *uudecode*:

37112 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 37113 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
 37114 Internationalization Variables for the precedence of internationalization variables  
 37115 used to determine the values of locale categories.)

37116 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 37117 internationalization variables.

37118 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 37119 characters (for example, single-byte as opposed to multi-byte characters in  
 37120 arguments and input files).

37121 *LC\_MESSAGES*

37122 Determine the locale that should be used to affect the format and contents of  
37123 diagnostic messages written to standard error.

37124 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

37125 **ASYNCHRONOUS EVENTS**

37126 Default.

37127 **STDOUT**

37128 If the file data header encoded by *uencode* is `-` or `/dev/stdout`, or the `-o /dev/stdout` option  
37129 overrides the file data, the standard output shall be in the same format as the file originally  
37130 encoded by *uencode*. Otherwise, the standard output shall not be used.

37131 **STDERR**

37132 The standard error shall be used only for diagnostic messages.

37133 **OUTPUT FILES**

37134 The output file shall be in the same format as the file originally encoded by *uencode*.

37135 **EXTENDED DESCRIPTION**

37136 None.

37137 **EXIT STATUS**

37138 The following exit values shall be returned:

37139 0 Successful completion.

37140 >0 An error occurred.

37141 **CONSEQUENCES OF ERRORS**

37142 Default.

37143 **APPLICATION USAGE**

37144 The user who is invoking *uudecode* must have write permission on any file being created.

37145 The output of *uencode* is essentially an encoded bit stream that is not cognizant of byte  
37146 boundaries. It is possible that a 9-bit byte target machine can process input from an 8-bit source,  
37147 if it is aware of the requirement, but the reverse is unlikely to be satisfying. Of course, the only  
37148 data that is meaningful for such a transfer between architectures is generally character data.

37149 **EXAMPLES**

37150 None.

37151 **RATIONALE**

37152 Input files are not necessarily text files, as stated by an early proposal. Although the *uencode*  
37153 output is a text file, that output could have been wrapped within another file or mail message  
37154 that is not a text file.

37155 The `-o` option is not historical practice, but was added at the request of WG15 so that the user  
37156 could override the target pathname without having to edit the input data itself.

37157 In early drafts, the `[-o outfile]` option-argument allowed the use of `-` to mean standard output.  
37158 The symbol `-` has only been used previously in IEEE Std 1003.1-200x as a standard input  
37159 indicator. The developers of the standard did not wish to overload the meaning of `-` in this  
37160 manner. The `/dev/stdout` concept exists on most modern systems. The `/dev/stdout` syntax does  
37161 not refer to a new special file. It is just a magic cookie to specify standard output.

37162 **FUTURE DIRECTIONS**

37163           None.

37164 **SEE ALSO**37165           *uuencode*37166 **CHANGE HISTORY**

37167           First released in Issue 4.

37168 **Issue 6**

37169           This utility is now marked as part of the User Portability Utilities option.

37170           The **-o** *outfile* option is added, as specified in the IEEE P1003.2b draft standard.

37171           The normative text is reworded to avoid use of the term “must” for application requirements.

37172 **NAME**

37173 uuencode — encode a binary file

37174 **SYNOPSIS**

37175 UP uuencode [-m][*file*] *decode\_pathname*

37176

37177 **DESCRIPTION**

37178 The *uuencode* utility shall write an encoded version of the named input file, or standard input if  
 37179 no *file* is specified, to standard output. The output shall be encoded using one of the algorithms  
 37180 described in the STDOUT section and shall include the file access permission bits (in *chmod* octal  
 37181 or symbolic notation) of the input file and the *decode\_pathname*, for re-creation of the file on  
 37182 another system that conforms to this volume of IEEE Std 1003.1-200x.

37183 **OPTIONS**

37184 The *uuencode* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x,  
 37185 Section 12.2, Utility Syntax Guidelines.

37186 The following option shall be supported by the implementation:

37187 **-m** Encode the output using the MIME Base64 algorithm described in STDOUT. If **-m**  
 37188 is not specified, the historical algorithm described in STDOUT shall be used.

37189 **OPERANDS**

37190 The following operands shall be supported:

37191 *decode\_pathname*

37192 The pathname of the file into which the *uudecode* utility shall place the decoded  
 37193 file. Specifying a *decode\_pathname* operand of */dev/stdout* shall indicate that  
 37194 *uudecode* is to use standard output. If there are characters in *decode\_pathname* that  
 37195 are not in the portable filename character set the results are unspecified.

37196 *file* A pathname of the file to be encoded.

37197 **STDIN**

37198 See the INPUT FILES section.

37199 **INPUT FILES**

37200 Input files can be files of any type.

37201 **ENVIRONMENT VARIABLES**

37202 The following environment variables shall affect the execution of *uuencode*:

37203 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 37204 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
 37205 Internationalization Variables for the precedence of internationalization variables  
 37206 used to determine the values of locale categories.)

37207 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 37208 internationalization variables.

37209 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 37210 characters (for example, single-byte as opposed to multi-byte characters in  
 37211 arguments and input files).

37212 **LC\_MESSAGES**

37213 Determine the locale that should be used to affect the format and contents of  
 37214 diagnostic messages written to standard error.

37215 xSI **NLSPATH** Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

37216 **ASYNCHRONOUS EVENTS**

37217 Default.

37218 **STDOUT**

37219 **uuencode Base64 Algorithm**

37220 The standard output shall be a text file (encoded in the character set of the current locale) that  
37221 begins with the line:

37222 "begin-base64 $\Delta$ %s $\Delta$ %s\n", <mode>, <decode\_pathname>

37223 and ends with the line:

37224 "====\n"

37225 In both cases, the lines shall have no preceding or trailing <blank>s.

37226 The encoding process represents 24-bit groups of input bits as output strings of four encoded  
37227 characters. Proceeding from left to right, a 24-bit input group shall be formed by concatenating  
37228 three 8-bit input groups. Each 24-bit input group then shall be treated as four concatenated 6-bit  
37229 groups, each of which shall be translated into a single digit in the base64 alphabet. When  
37230 encoding a bit stream via the base64 encoding, the bit stream shall be presumed to be ordered  
37231 with the most-significant bit first. That is, the first bit in the stream shall be the high-order bit in  
37232 the first byte, and the eighth bit shall be the low-order bit in the first byte, and so on. Each 6-bit  
37233 group is used as an index into an array of 64 printable characters, as shown in Table 4-21.

37234

**Table 4-21 uuencode Base64 Values**

37235

Value	Encoding	Value	Encoding	Value	Encoding	Value	Encoding
0	A	17	R	34	i	51	z
1	B	18	S	35	j	52	0
2	C	19	T	36	k	53	1
3	D	20	U	37	l	54	2
4	E	21	V	38	m	55	3
5	F	22	W	39	n	56	4
6	G	23	X	40	o	57	5
7	H	24	Y	41	p	58	6
8	I	25	Z	42	q	59	7
9	J	26	a	43	r	60	8
10	K	27	b	44	s	61	9
11	L	28	c	45	t	62	+
12	M	29	d	46	u	63	/
13	N	30	e	47	v	(pad)	=
14	O	31	f	48	w		
15	P	32	g	49	x		
16	Q	33	h	50	y		

37253

The character referenced by the index shall be placed in the output string.

37254

37255

37256

The output stream (encoded bytes) shall be represented in lines of no more than 76 characters each. All line breaks or other characters not found in the table shall be ignored by decoding software (see *uudecode*).

37257

37258

Special processing shall be performed if fewer than 24 bits are available at the end of a message or encapsulated part of a message. A full encoding quantum shall always be completed at the

37259 end of a message. When fewer than 24 input bits are available in an input group, zero bits shall  
 37260 be added (on the right) to form an integral number of 6-bit groups. Output character positions  
 37261 that are not required to represent actual input data shall be set to the character '='. Since all  
 37262 base64 input is an integral number of octets, only the following cases can arise:

- 37263 1. The final quantum of encoding input is an integral multiple of 24 bits; here, the final unit of  
 37264 encoded output shall be an integral multiple of 4 characters with no '=' padding.
- 37265 2. The final quantum of encoding input is exactly 16 bits; here, the final unit of encoded  
 37266 output shall be three characters followed by one '=' padding character.
- 37267 3. The final quantum of encoding input is exactly 8 bits; here, the final unit of encoded output  
 37268 shall be two characters followed by two '=' padding characters.

37269 A terminating "====" evaluates to nothing and denotes the end of the encoded data.

37270 **uuencode Historical Algorithm**

37271 The standard output shall be a text file (encoded in the character set of the current locale) that  
 37272 begins with the line:

37273 "beginΔ%sΔ%s\n" <mode>, <decode\_pathname>

37274 and ends with the line:

37275 "end\n"

37276 In both cases, the lines shall have no preceding or trailing <blank>s.

37277 The algorithm that shall be used for lines in between **begin** and **end** takes three octets as input  
 37278 and writes four characters of output by splitting the input at six-bit intervals into four octets,  
 37279 containing data in the lower six bits only. These octets shall be converted to characters by adding  
 37280 a value of 0x20 to each octet, so that each octet is in the range [0x20,0x5f], and then it shall be  
 37281 assumed to represent a printable character in the ISO/IEC 646: 1991 standard encoded character  
 37282 set. It then shall be translated into the corresponding character codes for the codeset in use in the  
 37283 current locale. (For example, the octet 0x41, representing 'A', would be translated to 'A' in the  
 37284 current codeset, such as 0xc1 if it were EBCDIC.)

37285 Where the bits of two octets are combined, the least significant bits of the first octet shall be  
 37286 shifted left and combined with the most significant bits of the second octet shifted right. Thus  
 37287 the three octets *A*, *B*, *C* shall be converted into the four octets:

$$\begin{aligned}
 37288 \quad & 0x20 + ((A \gg 2) \& 0x3F) \\
 37289 \quad & 0x20 + (((A \ll 4) \mid ((B \gg 4) \& 0xF)) \& 0x3F) \\
 37290 \quad & 0x20 + (((B \ll 2) \mid ((C \gg 6) \& 0x3)) \& 0x3F) \\
 37291 \quad & 0x20 + ((C) \& 0x3F)
 \end{aligned}$$

37292 These octets then shall be translated into the local character set.

37293 Each encoded line contains a length character, equal to the number of characters to be decoded  
 37294 plus 0x20 translated to the local character set as described above, followed by the encoded  
 37295 characters. The maximum number of octets to be encoded on each line shall be 45.

37296 **STDERR**

37297 The standard error shall be used only for diagnostic messages.

37298 **OUTPUT FILES**

37299 None.

37300 **EXTENDED DESCRIPTION**

37301 None.

37302 **EXIT STATUS**

37303 The following exit values shall be returned:

37304 0 Successful completion.

37305 &gt;0 An error occurred.

37306 **CONSEQUENCES OF ERRORS**

37307 Default.

37308 **APPLICATION USAGE**37309 The file is expanded by 35 percent (each three octets become four, plus control information)  
37310 causing it to take longer to transmit.

37311 Since this utility is intended to create files to be used for data interchange between systems with  
37312 possibly different codesets, and to represent binary data as a text file, the ISO/IEC 646:1991  
37313 standard was chosen for a midpoint in the algorithm as a known reference point. The output  
37314 from *uuencode* is a text file on the local system. If the output were in the ISO/IEC 646:1991  
37315 standard codeset, it might not be a text file (at least because the <newline>s might not match),  
37316 and the goal of creating a text file would be defeated. If this text file was then carried to another  
37317 machine with the same codeset, it would be perfectly compatible with that system's *uudecode*. If  
37318 it was transmitted over a mail system or sent to a machine with a different codeset, it is assumed  
37319 that, as for every other text file, some translation mechanism would convert it (by the time it  
37320 reached a user on the other system) into an appropriate codeset. This translation only makes  
37321 sense from the local codeset, not if the file has been put into a ISO/IEC 646:1991 standard  
37322 representation first. Similarly, files processed by *uuencode* can be placed in *pax* archives,  
37323 intermixed with other text files in the same codeset.

37324 **EXAMPLES**

37325 None.

37326 **RATIONALE**

37327 A new algorithm was added at the request of the international community to parallel work in  
37328 RFC 2045 (MIME). As with the historical *uuencode* format, the Base64 Content-Transfer-Encoding  
37329 is designed to represent arbitrary sequences of octets in a form that is not humanly readable. A  
37330 65-character subset of the ISO/IEC 646:1991 standard is used, enabling 6 bits to be represented  
37331 per printable character. (The extra 65th character, '=', is used to signify a special processing  
37332 function.)

37333 This subset has the important property that it is represented identically in all versions of the  
37334 ISO/IEC 646:1991 standard, including US ASCII, and all characters in the subset are also  
37335 represented identically in all versions of EBCDIC. The historical *uuencode* algorithm does not  
37336 share this property, which is the reason that a second algorithm was added to the ISO POSIX-2  
37337 standard.

37338 The string "====" was used for the termination instead of the end used in the original format  
37339 because the latter is a string that could be valid encoded input.

37340 In an early draft, the `-m` option was named `-b` (for Base64), but it was renamed to reflect its  
37341 relationship to the RFC 2045. A `-u` was also present to invoke the default algorithm, but since  
37342 this was not historical practice, it was omitted as being unnecessary.

37343 See the RATIONALE section in *uudecode* for the derivation of the `/dev/stdout` symbol.

37344 **FUTURE DIRECTIONS**

37345           None.

37346 **SEE ALSO**37347           *mailx, uudecode*37348 **CHANGE HISTORY**

37349           First released in Issue 4.

37350 **Issue 6**

37351           This utility is now marked as part of the User Portability Utilities option.

37352           The Base64 algorithm and the ability to output to **/dev/stdout** are added as specified in the  
37353           IEEE P1003.2b draft standard.

37354 **NAME**

37355 uustat — uucp status inquiry and job control

37356 **SYNOPSIS**37357 xSI uustat [ -q | -k *jobid* | -r *jobid* ]37358 uustat [-s *system*][-u *user*]

37359

37360 **DESCRIPTION**37361 The *uustat* utility shall display the status of, or cancel, previously specified *uucp* requests, or  
37362 provide general status on *uucp* connections to other systems.37363 When no options are given, *uustat* shall write to standard output the status of all *uucp* requests  
37364 issued by the current user.37365 Typical implementations of this utility require a communications line configured to use the Base  
37366 Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface, but other  
37367 communications means may be used. On systems where there are no available communications  
37368 means (either temporarily or permanently), this utility shall write an error message describing  
37369 the problem and exit with a non-zero exit status.37370 **OPTIONS**37371 The *uustat* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section  
37372 12.2, Utility Syntax Guidelines.

37373 The following options shall be supported:

37374 **-q** Write the jobs queued for each machine.37375 **-k *jobid*** Kill the *uucp* request whose job identification is *jobid*. The application shall ensure  
37376 that the killed *uucp* request belongs to the person invoking *uustat* unless that user  
37377 has appropriate privileges.37378 **-r *jobid*** Rejuvenate *jobid*. The files associated with *jobid* are touched so that their  
37379 modification time is set to the current time. This prevents the cleanup program  
37380 from deleting the job until the jobs modification time reaches the limit imposed by  
37381 the program.37382 **-s *system*** Write the status of all *uucp* requests for remote system *system*.37383 **-u *user*** Write the status of all *uucp* requests issued by *user*.37384 **OPERANDS**

37385 None.

37386 **STDIN**

37387 Not used.

37388 **INPUT FILES**

37389 None.

37390 **ENVIRONMENT VARIABLES**37391 The following environment variables shall affect the execution of *uustat*:37392 **LANG** Provide a default value for the internationalization variables that are unset or null.  
37393 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
37394 Internationalization Variables for the precedence of internationalization variables  
37395 used to determine the values of locale categories.)

- 37396 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
37397 internationalization variables.
- 37398 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
37399 characters (for example, single-byte as opposed to multi-byte characters in  
37400 arguments).
- 37401 *LC\_MESSAGES*  
37402 Determine the locale that should be used to affect the format and contents of  
37403 diagnostic messages written to standard error, and informative messages written  
37404 to standard output.
- 37405 *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 37406 **ASYNCHRONOUS EVENTS**  
37407 Default.
- 37408 **STDOUT**  
37409 The standard output shall consist of information about each job selected, in an unspecified  
37410 format. The information shall include at least the job ID, the user ID or name, and the remote  
37411 system name.
- 37412 **STDERR**  
37413 The standard error shall be used only for diagnostic messages.
- 37414 **OUTPUT FILES**  
37415 None.
- 37416 **EXTENDED DESCRIPTION**  
37417 None.
- 37418 **EXIT STATUS**  
37419 The following exit values shall be returned:  
37420 0 Successful completion.  
37421 >0 An error occurred.
- 37422 **CONSEQUENCES OF ERRORS**  
37423 Default.
- 37424 **APPLICATION USAGE**  
37425 None.
- 37426 **EXAMPLES**  
37427 None.
- 37428 **RATIONALE**  
37429 None.
- 37430 **FUTURE DIRECTIONS**  
37431 None.
- 37432 **SEE ALSO**  
37433 *uucp*
- 37434 **CHANGE HISTORY**  
37435 First released in Issue 2.

37436 **Issue 6**

37437 The normative text is reworded to avoid use of the term “must” for application requirements.

37438 The *LC\_TIME* and *TZ* entries are removed from the ENVIRONMENT VARIABLES section. |

37439 The UN margin code and associated shading are removed from the **-q** option in response to The |

37440 Open Group Base Resolution bwg2001-003. |

## 37441 NAME

37442 uux — remote command execution

## 37443 SYNOPSIS

37444 xSI uux [-np] *command-string* |37445 uux [-jnp] *command-string* |

37446 |

## 37447 DESCRIPTION

37448 The *uux* utility shall gather zero or more files from various systems, execute a shell pipeline (see  
 37449 Section 2.9 (on page 2248)) on a specified system, and then send the standard output of the  
 37450 command to a file on a specified system. Only the first command of a pipeline can have a  
 37451 *system-name!* prefix. All other commands in the pipeline shall be executed on the system of the  
 37452 first command.

37453 The following restrictions are applicable to the shell pipeline processed by *uux*:

- 37454 • In gathering files from different systems, pathname expansion shall not be performed by *uux*. |
- 37455 Thus, a request such as:

```
37456 uux "c99 remsys!~/*.c"
```

37457 would attempt to copy the file named literally \*.c to the local system.

- 37458 • The redirection operators ">>", "<<", ">|", and ">&" shall not be accepted. Any use of  
 37459 these redirection operators shall cause this utility to write an error message describing the  
 37460 problem and exit with a non-zero exit status.

- 37461 • The reserved word ! cannot be used at the head of the pipeline to modify the exit status. (See |
- 37462 the *command-string* operand description below.) |

- 37463 • Alias substitution shall not be performed. |

37464 A filename can be specified as for *uucp*; it can be an absolute pathname, a pathname preceded by  
 37465 *~name* (which is replaced by the corresponding login directory), a pathname specified as |  
 37466 *~/dest* (*dest* is prefixed by the public directory called *PUBDIR*; the actual location of *PUBDIR* is  
 37467 implementation-defined), or a simple filename (which is prefixed by *uux* with the current  
 37468 directory). See *uucp* (on page 3163) for the details.

37469 The execution of commands on remote systems shall take place in an execution directory known  
 37470 to the *uucp* system. All files required for the execution shall be put into this directory unless they  
 37471 already reside on that machine. Therefore, the application shall ensure that non-local filenames  
 37472 (without path or machine reference) are unique within the *uux* request.

37473 The *uux* utility shall attempt to get all files to the execution system. For files that are output files,  
 37474 the application shall ensure that the filename is escaped using parentheses.

37475 The remote system shall notify the user by mail if the requested command on the remote system  
 37476 was disallowed or the files were not accessible. This notification can be turned off by the *-n*  
 37477 option.

37478 Typical implementations of this utility require a communications line configured to use the Base  
 37479 Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface, but other  
 37480 communications means may be used. On systems where there are no available communications  
 37481 means (either temporarily or permanently), this utility shall write an error message describing  
 37482 the problem and exits with a non-zero exit status.

37483 The *uux* utility cannot guarantee support for all character encodings in all circumstances. For  
 37484 example, transmission data may be restricted to 7 bits by the underlying network, 8-bit data and

37485 filenames need not be portable to non-internationalized systems, and so on. Under these  
 37486 circumstances, it is recommended that only characters defined in the ISO/IEC 646:1991  
 37487 standard International Reference Version (equivalent to ASCII) 7-bit range of characters be used |  
 37488 and that only characters defined in the portable filename character set be used for naming files. |

#### 37489 OPTIONS

37490 The *uux* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section  
 37491 12.2, Utility Syntax Guidelines.

37492 The following options shall be supported:

37493 **-p** Make the standard input to *uux* the standard input to the *command-string*. |

37494 **-j** Write the job identification string to standard output. This job identification can be |  
 37495 used by *uustat* to obtain the status or terminate a job. |

37496 **-n** Do not notify the user if the command fails.

#### 37497 OPERANDS

37498 The following operand shall be supported:

37499 *command-string*

37500 A string made up of one or more arguments that are similar to normal command  
 37501 arguments, except that the command and any filenames can be prefixed by  
 37502 *system-name!*. A null *system-name* shall be interpreted as the local system.

#### 37503 STDIN

37504 The standard input shall not be used unless the *'-'* or **-p** option is specified; in those cases, the  
 37505 standard input shall be made the standard input of the *command-string*.

#### 37506 INPUT FILES

37507 Input files shall be selected according to the contents of *command-string*.

#### 37508 ENVIRONMENT VARIABLES

37509 The following environment variables shall affect the execution of *uux*:

37510 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 37511 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
 37512 Internationalization Variables for the precedence of internationalization variables  
 37513 used to determine the values of locale categories.)

37514 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 37515 internationalization variables.

37516 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 37517 characters (for example, single-byte as opposed to multi-byte characters in  
 37518 arguments).

37519 **LC\_MESSAGES**

37520 Determine the locale that should be used to affect the format and contents of  
 37521 diagnostic messages written to standard error.

37522 **NLSPATH** Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

#### 37523 ASYNCHRONOUS EVENTS

37524 Default.

37525 **STDOUT**

37526 The standard output shall be not used unless the `-j` option is specified; in that case, the job  
 37527 identification string shall be written to standard output in the following format:

37528 "%s\n", <jobid>

37529 **STDERR**

37530 The standard error shall be used only for diagnostic messages.

37531 **OUTPUT FILES**

37532 Output files shall be created or written, or both, according to the contents of *command-string*.

37533 If the `-n` is not used, mail files shall be modified following any command or file-access failures  
 37534 on the remote system.

37535 **EXTENDED DESCRIPTION**

37536 None.

37537 **EXIT STATUS**

37538 The following exit values shall be returned:

37539 0 Successful completion.

37540 >0 An error occurred.

37541 **CONSEQUENCES OF ERRORS**

37542 Default.

37543 **APPLICATION USAGE**

37544 Note that, for security reasons, many installations limit the list of commands executable on  
 37545 behalf of an incoming request from *uux*. Many sites permit little more than the receipt of mail  
 37546 via *uux*.

37547 Any characters special to the command interpreter should be quoted either by quoting the entire  
 37548 *command-string* or quoting the special characters as individual arguments.

37549 As noted in *uucp*, shell pattern matching notation characters appearing in pathnames are  
 37550 expanded on the appropriate local system. This is done under the control of local settings of  
 37551 *LC\_COLLATE* and *LC\_CTYPE*. Thus, care should be taken when using bracketed filename  
 37552 patterns, as collation and typing rules may vary from one system to another. Also be aware that  
 37553 certain types of expression (that is, equivalence classes, character classes, and collating symbols)  
 37554 need not be supported on non-internationalized systems.

37555 **EXAMPLES**

37556 1. The following command gets **file1** from system **a** and **file2** file from system **b**, executes *diff*  
 37557 on the local system, and puts the results in **file.diff** in the local *PUBDIR* directory.  
 37558 (*PUBDIR* is the *uucp* public directory on the local system.)

37559 `uux "!diff a!/usr/file1 b!/a4/file2 >!~/file.diff"`

37560 2. The following command fails because *uux* places all files copied to a system in the same  
 37561 working directory. Although the files **xyz** are from two different systems, their filenames  
 37562 are the same and conflict.

37563 `uux "!diff a!/usr1/xyz b!/usr2/xyz >!~/xyz.diff"`

37564 3. The following command succeeds (assuming *diff* is permitted on system **a**) because the file  
 37565 local to system **a** is not copied to the working directory, and hence does not conflict the file  
 37566 from system **c**.

37567 uux "a!diff a!/usr/xyz c!/usr/xyz >!~/xyz.diff"

37568 **RATIONALE**

37569 None.

37570 **FUTURE DIRECTIONS**

37571 None.

37572 **SEE ALSO**

37573 *uucp, uuencode, uustat*

37574 **CHANGE HISTORY**

37575 First released in Issue 2.

37576 **Issue 6**

37577 The obsolescent SYNOPSIS is removed.

37578 The normative text is reworded to avoid use of the term “must” for application requirements. |

37579 The UN margin code and associated shading are removed from the `-j` option in response to The |

37580 Open Group Base Resolution bwg2001-003. |

37581 **NAME**37582 `val` — validate SCCS files (**DEVELOPMENT**)37583 **SYNOPSIS**37584 XSI `val -`37585 `val [-s][-m name][-r SID][-y type] file...`

37586

37587 **DESCRIPTION**37588 The `val` utility shall determine whether the specified *file* is an SCCS file meeting the  
37589 characteristics specified by the options.37590 **OPTIONS**37591 The `val` utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2,  
37592 Utility Syntax Guidelines, except that the usage of the `'-'` operand is not strictly as intended by  
37593 the guidelines (that is, reading options and operands from standard input).

37594 The following options shall be supported:

37595 `-m name` Specify a *name*, which is compared with the SCCS `%M%` keyword in *file*; see *get*  
37596 (on page 2675).37597 `-r SID` Specify a *SID* (SCCS Identification String), an SCCS delta number. A check shall be  
37598 made to determine whether the *SID* is ambiguous (for example, `-r 1` is ambiguous  
37599 because it physically does not exist but implies 1.1, 1.2, and so on, which may  
37600 exist) or invalid (for example, `-r 1.0` or `-r 1.1.0` are invalid because neither case can  
37601 exist as a valid delta number). If the *SID* is valid and not ambiguous, a check shall  
37602 be made to determine whether it actually exists.37603 `-s` Silence the diagnostic message normally written to standard output for any error  
37604 that is detected while processing each named file on a given command line.37605 `-y type` Specify a *type*, which shall be compared with the SCCS `%Y%` keyword in *file*; see  
37606 *get* (on page 2675).37607 **OPERANDS**

37608 The following operands shall be supported:

37609 *file* A pathname of an existing SCCS file. If exactly one *file* operand appears, and it is  
37610 `'-'`, the standard input shall be read: each line shall be independently processed |  
37611 as if it were a command line argument list. (However, the line is not subjected to |  
37612 any of the shell word expansions, such as parameter expansion or quote removal.) |37613 **STDIN**37614 The standard input shall be a text file used only when the *file* operand is specified as `'-'`.37615 **INPUT FILES**

37616 Any SCCS files processed shall be files of an unspecified format. |

37617 **ENVIRONMENT VARIABLES**37618 The following environment variables shall affect the execution of *val*:37619 *LANG* Provide a default value for the internationalization variables that are unset or null.  
37620 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
37621 Internationalization Variables for the precedence of internationalization variables  
37622 used to determine the values of locale categories.)37623 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
37624 internationalization variables.

37625 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 37626 characters (for example, single-byte as opposed to multi-byte characters in  
 37627 arguments and input files).

37628 **LC\_MESSAGES**  
 37629 Determine the locale that should be used to affect the format and contents of  
 37630 diagnostic messages written to standard error, and informative messages written  
 37631 to standard output.

37632 **NLSPATH** Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

### 37633 ASYNCHRONOUS EVENTS

37634 Default.

### 37635 STDOUT

37636 The standard output shall consist of informative messages about either:

- 37637 1. Each file processed
- 37638 2. Each command line read from standard input

37639 If the standard input is not used, for each *file* operand yielding a discrepancy, the output line  
 37640 shall have the following format:

37641 "%s: %s\n", <pathname>, <unspecified string>

37642 If standard input is used, a line of input shall be written before each of the preceding lines for  
 37643 files containing discrepancies:

37644 "%s:\n", <input line>

### 37645 STDERR

37646 Not used.

### 37647 OUTPUT FILES

37648 None.

### 37649 EXTENDED DESCRIPTION

37650 None.

### 37651 EXIT STATUS

37652 The 8-bit code returned by *val* shall be a disjunction of the possible errors, that is, it can be |  
 37653 interpreted as a bit string where set bits are interpreted as follows: |

37654	0x80	=	Missing file argument.
37655	0x40	=	Unknown or duplicate option.
37656	0x20	=	Corrupted SCCS file.
37657	0x10	=	Cannot open file or file not SCCS.
37658	0x08	=	<i>SID</i> is invalid or ambiguous.
37659	0x04	=	<i>SID</i> does not exist.
37660	0x02	=	%Y%, -y mismatch.
37661	0x01	=	%M%, -m mismatch.

37662 Note that *val* can process two or more files on a given command line and can process multiple  
 37663 command lines (when reading the standard input). In these cases an aggregate code shall be  
 37664 returned: a logical OR of the codes generated for each command line and file processed.

37665 **CONSEQUENCES OF ERRORS**

37666 Default.

37667 **APPLICATION USAGE**

37668 Since the *val* exit status sets the 0x80 bit, shell applications checking "\$?" cannot tell if it  
37669 terminated due to a missing file argument or receipt of a signal.

37670 **EXAMPLES**

37671 In a directory with three SCCS files, *s.x* (of *t* type "text"), *s.y*, and *s.z* (a corrupted file), the  
37672 following command could produce the output shown:

37673 val - &lt;&lt;EOF

37674 -y source s.x

37675 -m y s.y

37676 s.z

37677 EOF

37678 -y source s.x

37679 s.x: %Y%, -y mismatch

37680 s.z

37681 s.z: corrupted SCCS file

37682 **RATIONALE**

37683 None.

37684 **FUTURE DIRECTIONS**

37685 None.

37686 **SEE ALSO**37687 *admin, delta, get, prs*37688 **CHANGE HISTORY**

37689 First released in Issue 2.

37690 **Issue 6**

37691 The Open Group Corrigendum U025/4 is applied, correcting a typographical error in the EXIT  
37692 STATUS.

37693 The normative text is reworded to emphasize the term "shall" for implementation requirements.

37694 **NAME**

37695 vi — screen-oriented (visual) display editor

37696 **SYNOPSIS**37697 UP `vi [-rR][-l][-c command][-t tagstring][-w size][file ...]`

37698

37699 **DESCRIPTION**37700 This utility shall be provided on systems that both support the User Portability Utilities option  
37701 and define the POSIX2\_CHAR\_TERM symbol. On other systems it is optional.37702 The *vi* (visual) utility is a screen-oriented text editor. Only the open and visual modes of the  
37703 editor are described in IEEE Std 1003.1-200x; see the line editor *ex* for additional editing  
37704 capabilities used in *vi*. The user can switch back and forth between *vi* and *ex* and execute *ex*  
37705 commands from within *vi*.37706 This reference page uses the term *edit buffer* to describe the current working text. No specific  
37707 implementation is implied by this term. All editing changes are performed on the edit buffer,  
37708 and no changes to it shall affect any file until an editor command writes the file.37709 When using *vi*, the terminal screen acts as a window into the editing buffer. Changes made to  
37710 the editing buffer shall be reflected in the screen display; the position of the cursor on the screen  
37711 shall indicate the position within the editing buffer.37712 Certain terminals do not have all the capabilities necessary to support the complete *vi* definition.  
37713 When these commands cannot be supported on such terminals, this condition shall not produce  
37714 an error message such as “not an editor command” or report a syntax error. The implementation  
37715 may either accept the commands and produce results on the screen that are the result of an  
37716 unsuccessful attempt to meet the requirements of this volume of IEEE Std 1003.1-200x or report  
37717 an error describing the terminal-related deficiency.37718 **OPTIONS**37719 The *vi* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2,  
37720 Utility Syntax Guidelines.

37721 The following options shall be supported:

37722 `-c command` See the *ex* command description of the `-c` option.37723 `-r` See the *ex* command description of the `-r` option.37724 `-R` See the *ex* command description of the `-R` option.37725 `-t tagstring` See the *ex* command description of the `-t` option.37726 `-w size` See the *ex* command description of the `-w` option.37727 **OPERANDS**37728 See the OPERANDS section of the *ex* command for a description of the operands supported by  
37729 the *vi* command.37730 **STDIN**37731 If standard input is not a terminal device, the results are undefined. The standard input consists  
37732 of a series of commands and input text, as described in the EXTENDED DESCRIPTION section.37733 If a read from the standard input returns an error, or if the editor detects an end-of-file condition  
37734 from the standard input, it shall be equivalent to a SIGHUP asynchronous event.

37735 **INPUT FILES**

37736 See the INPUT FILES section of the *ex* command for a description of the input files supported by  
37737 the *vi* command.

37738 **ENVIRONMENT VARIABLES**

37739 See the ENVIRONMENT VARIABLES section of the *ex* command for the environment variables  
37740 that affect the execution of the *vi* command.

37741 **ASYNCHRONOUS EVENTS**

37742 See the ASYNCHRONOUS EVENTS section of the *ex* for the asynchronous events that affect the  
37743 execution of the *vi* command.

37744 **STDOUT**

37745 If standard output is not a terminal device, undefined results occur.

37746 Standard output may be used for writing prompts to the user, for informational messages, and  
37747 for writing lines from the file.

37748 **STDERR**

37749 If standard output is not a terminal device, undefined results occur.

37750 The standard error shall be used only for diagnostic messages.

37751 **OUTPUT FILES**

37752 See the OUTPUT FILES section of the *ex* command for a description of the output files  
37753 supported by the *vi* command.

37754 **EXTENDED DESCRIPTION**

37755 If the terminal does not have the capabilities necessary to support an unspecified portion of the  
37756 *vi* definition, implementations shall start initially in *ex* mode or open mode. Otherwise, after  
37757 initialization, *vi* shall be in command mode; text input mode can be entered by one of several  
37758 commands used to insert or change text. In text input mode, <ESC> can be used to return to  
37759 command mode; other uses of <ESC> are described later in this section; see **Terminate**  
37760 **Command or Input Mode** (on page 3194).

37761 **Initialization in *ex* and *vi***

37762 See **Initialization in *ex* and *vi*** (on page 2559) for a description of *ex* and *vi* initialization for the *vi*  
37763 utility.

37764 **Command Descriptions in *vi***

37765 The following symbols are used in this reference page to represent arguments to commands.

37766 *buffer* See the description of *buffer* in the EXTENDED DESCRIPTION section of the *ex* utility;  
37767 see **Command Descriptions in *ex*** (on page 2569).

37768 In open and visual mode, when a command synopsis shows both [*buffer*] and [*count*]  
37769 preceding the command name, they can be specified in either order.

37770 *count* A positive integer used as an optional argument to most commands, either to give a  
37771 repeat count or as a size. This argument is optional and shall default to 1 unless  
37772 otherwise specified.

37773 The Synopsis lines for the *vi* commands <control>-G, <control>-L, <control>-R,  
37774 <control>-], %, &, ^, **D**, **m**, **M**, **Q**, **u**, **U**, and **ZZ** do not have *count* as an optional  
37775 argument. Regardless, it shall not be an error to specify a *count* to these commands, and  
37776 any specified *count* shall be ignored.

37777 *motion* An optional trailing argument used by the **!**, **<**, **>**, **c**, **d**, and **y** commands, which is used  
 37778 to indicate the region of text that shall be affected by the command. The motion can be  
 37779 either one of the command characters repeated or one of several other *vi* commands  
 37780 (listed in the following table). Each of the applicable commands specifies the region of  
 37781 text matched by repeating the command; each command that can be used as a motion  
 37782 command specifies the region of text it affects.

37783 Commands that take *motion* arguments operate on either lines or characters, depending  
 37784 on the circumstances. When operating on lines, all lines that fall partially or wholly  
 37785 within the text region specified for the command shall be affected. When operating on  
 37786 characters, only the exact characters in the specified text region shall be affected. Each  
 37787 motion command specifies this individually.

37788 When commands that may be motion commands are not used as motion commands,  
 37789 they shall set the current position to the current line and column as specified.

37790 The following commands shall be valid cursor motion commands: |

37791	<control>-H	(	E	]]	t	
37792	<newline>	)	F	^	w	
37793	<carriage-return>	+	G	b	{	
37794	<control>-N	,	H	e		
37795	<control>-P	-	L	f	}	
37796	<space>	/	M	h	0	
37797	\$	_	N	j		
37798	%	;	T	k		
37799	'character	?	W	l		
37800	`character	B	[	[	n	

37801 Any *count* that is specified to a command that has an associated motion command shall |  
 37802 be applied to the motion command. If a *count* is applied to both the command and its  
 37803 associated motion command, the effect shall be multiplicative.

37804 The following symbols are used in this section to specify locations in the edit buffer:

37805 *current character*

37806 The character that is currently indicated by the cursor. |

37807 *end of a line*

37808 The point located between the last non-<newline> (if any) and the terminating  
 37809 <newline> of a line. For an empty line, this location coincides with the beginning of the  
 37810 line.

37811 *end of the edit buffer*

37812 The location corresponding to the end of the last line in the edit buffer.

37813 The following symbols are used in this section to specify command actions:

37814 *bigword* In the POSIX locale, *vi* shall recognize four kinds of *bigwords*:

- 37815 1. A maximal sequence of non-<blank>s preceded and followed by <blank>s or the  
 37816 beginning or end of a line or the edit buffer
- 37817 2. One or more sequential blank lines
- 37818 3. The first character in the edit buffer
- 37819 4. The last non-<newline> in the edit buffer

- 37820        *word*        In the POSIX locale, *vi* shall recognize five kinds of words:
- 37821                    1. A maximal sequence of letters, digits, and underscores, delimited at both ends by:
- 37822                    — Characters other than letters, digits, or underscores
- 37823                    — The beginning or end of a line
- 37824                    — The beginning or end of the edit buffer
- 37825                    2. A maximal sequence of characters other than letters, digits, underscores, or
- 37826                    <blank>s, delimited at both ends by:
- 37827                    — A letter, digit, underscore
- 37828                    — <blank>s
- 37829                    — The beginning or end of a line
- 37830                    — The beginning or end of the edit buffer
- 37831                    3. One or more sequential blank lines
- 37832                    4. The first character in the edit buffer
- 37833                    5. The last non-<newline> in the edit buffer
- 37834        *section boundary*
- 37835                    A *section boundary* is one of the following:
- 37836                    1. A line whose first character is a <form-feed>
- 37837                    2. A line whose first character is an open curly brace ( ' { ' )
- 37838                    3. A line whose first character is a period and whose second and third characters
- 37839                    match a two-character pair in the **sections** edit option (see *ed*)
- 37840                    4. A line whose first character is a period and whose only other character matches
- 37841                    the first character of a two-character pair in the **sections** edit option, where the
- 37842                    second character of the two-character pair is a <space>
- 37843                    5. The first line of the edit buffer
- 37844                    6. The last line of the edit buffer if the last line of the edit buffer is empty or if it is a
- 37845                    ]] or } command; otherwise, the last non-<newline> of the last line of the edit
- 37846                    buffer
- 37847        *paragraph boundary*
- 37848                    A *paragraph boundary* is one of the following:
- 37849                    1. A section boundary
- 37850                    2. A line whose first character is a period and whose second and third characters
- 37851                    match a two-character pair in the **paragraphs** edit option (see *ed*)
- 37852                    3. A line whose first character is a period and whose only other character matches
- 37853                    the first character of a two-character pair in the *paragraphs* edit option, where the
- 37854                    second character of the two-character pair is a <space>
- 37855                    4. One or more sequential blank lines
- 37856        *remembered search direction*
- 37857                    See the description of remembered search direction in *ed*.

37858 *sentence boundary*

37859 A *sentence boundary* is one of the following:

- 37860 1. A paragraph boundary
- 37861 2. The first non-<blank> that occurs after a paragraph boundary
- 37862 3. The first non-<blank> that occurs after a period ( ' . ' ), exclamation mark ( ' ! ' ),  
37863 or question mark ( ' ? ' ), followed by two <space>s or the end of a line; any  
37864 number of closing parenthesis ( ' ) ' ), closing brackets ( ' ] ' ), double quote ( ' " ' ),  
37865 or single quote ( ' \ ' ' ) characters can appear between the punctuation mark and  
37866 the two <space>s or end-of-line

37867 In the remainder of the description of the *vi* utility, the term “buffer line” refers to a line in the  
37868 edit buffer and the term “display line” refers to the line or lines on the display screen used to  
37869 display one buffer line. The term “current line” refers to a specific “buffer line”.

37870 If there are display lines on the screen for which there are no corresponding buffer lines because  
37871 they correspond to lines that would be after the end of the file, they shall be displayed as a single  
37872 tilde ( ' ~ ' ) character, plus the terminating <newline>.

37873 The last line of the screen shall be used to report errors or display informational messages. It  
37874 shall also be used to display the input for “line-oriented commands” ( / , ? , : , and ! ). When a line-  
37875 oriented command is executed, the editor shall enter text input mode on the last line on the  
37876 screen, using the respective command characters as prompt characters. (In the case of the !  
37877 command, the associated motion shall be entered by the user before the editor enters text input  
37878 mode.) The line entered by the user shall be terminated by a <newline>, a non-<control>-V-  
37879 escaped <carriage-return>, or unescaped <ESC>. It is unspecified if more characters than  
37880 require a display width minus one column number of screen columns can be entered.

37881 If any command is executed that overwrites a portion of the screen other than the last line of the  
37882 screen (for example, the *ex suspend*, or ! commands), other than the *ex shell* command, the user  
37883 shall be prompted for a character before the screen is refreshed and the edit session continued.

37884 <tab>s shall take up the number of columns on the screen set by the **tabstop** edit option (see *ed*),  
37885 unless there are less than that number of columns before the display margin that will cause the  
37886 displayed line to be folded; in this case, they shall only take up the number of columns up to that  
37887 boundary.

37888 The cursor shall be placed on the current line and relative to the current column as specified by  
37889 each command described in the following sections.

37890 In open mode, if the current line is not already displayed, then it shall be displayed.

37891 In visual mode, if the current line is not displayed, then the lines that are displayed shall be  
37892 expanded, scrolled, or redrawn to cause an unspecified portion of the current line to be  
37893 displayed. If the screen is redrawn, no more than the number of display lines specified by the  
37894 value of the **window** edit option shall be displayed (unless the current line cannot be completely  
37895 displayed in the number of display lines specified by the **window** edit option) and the current  
37896 line shall be positioned as close to the center of the displayed lines as possible (within the  
37897 constraints imposed by the distance of the line from the beginning or end of the edit buffer). If  
37898 the current line is before the first line in the display and the screen is scrolled, an unspecified  
37899 portion of the current line shall be placed on the first line of the display. If the current line is after  
37900 the last line in the display and the screen is scrolled, an unspecified portion of the current line  
37901 shall be placed on the last line of the display.

37902 In visual mode, if a line from the edit buffer (other than the current line) does not entirely fit into  
37903 the lines at the bottom of the display that are available for its presentation, the editor may

37904 choose not to display any portion of the line. The lines of the display that do not contain text  
37905 from the edit buffer for this reason shall each consist of a single '@' character.

37906 In visual mode, the editor may choose for unspecified reasons to not update lines in the display  
37907 to correspond to the underlying edit buffer text. The lines of the display that do not correctly  
37908 correspond to text from the edit buffer for this reason shall consist of a single '@' character  
37909 (plus the terminating <newline>), and the <control>-R command shall cause the editor to  
37910 update the screen to correctly represent the edit buffer.

37911 Open and visual mode commands that set the current column set it to a column position in the  
37912 display, and not a character position in the line. In this case, however, the column position in the  
37913 display shall be calculated for a infinite width display; for example, the column related to a  
37914 character that is part of a line that has been folded onto additional screen lines will be offset from  
37915 the display line column where the buffer line begins, not from the beginning of a particular  
37916 display line.

37917 The display cursor column in the display is based on the value of the current column, as follows,  
37918 with each rule applied in turn:

- 37919 1. If the current column is after the last display line column used by the displayed line, the  
37920 display cursor column shall be set to the last display line column occupied by the last non-  
37921 <newline> in the current line; otherwise, the display cursor column shall be set to the  
37922 current column.
- 37923 2. If the character of which some portion is displayed in the display line column specified by  
37924 the display cursor column requires more than a single display line column:
  - 37925 a. If in text input mode, the display cursor column shall be adjusted to the first display  
37926 line column in which any portion of that character is displayed.
  - 37927 b. Otherwise, the display cursor column shall be adjusted to the last display line  
37928 column in which any portion of that character is displayed.

37929 The current column shall not be changed by these adjustments to the display cursor column.

37930 If an error occurs during the parsing or execution of a *vi* command:

- 37931 • The terminal shall be alerted. Execution of the *vi* command shall stop, and the cursor (for  
37932 example, the current line and column) shall not be further modified.
- 37933 • Unless otherwise specified by the following command sections, it is unspecified whether an  
37934 informational message shall be displayed.
- 37935 • Any partially entered *vi* command shall be discarded.
- 37936 • If the *vi* command resulted from a **map** expansion, all characters from that **map** expansion  
37937 shall be discarded, except as otherwise specified by the **map** command (see *ed*).
- 37938 • If the *vi* command resulted from the execution of a buffer, no further commands caused by  
37939 the execution of the buffer shall be executed.

37940 **Page Backwards**37941 *Synopsis:* [count] <control>-B37942 If in open mode, the <control>-B command shall behave identically to the **z** command.  
37943 Otherwise, if the current line is the first line of the edit buffer, it shall be an error.37944 If the **window** edit option is less than 3, display a screen where the last line of the display shall  
37945 be some portion of:37946 *(current first line) -1*

37947 otherwise, display a screen where the first line of the display shall be some portion of:

37948 *(current first line) - count x ((window edit option) -2)*37949 If this calculation would result in a line that is before the first line of the edit buffer, the first line  
37950 of the display shall display some portion of the first line of the edit buffer.37951 *Current line:* If no lines from the previous display remain on the screen, set to the last line of the  
37952 display; otherwise, set to *(line - the number of new lines displayed on this screen)*.37953 *Current column:* Set to non-<blank>.37954 **Scroll Forward**37955 *Synopsis:* [count] <control>-D

37956 If the current line is the last line of the edit buffer, it shall be an error.

37957 If no *count* is specified, *count* shall default to the *count* associated with the previous <control>-D  
37958 or <control>-U command. If there was no previous <control>-D or <control>-U command, *count*  
37959 shall default to the value of the **scroll** edit option.37960 If in open mode, write lines starting with the line after the current line, until *count* lines or the  
37961 last line of the file have been written.37962 *Current line:* If the current line + *count* is past the last line of the edit buffer, set to the last line of  
37963 the edit buffer; otherwise, set to the current line + *count*.37964 *Current column:* Set to non-<blank>.37965 **Scroll Forward by Line**37966 *Synopsis:* [count] <control>-E

37967 Display the line count lines after the last line currently displayed.

37968 If the last line of the edit buffer is displayed, it shall be an error. If there is no line *count* lines  
37969 after the last line currently displayed, the last line of the display shall display some portion of  
37970 the last line of the edit buffer.37971 *Current line:* Unchanged if the previous current character is displayed; otherwise, set to the first  
37972 line displayed.37973 *Current column:* Unchanged.

37974 **Page Forward**37975 *Synopsis:* [count] <control>-F37976 If in open mode, the <control>-F command shall behave identically to the **z** command.  
37977 Otherwise, if the current line is the last line of the edit buffer, it shall be an error.37978 If the **window** edit option is less than 3, display a screen where the first line of the display shall  
37979 be some portion of:37980 (*current last line*) +1

37981 otherwise, display a screen where the first line of the display shall be some portion of:

37982 (*current first line*) + *count* x ((*window edit option*) -2)37983 If this calculation would result in a line that is after the last line of the edit buffer, the last line of  
37984 the display shall display some portion of the last line of the edit buffer.37985 *Current line:* If no lines from the previous display remain on the screen, set to the first line of the  
37986 display; otherwise, set to (*line* + the number of new lines displayed on this screen).37987 *Current column:* Set to non-<blank>.37988 **Display Information**37989 *Synopsis:* <control>-G37990 This command shall be equivalent to the **ex file** command .37991 **Move Cursor Backwards**37992 *Synopsis:* [count] <control>-H

37993 [count] h

37994 the current erase character (see stty)

37995 If there are no characters before the current character on the current line, it shall be an error. If  
37996 there are less than *count* previous characters on the current line, *count* shall be adjusted to the  
37997 number of previous characters on the line.

37998 If used as a motion command:

- 37999 1. The text region shall be from the character before the starting cursor up to and including  
38000 the *count*th character before the starting cursor.
- 38001 2. Any text copied to a buffer shall be in character mode.

38002 If not used as a motion command:

38003 *Current line:* Unchanged.38004 *Current column:* Set to (*column* - the number of columns occupied by *count* characters ending  
38005 with the previous current column).

38006 **Move Down**

38007 *Synopsis:* [count] <newline>  
 38008 [count] <control>-J  
 38009 [count] <control>-M  
 38010 [count] <control>-N  
 38011 [count] j  
 38012 [count] <carriage-return>  
 38013 [count] +

38014 If there are less than *count* lines after the current line in the edit buffer, it shall be an error.

38015 If used as a motion command:

- 38016 1. The text region shall include the starting line and the next *count* – 1 lines.
- 38017 2. Any text copied to a buffer shall be in line mode.

38018 If not used as a motion command:

38019 *Current line:* Set to *current line* + *count*.

38020 *Current column:* Set to non-<blank> for the <carriage-return>, <control>-M, and + commands;  
 38021 otherwise, unchanged.

38022 **Clear and Redisplay**

38023 *Synopsis:* <control>-L

38024 If in open mode, clear the screen and redisplay the current line. Otherwise, clear and redisplay  
 38025 the screen.

38026 *Current line:* Unchanged.

38027 *Current column:* Unchanged.

38028 **Move Up**

38029 *Synopsis:* [count] <control>-P  
 38030 [count] k  
 38031 [count] -

38032 If there are less than *count* lines before the current line in the edit buffer, it shall be an error.

38033 If used as a motion command:

- 38034 1. The text region shall include the starting line and the previous *count* lines.
- 38035 2. Any text copied to a buffer shall be in line mode.

38036 If not used as a motion command:

38037 *Current line:* Set to *current line* – *count*.

38038 *Current column:* Set to non-<blank> for the – command; otherwise, unchanged.

**38039 Redraw Screen**

38040 *Synopsis:* <control>-R

38041 If any lines have been deleted from the display screen and flagged as deleted on the terminal  
38042 using the @ convention (see the beginning of the EXTENDED DESCRIPTION section), they shall  
38043 be redisplayed to match the contents of the edit buffer.

38044 It is unspecified whether lines flagged with @ because they do not fit on the terminal display  
38045 shall be affected.

38046 *Current line:* Unchanged.

38047 *Current column:* Unchanged.

**38048 Scroll Backward**

38049 *Synopsis:* [*count*] <control>-U

38050 If the current line is the first line of the edit buffer, it shall be an error.

38051 If no *count* is specified, *count* shall default to the *count* associated with the previous <control>-D  
38052 or <control>-U command. If there was no previous <control>-D or <control>-U command, *count*  
38053 shall default to the value of the **scroll** edit option.

38054 *Current line:* If *count* is greater than the current line, set to 1; otherwise, set to the current line -  
38055 *count*.

38056 *Current column:* Set to non-<blank>.

**38057 Scroll Backward by Line**

38058 *Synopsis:* [*count*] <control>-Y

38059 Display the line *count* lines before the first line currently displayed.

38060 If the current line is the first line of the edit buffer, it shall be an error. If this calculation would  
38061 result in a line that is before the first line of the edit buffer, the first line of the display shall  
38062 display some portion of the first line of the edit buffer.

38063 *Current line:* Unchanged if the previous current character is displayed; otherwise, set to the first  
38064 line displayed.

38065 *Current column:* Unchanged.

**38066 Edit the Alternate File**

38067 *Synopsis:* <control>-^

38068 This command shall be equivalent to the *ex edit* command, with the alternate pathname as its  
38069 argument.

**38070 Terminate Command or Input Mode**

38071 *Synopsis:* <ESC>

38072 If a partial *vi* command (as defined by at least one, non-*count* character) has been entered,  
38073 discard the *count* and the command character(s).

38074 Otherwise, if no command characters have been entered, and the <ESC> was the result of a map  
38075 expansion, the terminal shall be alerted and the <ESC> character shall be discarded, but it shall  
38076 not be an error.

38077 Otherwise, it shall be an error.

38078 *Current line*: Unchanged.

38079 *Current column*: Unchanged.

### 38080 **Search for tagstring**

38081 *Synopsis*:    <control>-]

38082 If the current character is not a word or <blank>, it shall be an error.

38083 This command shall be equivalent to the *ex tag* command, with the argument to that command  
38084 defined as follows.

38085 If the current character is a <blank>:

- 38086     1. Skip all <blank>s after the cursor up to the end of the line.
- 38087     2. If the end of the line is reached, it shall be an error.

38088 Then, the argument to the *ex tag* command shall be the current character and all subsequent  
38089 characters, up to the first non-word character or the end of the line.

### 38090 **Move Cursor Forward**

38091 *Synopsis*:    [*count*] <space>

38092               [*count*] 1 (ell)

38093 If there are less than *count* non-<newline>s after the cursor on the current line, *count* shall be  
38094 adjusted to the number of non-<newline>s after the cursor on the line.

38095 If used as a motion command:

- 38096     1. If the current or *count*th character after the cursor is the last non-<newline> in the line, the  
38097       text region shall be comprised of the current character up to and including the last non-  
38098       <newline> in the line. Otherwise, the text region shall be from the current character up to,  
38099       but not including, the *count*th character after the cursor.
- 38100     2. Any text copied to a buffer shall be in character mode.

38101 If not used as a motion command:

38102 If there are no non-<newline>s after the current character on the current line, it shall be an error.

38103 *Current line*: Unchanged.

38104 *Current column*: Set to the last column that displays any portion of the *count*th character after the  
38105 current character.

### 38106 **Replace Text with Results from Shell Command**

38107 *Synopsis*:    [*count*] ! *motion shell-commands* <newline>

38108 If the motion command is the ! command repeated:

- 38109     1. If the edit buffer is empty and no *count* was supplied, the command shall be the equivalent  
38110       of the *ex :read !* command, with the text input, and no text shall be copied to any buffer.
- 38111     2. Otherwise:
  - 38112       a. If there are less than *count* -1 lines after the current line in the edit buffer, it shall be  
38113         an error.

38114           b. The text region shall be from the current line up to and including the next *count* -1  
38115           lines.

38116           Otherwise, the text region shall be the lines in which any character of the text region specified by  
38117           the motion command appear.

38118           Any text copied to a buffer shall be in line mode.

38119           This command shall be equivalent to the *ex!* command for the specified lines.

### 38120           **Move Cursor to End-of-line**

38121           *Synopsis:*     [*count*] \$

38122           It shall be an error if there are less than (*count* -1) lines after the current line in the edit buffer.

38123           If used as a motion command:

38124           1. If *count* is 1:

38125           a. It shall be an error if the line is empty.

38126           b. Otherwise, the text region shall consist of all characters from the starting cursor to  
38127           the last non-<newline> in the line, inclusive, and any text copied to a buffer shall be  
38128           in character mode.

38129           2. Otherwise, if the starting cursor position is at or before the first non-<blank> in the line,  
38130           the text region shall consist of the current and the next *count* -1 lines, and any text saved to  
38131           a buffer shall be in line mode.

38132           3. Otherwise, the text region shall consist of all characters from the starting cursor to the last  
38133           non-<newline> in the line that is *count* -1 lines forward from the current line, and any text  
38134           copied to a buffer shall be in character mode.

38135           If not used as a motion command:

38136           *Current line:* Set to the *current line* + *count* -1.

38137           *Current column:* The current column is set to the last display line column of the last non-  
38138           <newline> in the line, or column position 1 if the line is empty.

38139           The current column shall be adjusted to be on the last display line column of the last non-  
38140           <newline> of the current line as subsequent commands change the current line, until a  
38141           command changes the current column.

### 38142           **Move to Matching Character**

38143           *Synopsis:*     %

38144           If the character at the current position is not a parenthesis, bracket, or curly brace, search  
38145           forward in the line to the first one of those characters. If no such character is found, it shall be an  
38146           error.

38147           The matching character shall be the parenthesis, bracket, or curly brace matching the  
38148           parenthesis, bracket, or curly brace, respectively, that was at the current position or that was  
38149           found on the current line.

38150           Matching shall be determined as follows, for an open parenthesis:

38151           1. Set a counter to 1.

38152           2. Search forwards until a parenthesis is found or the end of the edit buffer is reached.

- 38153           3. If the end of the edit buffer is reached, it shall be an error.
- 38154           4. If an open parenthesis is found, increment the counter by 1.
- 38155           5. If a close parenthesis is found, decrement the counter by 1.
- 38156           6. If the counter is zero, the current character is the matching character.
- 38157           Matching for a close parenthesis shall be equivalent, except that the search shall be backwards,  
38158           from the starting character to the beginning of the buffer, a close parenthesis shall increment the  
38159           counter by 1, and an open parenthesis shall decrement the counter by 1.
- 38160           Matching for brackets and curly braces shall be equivalent, except that searching shall be done  
38161           for open and close brackets or open and close curly braces. It is implementation-defined whether  
38162           other characters are searched for and matched as well.
- 38163           If used as a motion command:
- 38164           1. If the matching cursor was after the starting cursor in the edit buffer, and the starting  
38165           cursor position was at or before the first non-<blank> non-<newline> in the starting line,  
38166           and the matching cursor position was at or after the last non-<blank> non-<newline> in  
38167           the matching line, the text region shall consist of the current line to the matching line,  
38168           inclusive, and any text copied to a buffer shall be in line mode.
- 38169           2. If the matching cursor was before the starting cursor in the edit buffer, and the starting  
38170           cursor position was at or after the last non-<blank> non-<newline> in the starting line, and  
38171           the matching cursor position was at or before the first non-<blank> non-<newline> in the  
38172           matching line, the text region shall consist of the current line to the matching line,  
38173           inclusive, and any text copied to a buffer shall be in line mode.
- 38174           3. Otherwise, the text region shall consist of the starting character to the matching character,  
38175           inclusive, and any text copied to a buffer shall be in character mode.
- 38176           If not used as a motion command:
- 38177           *Current line*: Set to the line where the matching character is located.
- 38178           *Current column*: Set to the last column where any portion of the matching character is displayed.
- 38179           **Repeat Substitution**
- 38180           *Synopsis*:       &
- 38181           Repeat the previous substitution command. This command shall be equivalent to the *ex &*  
38182           command with the current line as its addresses, and without *options*, *count*, or *flags*.
- 38183           **Return to Previous Context at Beginning of Line**
- 38184           *Synopsis*:       ' *character*
- 38185           It shall be an error if there is no line in the edit buffer marked by *character*.
- 38186           If used as a motion command:
- 38187           1. If the starting cursor is after the marked cursor, then the locations of the starting cursor  
38188           and the marked cursor in the edit buffer shall be logically swapped.
- 38189           2. The text region shall consist of the starting line up to and including the marked line, and  
38190           any text copied to a buffer shall be in line mode.
- 38191           If not used as a motion command:

38192 *Current line*: Set to the line referenced by the mark.

38193 *Current column*: Set to non-<blank>.

### 38194 **Return to Previous Context**

38195 *Synopsis*:     \ *character*

38196 It shall be an error if the marked line is no longer in the edit buffer. If the marked line no longer  
38197 contains a character in the saved numbered character position, it shall be as if the marked  
38198 position is the first non-<blank>.

38199 If used as a motion command:

- 38200     1. It shall be an error if the marked cursor references the same character in the edit buffer as  
38201       the starting cursor.
- 38202     2. If the starting cursor is after the marked cursor, then the locations of the starting cursor  
38203       and the marked cursor in the edit buffer shall be logically swapped.
- 38204     3. If the starting line is empty or the starting cursor is at or before the first non-<blank> non-  
38205       <newline> of the starting line, and the marked cursor line is empty or the marked cursor  
38206       references the first character of the marked cursor line, the text region shall consist of all  
38207       lines containing characters from the starting cursor to the line before the marked cursor  
38208       line, inclusive, and any text copied to a buffer shall be in line mode.
- 38209     4. Otherwise, if the marked cursor line is empty or the marked cursor references a character  
38210       at or before the first non-<blank> non-<newline> of the marked cursor line, the region of  
38211       text shall be from the starting cursor to the last non-<newline> of the line before the  
38212       marked cursor line, inclusive, and any text copied to a buffer shall be in character mode.
- 38213     5. Otherwise, the region of text shall be from the starting cursor (inclusive), to the marked  
38214       cursor (exclusive), and any text copied to a buffer shall be in character mode.

38215 If not used as a motion command:

38216 *Current line*: Set to the line referenced by the mark.

38217 *Current column*: Set to the last column in which any portion of the character referenced by the  
38218 mark is displayed.

### 38219 **Return to Previous Section**

38220 *Synopsis*:     [ [

38221 Move the cursor backward through the edit buffer to the first character of the previous section  
38222 boundary, *count* times.

38223 If used as a motion command:

- 38224     1. If the starting cursor was at the first character of the starting line or the starting line was  
38225       empty, and the first character of the boundary was the first character of the boundary line,  
38226       the text region shall consist of the current line up to and including the line where the  
38227       *count*th next boundary starts, and any text copied to a buffer shall be in line mode.
- 38228     2. If the boundary was the last line of the edit buffer or the last non-<newline> of the last line  
38229       of the edit buffer, the text region shall consist of the last character in the edit buffer up to  
38230       and including the starting character, and any text saved to a buffer shall be in character  
38231       mode.

38232 3. Otherwise, the text region shall consist of the starting character up to but not including the  
 38233 first character in the *countth* next boundary, and any text copied to a buffer shall be in  
 38234 character mode.

38235 If not used as a motion command:

38236 *Current line*: Set to the line where the *countth* next boundary in the edit buffer starts.

38237 *Current column*: Set to the last column in which any portion of the first character of the *countth*  
 38238 next boundary is displayed, or column position 1 if the line is empty.

### 38239 **Move to Next Section**

38240 *Synopsis*:     ] ]

38241 Move the cursor forward through the edit buffer to the first character of the next section  
 38242 boundary, *count* times.

38243 If used as a motion command:

38244 1. If the starting cursor was at the first character of the starting line or the starting line was  
 38245 empty, and the first character of the boundary was the first character of the boundary line,  
 38246 the text region shall consist of the current line up to and including the line where the  
 38247 *countth* previous boundary starts, and any text copied to a buffer shall be in line mode.

38248 2. If the boundary was the first line of the edit buffer, the text region shall consist of the first  
 38249 character in the edit buffer up to but not including the starting character, and any text  
 38250 copied to a buffer shall be in character mode.

38251 3. Otherwise, the text region shall consist of the first character in the *countth* previous section  
 38252 boundary up to but not including the starting character, and any text copied to a buffer  
 38253 shall be in character mode.

38254 If not used as a motion command:

38255 *Current line*: Set to the line where the *countth* previous boundary in the edit buffer starts.

38256 *Current column*: Set to the last column in which any portion of the first character of the *countth*  
 38257 previous boundary is displayed, or column position 1 if the line is empty.

### 38258 **Move to First Non-<blank> Position on Current Line**

38259 *Synopsis*:     ^

38260 If used as a motion command:

38261 1. If the line has no non-<blank> non-<newline>s, or if the cursor is at the first non-<blank>  
 38262 non-<newline> of the line, it shall be an error.

38263 2. If the cursor is before the first non-<blank> non-<newline> of the line, the text region shall  
 38264 be comprised of the current character, up to, but not including, the first non-<blank> non-  
 38265 <newline> of the line.

38266 3. If the cursor is after the first non-<blank> non-<newline> of the line, the text region shall  
 38267 be from the character before the starting cursor up to and including the first non-<blank>  
 38268 non-<newline> of the line.

38269 4. Any text copied to a buffer shall be in character mode.

38270 If not used as a motion command:

38271 *Current line:* Unchanged.

38272 *Current column:* Set to non-<blank>.

### 38273 **Current and line above**

38274 *Synopsis:* [count] \_

38275 If there are less than *count* - 1 lines after the current line in the edit buffer, it shall be an error.

38276 If used as a motion command:

- 38277 1. If *count* is less than 2, the text region shall be the current line.
- 38278 2. Otherwise, the text region shall include the starting line and the next *count* - 1 lines.
- 38279 3. Any text copied to a buffer shall be in line mode.

38280 If not used as a motion command:

38281 *Current line:* Set to current line + *count* - 1.

38282 *Current column:* Set to non-<blank>.

### 38283 **Move Back to Beginning of Sentence**

38284 *Synopsis:* [count] (

38285 Move backward to the beginning of a sentence. This command shall be equivalent to the `[]`  
38286 command, with the exception that sentence boundaries shall be used instead of section  
38287 boundaries.

### 38288 **Move Forward to Beginning of Sentence**

38289 *Synopsis:* [count] )

38290 Move forward to the beginning of a sentence. This command shall be equivalent to the `[]`  
38291 command, with the exception that sentence boundaries shall be used instead of section  
38292 boundaries.

### 38293 **Move Back to Preceding Paragraph**

38294 *Synopsis:* [count] {

38295 Move back to the beginning of the preceding paragraph. This command shall be equivalent to  
38296 the `[]` command, with the exception that paragraph boundaries shall be used instead of section  
38297 boundaries.

### 38298 **Move Forward to Next Paragraph**

38299 *Synopsis:* [count] }

38300 Move forward to the beginning of the next paragraph. This command shall be equivalent to the  
38301 `[]` command, with the exception that paragraph boundaries shall be used instead of section  
38302 boundaries.

38303 **Move to Specific Column Position**38304 *Synopsis:* [count] |38305 For the purposes of this command, lines that are too long for the current display and that have  
38306 been folded shall be treated as having a single, 1-based, number of columns.38307 If there are less than *count* columns in which characters from the current line are displayed on  
38308 the screen, *count* shall be adjusted to be the last column in which any portion of the line is  
38309 displayed on the screen.

38310 If used as a motion command:

38311 1. If the line is empty, or the cursor character is the same as the character on the *count*th  
38312 column of the line, it shall be an error.38313 2. If the cursor is before the *count*th column of the line, the text region shall be comprised of  
38314 the current character, up to but not including the character on the *count*th column of the  
38315 line.38316 3. If the cursor is after the *count*th column of the line, the text region shall be from the  
38317 character before the starting cursor up to and including the character on the *count*th  
38318 column of the line.

38319 4. Any text copied to a buffer shall be in character mode.

38320 If not used as a motion command:

38321 *Current line:* Unchanged.38322 *Current column:* Set to the last column in which any portion of the character that is displayed in  
38323 the *count* column of the line is displayed.38324 **Reverse Find Character**38325 *Synopsis:* [count] ,38326 If the last **F**, **f**, **T**, or **t** command was **F**, **f**, **T**, or **t**, this command shall be equivalent to an **f**, **F**, **t**, or  
38327 **T** command, respectively, with the specified *count* and the same search character.38328 If there was no previous **F**, **f**, **T**, or **t** command, it shall be an error.38329 **Repeat**38330 *Synopsis:* [count] .38331 Repeat the last **!**, **<**, **>**, **A**, **C**, **D**, **I**, **J**, **O**, **P**, **R**, **S**, **X**, **Y**, **a**, **c**, **d**, **i**, **o**, **p**, **r**, **s**, **x**, **y**, or **~** command. It shall  
38332 be an error if none of these commands have been executed. Commands (other than commands  
38333 that enter text input mode) executed as a result of map expansions, shall not change the value of  
38334 the last repeatable command.38335 Repeated commands with associated motion commands shall repeat the motion command as  
38336 well; however, any specified *count* shall replace the *count*(s) that were originally specified to the  
38337 repeated command or its associated motion command.38338 If the motion component of the repeated command is **f**, **F**, **t**, or **T**, the repeated command shall  
38339 not set the remembered search character for the **;** and **,** commands.38340 If the repeated command is **p** or **P**, and the buffer associated with that command was a numeric  
38341 buffer named with a number less than 9, the buffer associated with the repeated command shall  
38342 be set to be the buffer named by the name of the previous buffer logically incremented by 1.

38343 If the repeated character is a text input command, the input text associated with that command  
38344 is repeated literally:

- 38345 • Input characters are neither macro or abbreviation-expanded.
- 38346 • Input characters are not interpreted in any special way with the exception that <newline>,  
38347 <carriage-return>, and <control>-T behave as described in **Input Mode Commands in vi** (on  
38348 page 3220).

38349 *Current line*: Set as described for the repeated command.

38350 *Current column*: Set as described for the repeated command.

### 38351 **Find Regular Expression**

38352 *Synopsis*: /

38353 If the input line contains no non-<newline>s, it shall be equivalent to a line containing only the  
38354 last regular expression encountered. The enhanced regular expressions supported by vi are  
38355 described in **Regular Expressions in ex** (on page 2592).

38356 Otherwise, the line shall be interpreted as one or more regular expressions, optionally followed  
38357 by an address offset or a vi z command.

38358 If the regular expression is not the last regular expression on the line, or if a line offset or z  
38359 command is specified, the regular expression shall be terminated by an unescaped '/'  
38360 character, which shall not be used as part of the regular expression. If the regular expression is  
38361 not the first regular expression on the line, it shall be preceded by zero or more <blank>s, a  
38362 semicolon, zero or more <blank>s, a leading '/' character, which shall not be interpreted as  
38363 part of the regular expression. It shall be an error to precede any regular expression with any  
38364 characters other than these.

38365 Each search shall begin from the character after the first character of the last match (or, if it is the  
38366 first search, after the cursor). If the **wraps**can edit option is set, the search shall continue to the  
38367 character before the starting cursor character; otherwise, to the end of the edit buffer. It shall be  
38368 an error if any search fails to find a match, and an informational message to this effect shall be  
38369 displayed.

38370 An optional address offset (see **Addressing in ex** (on page 2562)) can be specified after the last  
38371 regular expression by including a trailing '/' character after the regular expression and  
38372 specifying the address offset. This offset will be from the line containing the match for the last  
38373 regular expression specified. It shall be an error if the line offset would indicate a line address  
38374 less than 1 or greater than the last line in the edit buffer. An address offset of zero shall be  
38375 supported. It shall be an error to follow the address offset with any other characters than  
38376 <blank>s.

38377 If not used as a motion command, an optional z command (see **Redraw Window** (on page 3219))  
38378 can be specified after the last regular expression by including a trailing '/' character after the  
38379 regular expression, zero or more <blank>s, a 'z', zero or more <blank>s, an optional new  
38380 **window** edit option value, zero or more <blank>s, and a location character. The effect shall be as  
38381 if the z command was executed after the / command. It shall be an error to follow the z  
38382 command with any other characters than <blank>s.

38383 The remembered search direction shall be set to forward.

38384 If used as a motion command:

- 38385 1. It shall be an error if the last match references the same character in the edit buffer as the  
38386 starting cursor.

- 38387 2. If any address offset is specified, the last match shall be adjusted by the specified offset as  
38388 described previously.
- 38389 3. If the starting cursor is after the last match, then the locations of the starting cursor and the  
38390 last match in the edit buffer shall be logically swapped.
- 38391 4. If any address offset is specified, the text region shall consist of all lines containing  
38392 characters from the starting cursor to the last match line, inclusive, and any text copied to a  
38393 buffer shall be in line mode.
- 38394 5. Otherwise, if the starting line is empty or the starting cursor is at or before the first non-  
38395 <blank> non-<newline> of the starting line, and the last match line is empty or the last  
38396 match starts at the first character of the last match line, the text region shall consist of all  
38397 lines containing characters from the starting cursor to the line before the last match line,  
38398 inclusive, and any text copied to a buffer shall be in line mode.
- 38399 6. Otherwise, if the last match line is empty or the last match begins at a character at or  
38400 before the first non-<blank> non-<newline> of the last match line, the region of text shall  
38401 be from the current cursor to the last non-<newline> of the line before the last match line,  
38402 inclusive, and any text copied to a buffer shall be in character mode.
- 38403 7. Otherwise, the region of text shall be from the current cursor (inclusive), to the first  
38404 character of the last match (exclusive), and any text copied to a buffer shall be in  
38405 character mode.

38406 If not used as a motion command:

38407 *Current line:* If a match is found, set to the last matched line plus the address offset, if any;  
38408 otherwise, unchanged.

38409 *Current column:* Set to the last column on which any portion of the first character in the last  
38410 matched string is displayed, if a match is found; otherwise, unchanged.

### 38411 **Move to First Character in Line**

38412 *Synopsis:* 0 (zero)

38413 Move to the first character on the current line. The character '0' shall not be interpreted as a  
38414 command if it is immediately preceded by a digit.

38415 If used as a motion command:

- 38416 1. If the cursor character is the first character in the line, it shall be an error.
- 38417 2. The text region shall be from the character before the cursor character up to and including  
38418 the first character in the line.
- 38419 3. Any text copied to a buffer shall be in character mode.

38420 If not used as a motion command:

38421 *Current line:* Unchanged.

38422 *Current column:* The last column in which any portion of the first character in the line is  
38423 displayed, or if the line is empty, unchanged.

38424 **Execute an ex Command**38425 *Synopsis:*       :38426 Execute one or more *ex* commands.

38427 If any portion of the screen other than the last line of the screen was overwritten by any *ex*  
 38428 command (except **shell**), *vi* shall display a message indicating that it is waiting for an input from  
 38429 the user, and shall then read a character. This action may also be taken for other, unspecified  
 38430 reasons.

38431 If the next character entered is a ' : ', another *ex* command shall be accepted and executed. Any  
 38432 other character shall cause the screen to be refreshed and *vi* shall return to command mode.

38433 *Current line:* As specified for the *ex* command.38434 *Current column:* As specified for the *ex* command.38435 **Repeat Find**38436 *Synopsis:*       [*count*] ;

38437 This command shall be equivalent to the last **F**, **f**, **T**, or **t** command, with the specified *count*, and  
 38438 with the same search character used for the last **F**, **f**, **T**, or **t** command. If there was no previous **F**,  
 38439 **f**, **T**, or **t** command, it shall be an error.

38440 **Shift Left**38441 *Synopsis:*       [*count*] < *motion*

38442 If the motion command is the &lt; command repeated:

- 38443 1. If there are less than *count* -1 lines after the current line in the edit buffer, it shall be an  
 38444 error.
- 38445 2. The text region shall be from the current line, up to and including the next *count* -1 lines.

38446 Shift any line in the text region specified by the *count* and motion command one shiftwidth (see  
 38447 the *ex* **shiftwidth** option) toward the start of the line, as described by the *ex* < command. The  
 38448 unshifted lines shall be copied to the unnamed buffer in line mode.

38449 *Current line:* If the motion was from the current cursor position toward the end of the edit  
 38450 buffer, unchanged. Otherwise, set to the first line in the edit buffer that is part of the text region  
 38451 specified by the motion command.

38452 *Current column:* Set to non-<blank>.38453 **Shift Right**38454 *Synopsis:*       [*count*] > *motion*

38455 If the motion command is the &gt; command repeated:

- 38456 1. If there are less than *count* -1 lines after the current line in the edit buffer, it shall be an  
 38457 error.
- 38458 2. The text region shall be from the current line, up to and including the next *count* -1 lines.

38459 Shift any line with characters in the text region specified by the *count* and motion command one  
 38460 shiftwidth (see the *ex* **shiftwidth** option) away from the start of the line, as described by the *ex* >  
 38461 command. The unshifted lines shall be copied into the unnamed buffer in line mode.

38462 *Current line:* If the motion was from the current cursor position toward the end of the edit  
 38463 buffer, unchanged. Otherwise, set to the first line in the edit buffer that is part of the text region  
 38464 specified by the motion command.

38465 *Current column:* Set to non-<blank>.

### 38466 **Scan Backwards for Regular Expression**

38467 *Synopsis:* ?

38468 Scan backwards; The ? command shall be equivalent to the / command (see **Find Regular**  
 38469 **Expression** (on page 3202)) with the following exceptions:

- 38470 1. The input prompt shall be a ' ? '.
- 38471 2. Each search shall begin from the character before the first character of the last match (or, if  
 38472 it is the first search, the character before the cursor character).
- 38473 3. The search direction shall be from the cursor toward the beginning of the edit buffer, and  
 38474 the **wrapscan** edit option shall affect whether the search wraps to the end of the edit buffer  
 38475 and continues.
- 38476 4. The remembered search direction shall be set to backward.

### 38477 **Execute**

38478 *Synopsis:* @*buffer*

38479 If the *buffer* is specified as @, the last buffer executed shall be used. If no previous buffer has been  
 38480 executed, it shall be an error.

38481 Behave as if the contents of the named buffer were entered as standard input. After each line of a  
 38482 line-mode buffer, and all but the last line of a character mode buffer, behave as if a <newline>  
 38483 were entered as standard input.

38484 If an error occurs during this process, an error message shall be written, and no more characters  
 38485 resulting from the execution of this command shall be processed.

38486 If a *count* is specified, behave as if that count were entered as user input before the characters  
 38487 from the @ buffer were entered.

38488 *Current line:* As specified for the individual commands.

38489 *Current column:* As specified for the individual commands.

### 38490 **Reverse Case**

38491 *Synopsis:* [*count*] ~

38492 Reverse the case of the current character and the next *count* - 1 characters, such that lowercase  
 38493 characters that have uppercase counterparts shall be changed to uppercase characters, and  
 38494 uppercase characters that have lowercase counterparts shall be changed to lowercase characters,  
 38495 as prescribed by the current locale. No other characters shall be affected by this command.

38496 If there are less than *count* - 1 characters after the cursor in the edit buffer, *count* shall be adjusted  
 38497 to the number of characters after the cursor in the edit buffer minus 1.

38498 For the purposes of this command, the next character after the last non-<newline> on the line  
 38499 shall be the next character in the edit buffer.

38500 *Current line:* Set to the line including the (*count*-1)th character after the cursor.

38501 *Current column*: Set to the last column in which any portion of the (*count*-1)th character after the  
38502 cursor is displayed.

### 38503 **Append**

38504 *Synopsis*: [ *count* ] a

38505 Enter text input mode after the current cursor position. No characters already in the edit buffer  
38506 shall be affected by this command. A *count* shall cause the input text to be appended *count* -1  
38507 more times to the end of the input.

38508 *Current line/column*: As specified for the text input commands (see **Input Mode Commands in vi**  
38509 (on page 3220)).

### 38510 **Append at End-of-Line**

38511 *Synopsis*: [ *count* ] A

38512 This command shall be equivalent to the *vi* command:

38513 \$ [ *count* ] a

38514 (see **Append**).

### 38515 **Move Backward to Preceding Word**

38516 *Synopsis*: [ *count* ] b

38517 With the exception that words are used as the delimiter instead of bigwords, this command shall  
38518 be equivalent to the **B** command.

### 38519 **Move Backward to Preceding Bigword**

38520 *Synopsis*: [ *count* ] B

38521 If the edit buffer is empty or the cursor is on the first character of the edit buffer, it shall be an  
38522 error. If less than *count* bigwords begin between the cursor and the start of the edit buffer, *count*  
38523 shall be adjusted to the number of bigword beginnings between the cursor and the start of the  
38524 edit buffer.

38525 If used as a motion command:

38526 1. The text region shall be from the first character of the *count*th previous bigword beginning  
38527 up to but not including the cursor character.

38528 2. Any text copied to a buffer shall be in character mode.

38529 If not used as a motion command:

38530 *Current line*: Set to the line containing the *current column*.

38531 *Current column*: Set to the last column upon which any part of the first character of the *count*th  
38532 previous bigword is displayed.

- 38533           **Change**
- 38534           *Synopsis:*     [*buffer*][*count*] *c motion*
- 38535           If the motion command is the **c** command repeated:
- 38536           1. The buffer text shall be in line mode.
- 38537           2. If there are less than *count* - 1 lines after the current line in the edit buffer, it shall be an
- 38538           error.
- 38539           3. The text region shall be from the current line up to and including the next *count* - 1 lines.
- 38540           Otherwise, the buffer text mode and text region shall be as specified by the motion command.
- 38541           The replaced text shall be copied into *buffer*, if specified, and into the unnamed buffer. If the text
- 38542           to be replaced contains characters from more than a single line, or the buffer text is in line mode,
- 38543           the replaced text shall be copied into the numeric buffers as well.
- 38544           If the buffer text is in line mode:
- 38545           1. Any lines that contain characters in the region shall be deleted, and the editor shall enter
- 38546           text input mode at the beginning of a new line which shall replace the first line deleted.
- 38547           2. If the **autoindent** edit option is set, **autoindent** characters equal to the **autoindent**
- 38548           characters on the first line deleted shall be inserted as if entered by the user.
- 38549           Otherwise, if characters from more than one line are in the region of text:
- 38550           1. The text shall be deleted.
- 38551           2. Any text remaining in the last line in the text region shall be appended to the first line in
- 38552           the region, and the last line in the region shall be deleted.
- 38553           3. The editor shall enter text input mode after the last character not deleted from the first line
- 38554           in the text region, if any; otherwise, on the first column of the first line in the region.
- 38555           Otherwise:
- 38556           1. If the glyph for ' \$ ' is smaller than the region, the end of the region shall be marked with a
- 38557           ' \$ '.
- 38558           2. The editor shall enter text input mode, overwriting the region of text.
- 38559           *Current line/column:* As specified for the text input commands (see **Input Mode Commands in vi**
- 38560           (on page 3220)).
- 38561           **Change to End-of-Line**
- 38562           *Synopsis:*     [*buffer*][*count*] **C**
- 38563           This command shall be equivalent to the *vi* command:
- 38564           [*buffer*][*count*] **c\$**
- 38565           See the **c** command.

38566 **Delete**38567 *Synopsis:* `[buffer][count] d motion`38568 If the motion command is the **d** command repeated:

- 38569 1. The buffer text shall be in line mode.
- 38570 2. If there are less than *count* - 1 lines after the current line in the edit buffer, it shall be an
- 38571 error.
- 38572 3. The text region shall be from the current line up to and including the next *count* - 1 lines.

38573 Otherwise, the buffer text mode and text region shall be as specified by the motion command.

38574 If in open mode, and the current line is deleted, and the line remains on the display, an '@' character shall be displayed as the first glyph of that line.

38576 Delete the region of text into *buffer*, if specified, and into the unnamed buffer. If the text to be  
38577 deleted contains characters from more than a single line, or the buffer text is in line mode, the  
38578 deleted text shall be copied into the numeric buffers, as well.38579 *Current line:* Set to the first text region line that appears in the edit buffer, unless that line has  
38580 been deleted, in which case it shall be set to the last line in the edit buffer, or line 1 if the edit  
38581 buffer is empty.38582 *Current column:*

- 38583 1. If the line is empty, set to column position 1.
- 38584 2. Otherwise, if the buffer text is in line mode or the motion was from the cursor toward the
- 38585 end of the edit buffer:
- 38586 a. If a character from the current line is displayed in the current column, set to the last
- 38587 column that displays any portion of that character.
- 38588 b. Otherwise, set to the last column in which any portion of any character in the line is
- 38589 displayed.
- 38590 3. Otherwise, if a character is displayed in the column that began the text region, set to the
- 38591 last column that displays any portion of that character.
- 38592 4. Otherwise, set to the last column in which any portion of any character in the line is
- 38593 displayed.

38594 **Delete to End-of-Line**38595 *Synopsis:* `[buffer] D`38596 Delete the text from the current position to the end of the current line; equivalent to the *vi*  
38597 command:38598 `[buffer] d$`

38599 **Move to End-of-Word**38600 *Synopsis:* [count] e38601 With the exception that words are used instead of bigwords as the delimiter, this command shall  
38602 be equivalent to the E command.38603 **Move to End-of-Bigword**38604 *Synopsis:* [count] E38605 If the edit buffer is empty it shall be an error. If less than *count* bigwords end between the cursor  
38606 and the end of the edit buffer, *count* shall be adjusted to the number of bigword endings between  
38607 the cursor and the end of the edit buffer.

38608 If used as a motion command:

- 38609 1. The text region shall be from the last character of the
- count*
- th next bigword up to and
- 
- 38610 including the cursor character.
- 
- 38611 2. Any text copied to a buffer shall be in character mode.

38612 If not used as a motion command:

38613 *Current line:* Set to the line containing the current column.38614 *Current column:* Set to the last column upon which any part of the last character of the *count*th  
38615 next bigword is displayed.38616 **Find Character in Current Line (Forward)**38617 *Synopsis:* [count] f *character*38618 It shall be an error if *count* occurrences of the character do not occur after the cursor in the line.

38619 If used as a motion command:

- 38620 1. The text range shall be from the cursor character up to and including the
- count*
- th
- 
- 38621 occurrence of the specified character after the cursor.
- 
- 38622 2. Any text copied to a buffer shall be in character mode.

38623 If not used as a motion command:

38624 *Current line:* Unchanged.38625 *Current column:* Set to the last column in which any portion of the *count*th occurrence of the  
38626 specified character after the cursor appears in the line.38627 **Find Character in Current Line (Reverse)**38628 *Synopsis:* [count] F *character*38629 It shall be an error if *count* occurrences of the character do not occur before the cursor in the line.

38630 If used as a motion command:

- 38631 1. The text region shall be from the
- count*
- th occurrence of the specified character before the
- 
- 38632 cursor, up to, but not including the cursor character.
- 
- 38633 2. Any text copied to a buffer shall be in character mode.

38634 If not used as a motion command:

- 38635 *Current line*: Unchanged.
- 38636 *Current column*: Set to the last column in which any portion of the *count*th occurrence of the  
38637 specified character before the cursor appears in the line.
- 38638 **Move to Line**
- 38639 *Synopsis*: [ *count* ] G
- 38640 If *count* is not specified, it shall default to the last line of the edit buffer. If *count* is greater than  
38641 the last line of the edit buffer, it shall be an error.
- 38642 If used as a motion command:
- 38643 1. The text region shall be from the cursor line up to and including the specified line.
  - 38644 2. Any text copied to a buffer shall be in line mode.
- 38645 If not used as a motion command:
- 38646 *Current line*: Set to *count* if *count* is specified; otherwise, the last line.
- 38647 *Current column*: Set to non-<blank>.
- 38648 **Move to Top of Screen**
- 38649 *Synopsis*: [ *count* ] H
- 38650 If the beginning of the line *count* greater than the first line of which any portion appears on the  
38651 display does not exist, it shall be an error.
- 38652 If used as a motion command:
- 38653 1. If in open mode, the text region shall be the current line.
  - 38654 2. Otherwise, the text region shall be from the starting line up to and including (the first line  
38655 of the display + *count* -1).
  - 38656 3. Any text copied to a buffer shall be in line mode.
- 38657 If not used as a motion command:
- 38658 If in open mode, this command shall set the current column to non-<blank> and do nothing else.
- 38659 Otherwise, it shall set the current line and current column as follows.
- 38660 *Current line*: Set to (the first line of the display + *count* -1).
- 38661 *Current column*: Set to non-<blank>.
- 38662 **Insert Before Cursor**
- 38663 *Synopsis*: [ *count* ] i
- 38664 Enter text input mode before the current cursor position. No characters already in the edit buffer  
38665 shall be affected by this command. A *count* shall cause the input text to be appended *count* -1  
38666 more times to the end of the input.
- 38667 *Current line/column*: As specified for the text input commands (see **Input Mode Commands in vi**  
38668 (on page 3220)).

38669 **Insert at Beginning of Line**38670 *Synopsis:* [count] I38671 This command shall be equivalent to the *vi* command `^[count]i` command.38672 **Join**38673 *Synopsis:* [count] J

38674 If the current line is the last line in the edit buffer, it shall be an error.

38675 This command shall be equivalent to the *ex* **join** command with no addresses, and an *ex*  
 38676 command *count* value of 1 if *count* was not specified or if a *count* of 1 was specified, and an *ex*  
 38677 command *count* value of *count* -1 for any other value of *count*, except that the current line and  
 38678 column shall be set as follows.

38679 *Current line:* Unchanged.

38680 *Current column:* The last column in which any portion of the character following the last  
 38681 character in the initial line is displayed, or the last non-<newline> in the line if no characters  
 38682 were appended.

38683 **Move to Bottom of Screen**38684 *Synopsis:* [count] L

38685 If the beginning of the line count less than the last line of which any portion appears on the  
 38686 display does not exist, it shall be an error.

38687 If used as a motion command:

- 38688 1. If in open mode, the text region shall be the current line.
- 38689 2. Otherwise, the text region shall include all lines from the starting cursor line to (the last  
38690 line of the display -(count -1)).
- 38691 3. Any text copied to a buffer shall be in line mode.

38692 If not used as a motion command:

- 38693 1. If in open mode, this command shall set the current column to non-<blank> and do  
38694 nothing else.
- 38695 2. Otherwise, it shall set the current line and current column as follows.

38696 *Current line:* Set to (the last line of the display -(count -1)).38697 *Current column:* Set to non-<blank>.38698 **Mark Position**38699 *Synopsis:* m letter

38700 This command shall be equivalent to the *ex* **mark** command with the specified character as an  
 38701 argument.

38702 **Move to Middle of Screen**38703 *Synopsis:* M

38704 The middle line of the display shall be calculated as follows:

38705  $(\text{the top line of the display}) + (((\text{number of lines displayed}) + 1) / 2) - 1$ 

38706 If used as a motion command:

- 38707 1. If in open mode, the text region shall be the current line.
- 38708 2. Otherwise, the text region shall include all lines from the starting cursor line up to and  
38709 including the middle line of the display.
- 38710 3. Any text copied to a buffer shall be in line mode.

38711 If not used as a motion command:

38712 If in open mode, this command shall set the current column to non-&lt;blank&gt; and do nothing else.

38713 Otherwise, it shall set the current line and current column as follows.

38714 *Current line:* Set to the middle line of the display.38715 *Current column:* Set to non-<blank>.38716 **Repeat Regular Expression Find (Forward)**38717 *Synopsis:* n

38718 If the remembered search direction was forward, the **n** command shall be equivalent to the *vi* /  
38719 command with no characters entered by the user. Otherwise, it shall be equivalent to the *vi* ?  
38720 command with no characters entered by the user.

38721 If the **n** command is used as a motion command for the **!** command, the editor shall not enter  
38722 text input mode on the last line on the screen, and shall behave as if the user entered a single '!'  
38723 character as the text input.

38724 **Repeat Regular Expression Find (Reverse)**38725 *Synopsis:* N

38726 Scan for the next match of the last pattern given to / or ?, but in the reverse direction; this is the  
38727 reverse of **n**.

38728 If the remembered search direction was forward, the **N** command shall be equivalent to the *vi* ?  
38729 command with no characters entered by the user. Otherwise, it shall be equivalent to the *vi* /  
38730 command with no characters entered by the user. If the **N** command is used as a motion  
38731 command for the **!** command, the editor shall not enter text input mode on the last line on the  
38732 screen, and shall behave as if the user entered a single **!** character as the text input.

38733 **Insert Empty Line Below**38734 *Synopsis:* o

38735 Enter text input mode in a new line appended after the current line. A *count* shall cause the input  
38736 text to be appended *count* -1 more times to the end of the already added text, each time starting  
38737 on a new, appended line.

38738 *Current line/column:* As specified for the text input commands (see **Input Mode Commands in vi**  
38739 (on page 3220)).

38740 **Insert Empty Line Above**38741 *Synopsis:*     ○

38742 Enter text input mode in a new line inserted before the current line. A *count* shall cause the input  
 38743 text to be appended *count* -1 more times to the end of the already added text, each time starting  
 38744 on a new, appended line.

38745 *Current line/column:* As specified for the text input commands (see **Input Mode Commands in vi**  
 38746 (on page 3220)).

38747 **Put from Buffer Following**38748 *Synopsis:*     [*buffer*] p38749 If no *buffer* is specified, the unnamed buffer shall be used.

38750 If the buffer text is in line mode, the text shall be appended below the current line, and each line  
 38751 of the buffer shall become a new line in the edit buffer. A *count* shall cause the buffer text to be  
 38752 appended *count* -1 more times to the end of the already added text, each time starting on a new,  
 38753 appended line.

38754 If the buffer text is in character mode, the text shall be appended into the current line after the  
 38755 cursor, and each line of the buffer other than the first and last shall become a new line in the edit  
 38756 buffer. A *count* shall cause the buffer text to be appended *count* -1 more times to the end of the  
 38757 already added text, each time starting after the last added character.

38758 *Current line:* If the buffer text is in line mode, set the line to line +1; otherwise, unchanged.38759 *Current column:* If the buffer text is in line mode:

- 38760 1. If there is a non-<blank> in the first line of the buffer, set to the last column on which any  
 38761 portion of the first non-<blank> in the line is displayed.
- 38762 2. If there is no non-<blank> in the first line of the buffer, set to the last column on which any  
 38763 portion of the last non-<newline> in the first line of the buffer is displayed.

38764 If the buffer text is in character mode:

- 38765 1. If the text in the buffer is from more than a single line, then set to the last column on which  
 38766 any portion of the first character from the buffer is displayed.
- 38767 2. Otherwise, if the buffer is the unnamed buffer, set to the last column on which any portion  
 38768 of the last character from the buffer is displayed.
- 38769 3. Otherwise, set to the first column on which any portion of the first character from the  
 38770 buffer is displayed.

38771 **Put from Buffer Before**38772 *Synopsis:*     [*buffer*] P38773 If no *buffer* is specified, the unnamed buffer shall be used.

38774 If the buffer text is in line mode, the text shall be inserted above the current line, and each line of  
 38775 the buffer shall become a new line in the edit buffer. A *count* shall cause the buffer text to be  
 38776 appended *count* -1 more times to the end of the already added text, each time starting on a new,  
 38777 appended line.

38778 If the buffer text is in character mode, the text shall be inserted into the current line before the  
 38779 cursor, and each line of the buffer other than the first and last shall become a new line in the edit  
 38780 buffer. A *count* shall cause the buffer text to be appended *count* -1 more times to the end of the

- 38781 already added text, each time starting after the last added character.
- 38782 *Current line*: Unchanged.
- 38783 *Current column*: If the buffer text is in line mode:
- 38784 1. If there is a non-<blank> in the first line of the buffer, set to the last column on which any
  - 38785 portion of that character is displayed.
  - 38786 2. If there is no non-<blank> in the first line of the buffer, set to the last column on which any
  - 38787 portion of the last non-<newline> in the first line of the buffer is displayed.
- 38788 If the buffer text is in character mode:
- 38789 1. If the buffer is the unnamed buffer, set to the last column on which any portion of the last
  - 38790 character from the buffer is displayed.
  - 38791 2. Otherwise, set to the first column on which any portion of the first character from the
  - 38792 buffer is displayed.
- 38793 **Enter ex Mode**
- 38794 *Synopsis*:     Q
- 38795 Leave visual or open mode and enter *ex* command mode.
- 38796 *Current line*: Unchanged.
- 38797 *Current column*: Unchanged.
- 38798 **Replace Character**
- 38799 *Synopsis*:     [*count*] *r character*
- 38800 Replace the *count* characters at and after the cursor with the specified character. If there are less
- 38801 than *count* non-<newline>s at and after the cursor on the line, it shall be an error.
- 38802 If character is <control>-V, any next character other than the <newline> shall be stripped of any
- 38803 special meaning and used as a literal character.
- 38804 If character is <ESC>, no replacement shall be made and the current line and current column
- 38805 shall be unchanged.
- 38806 If character is <carriage-return> or <newline>, *count* new lines shall be appended to the current
- 38807 line. All but the last of these lines shall be empty. *count* characters at and after the cursor shall be
- 38808 discarded, and any remaining characters after the cursor in the current line shall be moved to the
- 38809 last of the new lines. If the **autoindent** edit option is set, they shall be preceded by the same
- 38810 number of **autoindent** characters found on the line from which the command was executed.
- 38811 *Current line*: Unchanged unless the replacement character is a <carriage-return> or <newline>,
- 38812 in which case it shall be set to line + *count*.
- 38813 *Current column*: Set to the last column position on which a portion of the last replaced character
- 38814 is displayed, or if the replacement character caused new lines to be created, set to non-<blank>.

38815 **Replace Characters**38816 *Synopsis:* R38817 Enter text input mode at the current cursor position possibly replacing text on the current line. A |  
38818 *count* shall cause the input text to be appended *count* - 1 more times to the end of the input.38819 *Current line/column:* As specified for the text input commands (see **Input Mode Commands in vi**  
38820 (on page 3220)).38821 **Substitute Character**38822 *Synopsis:* [*buffer*][*count*] s38823 This command shall be equivalent to the *vi* command:38824 [*buffer*][*count*] c<space>38825 **Substitute Lines**38826 *Synopsis:* [*buffer*][*count*] S38827 This command shall be equivalent to the *vi* command:38828 [*buffer*][*count*] c\_38829 **Move Cursor to Before Character (Forward)**38830 *Synopsis:* [*count*] t *character*38831 It shall be an error if *count* occurrences of the character do not occur after the cursor in the line.

38832 If used as a motion command:

- 38833 1. The text region shall be from the cursor up to but not including the *count*th occurrence of  
38834 the specified character after the cursor.
- 38835 2. Any text copied to a buffer shall be in character mode.

38836 If not used as a motion command:

38837 *Current line:* Unchanged.38838 *Current column:* Set to the last column in which any portion of the character before the *count*th  
38839 occurrence of the specified character after the cursor appears in the line.38840 **Move Cursor to After Character (Reverse)**38841 *Synopsis:* [*count*] T *character*38842 It shall be an error if *count* occurrences of the character do not occur before the cursor in the line.

38843 If used as a motion command:

- 38844 1. If the character before the cursor is the specified character, it shall be an error.
- 38845 2. The text region shall be from the character before the cursor up to but not including the  
38846 *count*th occurrence of the specified character before the cursor.
- 38847 3. Any text copied to a buffer shall be in character mode.

38848 If not used as a motion command:

38849 *Current line:* Unchanged.

38850 *Current column:* Set to the last column in which any portion of the character after the *count*th  
38851 occurrence of the specified character before the cursor appears in the line.

## 38852 **Undo**

38853 *Synopsis:*     u

38854 This command shall be equivalent to the *ex* **undo** command except that the current line and  
38855 current column shall be set as follows:

38856 *Current line:* Set to the first line added or changed if any; otherwise, move to the line preceding  
38857 any deleted text if one exists; otherwise, move to line 1.

38858 *Current column:* If undoing an *ex* command, set to the first non-<blank>.

38859 Otherwise, if undoing a text input command:

- 38860 1. If the command was an **C**, **c**, **O**, **o**, **R**, **S**, or **s** command, the current column shall be set to  
38861 the value it held when the text input command was entered.
- 38862 2. Otherwise, set to the last column in which any portion of the first character after the  
38863 deleted text is displayed, or, if no non-<newline>s follow the text deleted from this line, set  
38864 to the last column in which any portion of the last non-<newline> in the line is displayed,  
38865 or 1 if the line is empty.

38866 Otherwise, if a single line was modified (that is, not added or deleted) by the **u** command:

- 38867 1. If text was added or changed, set to the last column in which any portion of the first  
38868 character added or changed is displayed.
- 38869 2. If text was deleted, set to the last column in which any portion of the first character after  
38870 the deleted text is displayed, or, if no non-<newline>s follow the deleted text, set to the last  
38871 column in which any portion of the last non-<newline> in the line is displayed, or 1 if the  
38872 line is empty.

38873 Otherwise, set to non-<blank>.

## 38874 **Undo Current Line**

38875 *Synopsis:*     U

38876 Restore the current line to its state immediately before the most recent time that it became the  
38877 current line.

38878 *Current line:* Unchanged.

38879 *Current column:* Set to the first column in the line in which any portion of the first character in  
38880 the line is displayed.

## 38881 **Move to Beginning of Word**

38882 *Synopsis:*     [*count*] w

38883 With the exception that words are used as the delimiter instead of bigwords, this command shall  
38884 be equivalent to the **W** command.

38885 **Move to Beginning of Bigword**38886 *Synopsis:* [count] W38887 If the edit buffer is empty, it shall be an error. If there are less than *count* bigwords between the  
38888 cursor and the end of the edit buffer, *count* shall be adjusted to move the cursor to the last  
38889 bigword in the edit buffer.

38890 If used as a motion command:

- 38891 1. If the associated command is **c**, *count* is 1, and the cursor is on a <blank>, the region of text  
38892 shall be the current character and no further action shall be taken.
- 38893 2. If there are less than *count* bigwords between the cursor and the end of the edit buffer, then  
38894 the command shall succeed, and the region of text shall include the last character of the  
38895 edit buffer.
- 38896 3. If there are <blank>s or an end-of-line that precede the *count*th bigword, and the associated  
38897 command is **c**, the region of text shall be up to and including the last character before the  
38898 preceding <blank>s or end-of-line.
- 38899 4. If there are <blank>s or an end-of-line that precede the bigword, and the associated  
38900 command is **d** or **y**, the region of text shall be up to and including the last <blank> the start  
38901 of the bigword or end-of-line.
- 38902 5. Any text copied to a buffer shall be in character mode.

38903 If not used as a motion command:

- 38904 1. If the cursor is on the last character of the edit buffer, it shall be an error.

38905 *Current line:* Set to the line containing the current column.38906 *Current column:* Set to the last column in which any part of the first character of the *count*th next  
38907 bigword is displayed.38908 **Delete Character at Cursor**38909 *Synopsis:* [buffer][count] x38910 Delete the *count* characters at and after the current character into *buffer*, if specified, and into the  
38911 unnamed buffer.38912 If the line is empty, it shall be an error. If there are less than *count* non-<newline>s at and after  
38913 the cursor on the current line, *count* shall be adjusted to the number of non-<newline>s at and  
38914 after the cursor.38915 *Current line:* Unchanged.38916 *Current column:* If the line is empty, set to column position 1. Otherwise, if there were *count* or  
38917 less non-<newline>s at and after the cursor on the current line, set to the last column that  
38918 displays any part of the last non-<newline> of the line. Otherwise, unchanged.

38919 **Delete Character Before Cursor**38920 *Synopsis:* [buffer][count] X38921 Delete the *count* characters before the current character into *buffer*, if specified, and into the  
38922 unnamed buffer.38923 If there are no characters before the current character on the current line, it shall be an error. If  
38924 there are less than *count* previous characters on the current line, *count* shall be adjusted to the  
38925 number of previous characters on the line.38926 *Current line:* Unchanged.38927 *Current column:* Set to (*current column* – *the width of the deleted characters*).38928 **Yank**38929 *Synopsis:* [buffer][count] y *motion*38930 Copy (yank) the region of text into *buffer*, if specified, and into the unnamed buffer.

38931 If the motion command is the y command repeated:

- 38932 1. The buffer shall be in line mode.
- 38933 2. If there are less than *count* – 1 lines after the current line in the edit buffer, it shall be an  
38934 error.
- 38935 3. The text region shall be from the current line up to and including the next *count* – 1 lines.

38936 Otherwise, the buffer text mode and text region shall be as specified by the motion command.

38937 *Current line:* If the motion was from the current cursor position toward the end of the edit  
38938 buffer, unchanged. Otherwise, set to the first line in the edit buffer that is part of the text region  
38939 specified by the motion command.38940 *Current column:*

- 38941 1. If the motion was from the current cursor position toward the end of the edit buffer,  
38942 unchanged.
- 38943 2. Otherwise, if the current line is empty, set to column position 1.
- 38944 3. Otherwise, set to the last column that displays any part of the first character in the file that  
38945 is part of the text region specified by the motion command.

38946 **Yank Current Line**38947 *Synopsis:* [buffer][count] Y38948 This command shall be equivalent to the *vi* command:

38949 [buffer][count] y\_

38950 **Redraw Window**38951 If in open mode, the **z** command shall have the Synopsis:38952 *Synopsis:* [count] z

38953 If *count* is not specified, it shall default to the **window** edit option  $-1$ . The **z** command shall be  
 38954 equivalent to the *ex z* command, with a type character of = and a *count* of *count*  $-2$ , except that  
 38955 the current line and current column shall be set as follows, and the **window** edit option shall not  
 38956 be affected. If the calculation for the *count* argument would result in a negative number, the  
 38957 *count* argument to the *ex z* command shall be zero. A blank line shall be written after the last line  
 38958 is written.

38959 *Current line:* Unchanged.38960 *Current column:* Unchanged.38961 If not in open mode, the **z** command shall have the following Synopsis:38962 *Synopsis:* [line] z [count] character

38963 If *line* is not specified, it shall default to the current line. If *line* is specified, but is greater than the  
 38964 number of lines in the edit buffer, it shall default to the number of lines in the edit buffer.

38965 If *count* is specified, the value of the **window** edit option shall be set to *count* (as described in the  
 38966 *ex window* command), and the screen shall be redrawn.

38967 *line* shall be placed as specified by the following characters:

38968 &lt;newline&gt;, &lt;carriage-return&gt;

38969 Place the beginning of the line on the first line of the display.

38970 . Place the beginning of the line in the center of the display. The middle line of the display  
 38971 shall be calculated as described for the **M** command.

38972 - Place an unspecified portion of the line on the last line of the display.

38973 + If *line* was specified, equivalent to the <newline> case. If *line* was not specified, display a  
 38974 screen where the first line of the display shall be (current last line) +1. If there are no lines  
 38975 after the last line in the display, it shall be an error.

38976 ^ If *line* was specified, display a screen where the last line of the display shall contain an  
 38977 unspecified portion of the first line of a display that had an unspecified portion of the  
 38978 specified line on the last line of the display. If this calculation results in a line before the  
 38979 beginning of the edit buffer, display the first screen of the edit buffer.

38980 Otherwise, display a screen where the last line of the display shall contain an unspecified  
 38981 portion of (current first line  $-1$ ). If this calculation results in a line before the beginning of  
 38982 the edit buffer, it shall be an error.

38983 *Current line:* If *line* and the '^' character were specified:

38984 1. If the first screen was displayed as a result of the command attempting to display lines  
 38985 before the beginning of the edit buffer: if the first screen was already displayed,  
 38986 unchanged; otherwise, set to (current first line  $-1$ ).

38987 2. Otherwise, set to the last line of the display.

38988 If *line* and the '+' character were specified, set to the first line of the display.38989 Otherwise, if *line* was specified, set to *line*.

38990 Otherwise, unchanged.

38991 *Current column*: Set to non-`<blank>`.

## 38992 **Exit**

38993 *Synopsis*:     ZZ

38994 This command shall be equivalent to the `ex xit` command with no addresses, trailing `!`, or  
38995 filename (see the `ex xit` command).

## 38996 **Input Mode Commands in vi**

38997 In text input mode, the current line shall consist of zero or more of the following categories, plus  
38998 the terminating `<newline>`:

38999     1. Characters preceding the text input entry point

39000         Characters in this category shall not be modified during text input mode.

39001     2. **autoindent** characters

39002         **autoindent** characters shall be automatically inserted into each line that is created in text  
39003 input mode, either as a result of entering a `<newline>` or `<carriage-return>` while in text  
39004 input mode, or as an effect of the command itself; for example, `O` or `o` (see the `ex`  
39005 **autoindent** command), as if entered by the user.

39006         It shall be possible to erase **autoindent** characters with the `<control>-D` command; it is  
39007 unspecified whether they can be erased by `<control>-H`, `<control>-U`, and `<control>-W`  
39008 characters. Erasing any **autoindent** character turns the glyph into erase-columns and  
39009 deletes the character from the edit buffer, but does not change its representation on the  
39010 screen.

39011     3. Text input characters

39012         Text input characters are the characters entered by the user. Erasing any text input  
39013 character turns the glyph into erase-columns and deletes the character from the edit buffer,  
39014 but does not change its representation on the screen.

39015         Each text input character entered by the user (that does not have a special meaning) shall  
39016 be treated as follows:

39017         a. The text input character shall be appended to the last character in the edit buffer  
39018 from the first, second, or third categories.

39019         b. If there are no erase-columns on the screen, the text input command was the **R**  
39020 command, and characters in the fifth category from the original line follow the  
39021 cursor, the next such character shall be deleted from the edit buffer. If the **slowopen**  
39022 edit option is not set, the corresponding glyph on the screen shall become erase-  
39023 columns.

39024         c. If there are erase-columns on the screen, as many columns as they occupy, or as are  
39025 necessary, shall be overwritten to display the text input character. (If only part of a  
39026 multi-column glyph is overwritten, the remainder shall be left on the screen, and  
39027 continue to be treated as erase-columns; it is unspecified whether the remainder of  
39028 the glyph is modified in any way.)

39029         d. If additional display line columns are needed to display the text input character:

39030             1. If the **slowopen** edit option is set, the text input characters shall be displayed  
39031 on subsequent display line columns, overwriting any characters displayed in

- 39032 those columns.
- 39033 2. Otherwise, any characters currently displayed on or after the column on the  
39034 display line where the text input character is to be displayed shall be pushed  
39035 ahead the number of display line columns necessary to display the rest of the  
39036 text input character.
- 39037 4. Erase-columns
- 39038 Erase-columns are not logically part of the edit buffer, appearing only on the screen, and  
39039 may be overwritten on the screen by subsequent text input characters. When text input  
39040 mode ends, all erase-columns shall no longer appear on the screen.
- 39041 Erase-columns are initially the region of text specified by the **c** command ( see **Change** (on  
39042 page 3207)) however, erasing **autoindent** or text input characters causes the glyphs of the  
39043 erased characters to be treated as erase-columns.
- 39044 5. Characters following the text region for the **c** command, or the text input entry point for all  
39045 other commands
- 39046 Characters in this category shall not be modified during text input mode, except as  
39047 specified in category 3.b. for the **R** text input command, or as <blank>s deleted when a  
39048 <newline> or <carriage-return> is entered.
- 39049 It is unspecified whether it is an error to attempt to erase past the beginning of a line that was  
39050 created by the entry of a <newline> or <carriage-return> during text input mode. If it is not an  
39051 error, the editor shall behave as if the erasing character was entered immediately after the last  
39052 text input character entered on the previous line, and all of the non-<newline>s on the current  
39053 line shall be treated as erase-columns.
- 39054 When text input mode is entered, or after a text input mode character is entered (except as  
39055 specified for the special characters below), the cursor shall be positioned as follows:
- 39056 1. On the first column that displays any part of the first erase-column, if one exists
- 39057 2. Otherwise, if the **slowopen** edit option is set, on the first display line column after the last  
39058 character in the first, second, or third categories, if one exists
- 39059 3. Otherwise, the first column that displays any part of the first character in the fifth category,  
39060 if one exists
- 39061 4. Otherwise, the display line column after the last character in the first, second, or third  
39062 categories, if one exists
- 39063 5. Otherwise, on column position 1
- 39064 The characters that are updated on the screen during text input mode are unspecified, other than  
39065 that the last text input character shall always be updated, and, if the **slowopen** edit option is not  
39066 set, the current cursor character shall always be updated.
- 39067 The following specifications are for command characters entered during text input mode.

39068 **NUL**

39069 *Synopsis:* NUL

39070 If the first character of the text input is a NUL, the most recently input text shall be input as if  
39071 entered by the user, and then text input mode shall be exited. The text shall be input literally;  
39072 that is, characters are neither macro or abbreviation expanded, nor are any characters interpreted  
39073 in any special manner. It is unspecified whether implementations shall support more than 256  
39074 bytes of remembered input text.

39075 **<control>-D**

39076 *Synopsis:* <control>-D

39077 The <control>-D character shall have no special meaning when in text input mode for a line-  
39078 oriented command (see **Command Descriptions in vi** (on page 3186)).

39079 This command need not be supported on block-mode terminals.

39080 If the cursor does not follow an **autoindent** character, or an **autoindent** character and a '0' or  
39081 '^' character:

- 39082 1. If the cursor is in column position 1, the <control>-D character shall be discarded and no  
39083 further action taken.
- 39084 2. Otherwise, the <control>-D character shall have no special meaning.

39085 If the last input character was a '0', the cursor shall be moved to column position 1.

39086 Otherwise, if the last input character was a '^', the cursor shall be moved to column position 1.  
39087 In addition, the **autoindent** level for the next input line shall be derived from the same line from  
39088 which the **autoindent** level for the current input line was derived.

39089 Otherwise, the cursor shall be moved back to the column after the previous shiftwidth (see the  
39090 *ex* **shiftwidth** command) boundary.

39091 All of the glyphs on columns between the starting cursor position and (inclusively) the ending  
39092 cursor position shall become erase-columns as described in **Input Mode Commands in vi** (on  
39093 page 3220).

39094 *Current line:* Unchanged.

39095 *Current column:* Set to 1 if the <control>-D was preceded by a '^' or '0'; otherwise, set to  
39096 (column - 1) - ((column - 2) % **shiftwidth**).

39097 **<control>-H**

39098 *Synopsis:* <control>-H

39099 If in text input mode for a line-oriented command, and there are no characters to erase, text  
39100 input mode shall be terminated, no further action shall be done for this command, and the  
39101 current line and column shall be unchanged.

39102 If there are characters other than **autoindent** characters that have been input on the current line  
39103 before the cursor, the cursor shall move back one character.

39104 Otherwise, if there are **autoindent** characters on the current line before the cursor, it is  
39105 implementation-defined whether the <control>-H command is an error or if the cursor moves  
39106 back one **autoindent** character.

39107 Otherwise, if the cursor is in column position 1 and there are previous lines that have been input,  
39108 it is implementation-defined whether the <control>-H command is an error or if it is equivalent

- 39109 to entering <control>-H after the last input character on the previous input line.
- 39110 Otherwise, it shall be an error.
- 39111 All of the glyphs on columns between the starting cursor position and (inclusively) the ending  
39112 cursor position shall become erase-columns as described in **Input Mode Commands in vi** (on  
39113 page 3220).
- 39114 The current erase character (see *stty*) shall cause an equivalent action to the <control>-H  
39115 command, unless the previously inserted character was a backslash, in which case it shall be as  
39116 if the literal current erase character had been inserted instead of the backslash.
- 39117 *Current line*: Unchanged, unless previously input lines are erased, in which case it shall be set to  
39118 line -1.
- 39119 *Current column*: Set to the first column that displays any portion of the character backed up  
39120 over.
- 39121 **<newline>**
- 39122 *Synopsis*: <newline>  
39123 <carriage-return>  
39124 <control>-J  
39125 <control>-M
- 39126 If input was part of a line-oriented command, text input mode shall be terminated and the  
39127 command shall continue execution with the input provided.
- 39128 Otherwise, terminate the current line. If there are no characters other than **autoindent** characters  
39129 on the line, all characters on the line shall be discarded. Otherwise, it is unspecified whether the  
39130 **autoindent** characters in the line are modified by entering these characters.
- 39131 Continue text input mode on a new line appended after the current line. If the **slowopen** edit  
39132 option is set, the lines on the screen below the current line shall not be pushed down, but the  
39133 first of them shall be cleared and shall appear to be overwritten. Otherwise, the lines of the  
39134 screen below the current line shall be pushed down.
- 39135 If the **autoindent** edit option is set, an appropriate number of **autoindent** characters shall be  
39136 added as a prefix to the line as described by the *ex autoindent* edit option.
- 39137 All columns after the cursor that are erase-columns (as described in **Input Mode Commands in**  
39138 **vi** (on page 3220)) shall be discarded.
- 39139 If the **autoindent** edit option is set, all <blank>s immediately following the cursor shall be  
39140 discarded.
- 39141 All remaining characters after the cursor shall be transferred to the new line, positioned after any  
39142 **autoindent** characters.
- 39143 *Current line*: Set to current line +1.
- 39144 *Current column*: Set to the first column that displays any portion of the first character after the  
39145 **autoindent** characters on the new line, if any, or the first column position after the last  
39146 **autoindent** character, if any, or column position 1.

- 39147 **<control>-T**
- 39148 *Synopsis:*     <control>-T
- 39149 The <control>-T character shall have no special meaning when in text input mode for a line-  
39150 oriented command (see **Command Descriptions in vi** (on page 3186)).
- 39151 This command need not be supported on block-mode terminals.
- 39152 Behave as if the user entered the minimum number of <blank>s necessary to move the cursor  
39153 forward to the column position after the next **shiftwidth** (see the *ex* **shiftwidth** command)  
39154 boundary.
- 39155 *Current line:* Unchanged.
- 39156 *Current column:* Set to  $column + \mathbf{shiftwidth} - ((column - 1) \% \mathbf{shiftwidth})$ .
- 39157 **<control>-U**
- 39158 *Synopsis:*     <control>-U
- 39159 If there are characters other than **autoindent** characters that have been input on the current line  
39160 before the cursor, the cursor shall move to the first character input after the **autoindent**  
39161 characters.
- 39162 Otherwise, if there are **autoindent** characters on the current line before the cursor, it is  
39163 implementation-defined whether the <control>-U command is an error or if the cursor moves to  
39164 the first column position on the line.
- 39165 Otherwise, if the cursor is in column position 1 and there are previous lines that have been input,  
39166 it is implementation-defined whether the <control>-U command is an error or if it is equivalent  
39167 to entering <control>-U after the last input character on the previous input line.
- 39168 Otherwise, it shall be an error.
- 39169 All of the glyphs on columns between the starting cursor position and (inclusively) the ending  
39170 cursor position shall become erase-columns as described in **Input Mode Commands in vi** (on  
39171 page 3220).
- 39172 The current *kill* character (see *stty*) shall cause an equivalent action to the <control>-U  
39173 command, unless the previously inserted character was a backslash, in which case it shall be as  
39174 if the literal current *kill* character had been inserted instead of the backslash.
- 39175 *Current line:* Unchanged, unless previously input lines are erased, in which case it shall be set to  
39176 line -1.
- 39177 *Current column:* Set to the first column that displays any portion of the last character backed up  
39178 over.
- 39179 **<control>-V**
- 39180 *Synopsis:*     <control>-V  
39181                 <control>-Q
- 39182 Allow the entry of any subsequent character, other than <control>-J or the <newline>, as a literal  
39183 character, removing any special meaning that it may have to the editor in text input mode. If a  
39184 <control>-V or <control>-Q is entered before a <control>-J or <newline>, the <control>-V or  
39185 <control>-Q character shall be discarded, and the <control>-J or <newline> shall behave as  
39186 described in the <newline> command character during input mode.

39187 For purposes of the display only, the editor shall behave as if a '^' character was entered, and  
 39188 the cursor shall be positioned as if overwriting the '^' character. When a subsequent character  
 39189 is entered, the editor shall behave as if that character was entered instead of the original  
 39190 <control>-V or <control>-Q character.

39191 *Current line:* Unchanged.

39192 *Current column:* Unchanged.

39193 **<control>-W**

39194 *Synopsis:* <control>-W

39195 If there are characters other than **autoindent** characters that have been input on the current line  
 39196 before the cursor, the cursor shall move back over the last word preceding the cursor (including  
 39197 any <blank>s between the end of the last word and the current cursor); the cursor shall not  
 39198 move to before the first character after the end of any **autoindent** characters.

39199 Otherwise, if there are **autoindent** characters on the current line before the cursor, it is  
 39200 implementation-defined whether the <control>-W command is an error or if the cursor moves to  
 39201 the first column position on the line.

39202 Otherwise, if the cursor is in column position 1 and there are previous lines that have been input,  
 39203 it is implementation-defined whether the <control>-W command is an error or if it is equivalent  
 39204 to entering <control>-W after the last input character on the previous input line.

39205 Otherwise, it shall be an error.

39206 All of the glyphs on columns between the starting cursor position and (inclusively) the ending  
 39207 cursor position shall become erase-columns as described in **Input Mode Commands in vi** (on  
 39208 page 3220).

39209 *Current line:* Unchanged, unless previously input lines are erased, in which case it shall be set to  
 39210 line -1.

39211 *Current column:* Set to the first column that displays any portion of the last character backed up  
 39212 over.

39213 **<ESC>**

39214 *Synopsis:* <ESC>

39215 If input was part of a line-oriented command:

- 39216 1. If *interrupt* was entered, text input mode shall be terminated and the editor shall return to  
 39217 command mode. The terminal shall be alerted.
- 39218 2. If <ESC> was entered, text input mode shall be terminated and the command shall  
 39219 continue execution with the input provided.

39220 Otherwise, terminate text input mode and return to command mode.

39221 Any **autoindent** characters entered on newly created lines that have no other non-<newline>s  
 39222 shall be deleted.

39223 Any leading **autoindent** and <blank>s on newly created lines shall be rewritten to be the  
 39224 minimum number of <blank>s possible.

39225 The screen shall be redisplayed as necessary to match the contents of the edit buffer.

39226 *Current line:* Unchanged.

- 39227 *Current column:*
- 39228 1. If there are text input characters on the current line, the column shall be set to the last  
39229 column where any portion of the last text input character is displayed.
  - 39230 2. Otherwise, if a character is displayed in the current column, unchanged.
  - 39231 3. Otherwise, set to column position 1.
- 39232 **EXIT STATUS**
- 39233 The following exit values shall be returned:
- 39234 0 Successful completion.
- 39235 >0 An error occurred.
- 39236 **CONSEQUENCES OF ERRORS**
- 39237 When any error is encountered and the standard input is not a terminal device file, *vi* shall not  
39238 write the file or return to command or text input mode, and shall terminate with a non-zero exit  
39239 status.
- 39240 Otherwise, when an unrecoverable error is encountered it shall be equivalent to a SIGHUP  
39241 asynchronous event.
- 39242 Otherwise, when an error is encountered, the editor shall behave as specified in **Command**  
39243 **Descriptions in vi** (on page 3186).
- 39244 **APPLICATION USAGE**
- 39245 None.
- 39246 **EXAMPLES**
- 39247 None.
- 39248 **RATIONALE**
- 39249 See the RATIONALE for *ex* for more information on *vi*. Major portions of the *vi* utility  
39250 specification point to *ex* to avoid inadvertent divergence. While *ex* and *vi* have historically been  
39251 implemented as a single utility, this is not required by IEEE Std 1003.1-200x.
- 39252 It is recognized that portions of *vi* would be difficult, if not impossible, to implement  
39253 satisfactorily on a block-mode terminal, or a terminal without any form of cursor addressing,  
39254 thus it is not a mandatory requirement that such features should work on all terminals. It is the  
39255 intention, however, that a *vi* implementation should provide the full set of capabilities on all  
39256 terminals capable of supporting them.
- 39257 Historically, *vi* exited immediately if the standard input was not a terminal. IEEE Std 1003.1-200x  
39258 permits, but does not require, this behavior. An end-of-file condition is not equivalent to an  
39259 end-of-file character. A common end-of-file character, <control>-D, is historically a *vi* command.
- 39260 The text in the STANDARD OUTPUT section reflects the usage of the verb *display* in this section;  
39261 some implementations of *vi* use standard output to write to the terminal, but  
39262 IEEE Std 1003.1-200x does not require that to be the case.
- 39263 Historically, implementations reverted to open mode if the terminal was incapable of  
39264 supporting full visual mode. IEEE Std 1003.1-200x requires this behavior. Historically, the open  
39265 mode of *vi* behaved roughly equivalently to the visual mode, with the exception that only a  
39266 single line from the edit buffer (one “buffer line”) was kept current at any time. This line was  
39267 normally displayed on the next-to-last line of a terminal with cursor addressing (and the last line  
39268 performed its normal visual functions for line-oriented commands and messages). In addition,  
39269 some few commands behaved differently in open mode than in visual mode.  
39270 IEEE Std 1003.1-200x requires conformance to historical practice.

39271 Historically, *ex* and *vi* implementations have expected text to proceed in the usual  
 39272 European/Latin order of left to right, top to bottom. There is no requirement in  
 39273 IEEE Std 1003.1-200x that this be the case. The specification was deliberately written using  
 39274 words like “before”, “after”, “first”, and “last” in order to permit implementations to support  
 39275 the natural text order of the language.

39276 Historically, lines past the end of the edit buffer were marked with single tilde ('~') characters;  
 39277 that is, if the one-based display was 20 lines in length, and the last line of the file was on line one,  
 39278 then lines 2-20 would contain only a single '~' character.

39279 Historically, the *vi* editor attempted to display only complete lines at the bottom of the screen (it  
 39280 did display partial lines at the top of the screen). If a line was too long to fit in its entirety at the  
 39281 bottom of the screen, the screen lines where the line would have been displayed were displayed  
 39282 as single '@' characters, instead of displaying part of the line. IEEE Std 1003.1-200x permits, but  
 39283 does not require, this behavior. Implementations are encouraged to attempt always to display a  
 39284 complete line at the bottom of the screen when doing scrolling or screen positioning by buffer  
 39285 lines.

39286 Historically, lines marked with '@' were also used to minimize output to dumb terminals over  
 39287 slow lines; that is, changes local to the cursor were updated, but changes to lines on the screen  
 39288 that were not close to the cursor were simply marked with an '@' sign instead of being updated  
 39289 to match the current text. IEEE Std 1003.1-200x permits, but does not require this feature because  
 39290 it is used ever less frequently as terminals become smarter and connections are faster.

### 39291 Initialization in *ex* and *vi*

39292 Historically, *vi* always had a line in the edit buffer, even if the edit buffer was “empty”. For  
 39293 example:

- 39294 1. The *ex* command = executed from visual mode wrote “1” when the buffer was empty.
- 39295 2. Writes from visual mode of an empty edit buffer wrote files of a single character (a  
 39296 <newline>), while writes from *ex* mode of an empty edit buffer wrote empty files.
- 39297 3. Put and read commands into an empty edit buffer left an empty line at the top of the edit  
 39298 buffer.

39299 For consistency, IEEE Std 1003.1-200x does not permit any of these behaviors.

39300 Historically, *vi* did not always return the terminal to its original modes; for example, ICRNL was  
 39301 modified if it was not originally set. IEEE Std 1003.1-200x does not permit this behavior.

### 39302 Command Descriptions in *vi*

39303 Motion commands are among the most complicated aspects of *vi* to describe. With some  
 39304 exceptions, the text region and buffer type effect of a motion command on a *vi* command are  
 39305 described on a case-by-case basis. The descriptions of text regions in IEEE Std 1003.1-200x are  
 39306 not intended to imply direction; that is, an inclusive region from line *n* to line *n*+5 is identical to  
 39307 a region from line *n*+5 to line *n*. This is of more than academic interest—movements to marks  
 39308 can be in either direction, and, if the **wrapsan** option is set, so can movements to search points.  
 39309 Historically, lines are always stored into buffers in text order; that is, from the start of the edit  
 39310 buffer to the end. IEEE Std 1003.1-200x requires conformance to historical practice.

39311 Historically, command counts were applied to any associated motion, and were multiplicative  
 39312 to any supplied motion count. For example, **2cw** is the same as **c2w**, and **2c3w** is the same as  
 39313 **c6w**. IEEE Std 1003.1-200x requires this behavior. Historically, *vi* commands that used bigwords,  
 39314 words, paragraphs, and sentences as objects treated groups of empty lines, or lines that  
 39315 contained only <blank>s, inconsistently. Some commands treated them as a single entity, while

39316 others treated each line separately. For example, the **w**, **W**, and **B** commands treated groups of  
39317 empty lines as individual words; that is, the command would move the cursor to each new  
39318 empty line. The **e** and **E** commands treated groups of empty lines as a single word; that is, the  
39319 first use would move past the group of lines. The **b** command would just beep at the user, or if  
39320 done from the start of the line as a motion command, fail in unexpected ways. If the lines  
39321 contained only (or ended with) <blank>s, the **w** and **W** commands would just beep at the user,  
39322 the **E** and **e** commands would treat the group as a single word, and the **B** and **b** commands  
39323 would treat the lines as individual words. For consistency and simplicity of specification,  
39324 IEEE Std 1003.1-200x requires that all *vi* commands treat groups of empty or blank lines as a  
39325 single entity, and that movement through lines ending with <blank>s be consistent with other  
39326 movements.

39327 Historically, *vi* documentation indicated that any number of double quotes were skipped after  
39328 punctuation marks at sentence boundaries; however, implementations only skipped single  
39329 quotes. IEEE Std 1003.1-200x requires both to be skipped.

39330 Historically, the first and last characters in the edit buffer were word boundaries. This historical  
39331 practice is required by IEEE Std 1003.1-200x.

39332 Historically, *vi* attempted to update the minimum number of columns on the screen possible,  
39333 which could lead to misleading information being displayed. IEEE Std 1003.1-200x makes no  
39334 requirements other than that the current character being entered is displayed correctly, leaving  
39335 all other decisions in this area up to the implementation.

39336 Historically, lines were arbitrarily folded between columns of any characters that required  
39337 multiple column positions on the screen, with the exception of tabs, which terminated at the  
39338 right-hand margin. IEEE Std 1003.1-200x permits the former and requires the latter.  
39339 Implementations that do not arbitrarily break lines between columns of characters that occupy  
39340 multiple column positions should not permit the cursor to rest on a column that does not  
39341 contain any part of a character.

39342 The historical *vi* had a problem in that all movements were by buffer lines, not by display or  
39343 screen lines. This is often the right thing to do; for example, single line movements, such as **j** or  
39344 **k**, should work on buffer lines. Commands like **dj**, or **j.**, where **.** is a change command, only  
39345 make sense for buffer lines. It is not, however, the right thing to do for screen motion or scrolling  
39346 commands like <control>-D, <control>-F, and **H**. If the window is fairly small, using buffer lines  
39347 in these cases can result in completely random motion; for example, **1<control>-D** can result in a  
39348 completely changed screen, without any overlap. This is clearly not what the user wanted. The  
39349 problem is even worse in the case of the **H**, **L**, and **M** commands—as they position the cursor at  
39350 the first non-<blank> of the line, they may all refer to the same location in large lines, and will  
39351 result in no movement at all.

39352 In addition, if the line is larger than the screen, using buffer lines can make it impossible to  
39353 display parts of the line—there are not any commands that do not display the beginning of the  
39354 line in historical *vi*, and if both the beginning and end of the line cannot be on the screen at  
39355 the same time, the user suffers. Finally, the page and half-page scrolling commands historically  
39356 moved to the first non-<blank> in the new line. If the line is approximately the same size as the  
39357 screen, this is inadequate because the cursor before and after a <control>-D command will refer  
39358 to the same location on the screen.

39359 Implementations of *ex* and *vi* exist that do not have these problems because the relevant  
39360 commands (<control>-B, <control>-D, <control>-F, <control>-U, <control>-Y, <control>-E, **H**, **L**,  
39361 and **M**) operate on display (screen) lines, not (edit) buffer lines.

39362 IEEE Std 1003.1-200x does not permit this behavior by default because the standard developers  
39363 believed that users would find it too confusing. However, historical practice has been relaxed.

39364 For example, *ex* and *vi* historically attempted, albeit sometimes unsuccessfully, to never put part  
39365 of a line on the last lines of a screen; for example, if a line would not fit in its entirety, no part of  
39366 the line was displayed, and the screen lines corresponding to the line contained single '@'  
39367 characters. This behavior is permitted, but not required by IEEE Std 1003.1-200x, so that it is  
39368 possible for implementations to support long lines in small screens more reasonably without  
39369 changing the commands to be oriented to the display (instead of oriented to the buffer).  
39370 IEEE Std 1003.1-200x also permits implementations to refuse to edit any edit buffer containing a  
39371 line that will not fit on the screen in its entirety.

39372 The display area (for example, the value of the **window** edit option) has historically been  
39373 “grown”, or expanded, to display new text when local movements are done in displays where  
39374 the number of lines displayed is less than the maximum possible. Expansion has historically  
39375 been the first choice, when the target line is less than the maximum possible expansion value  
39376 away. Scrolling has historically been the next choice, done when the target line is less than half a  
39377 display away, and otherwise, the screen was redrawn. There were exceptions, however, in that  
39378 *ex* commands generally always caused the screen to be redrawn. IEEE Std 1003.1-200x does not  
39379 specify a standard behavior because there may be external issues, such as connection speed, the  
39380 number of characters necessary to redraw as opposed to scroll, or terminal capabilities that  
39381 implementations will have to accommodate.

39382 The current line in IEEE Std 1003.1-200x maps one-to-one to a buffer line in the file. The current  
39383 column does not. There are two different column values that are described by  
39384 IEEE Std 1003.1-200x. The first is the current column value as set by many of the *vi* commands.  
39385 This value is remembered for the lifetime of the editor. The second column value is the actual  
39386 position on the screen where the cursor rests. The two are not always the same. For example,  
39387 when the cursor is backed by a multi-column character, the actual cursor position on the screen  
39388 has historically been the last column of the character in command mode, and the first column of  
39389 the character in input mode.

39390 Commands that set the current line, but that do not set the current cursor value (for example, **j**  
39391 and **k**) attempt to get as close as possible to the remembered column position, so that the cursor  
39392 tends to restrict itself to a vertical column as the user moves around in the edit buffer.  
39393 IEEE Std 1003.1-200x requires conformance to historical practice, requiring that the display  
39394 location of the cursor on the display line be adjusted from the current column value as necessary  
39395 to support this historical behavior.

39396 Historically, only a single line (and for some terminals, a single line minus 1 column) of  
39397 characters could be entered by the user for the line oriented commands; that is, **:**, **!**, **/**, or **?**.  
39398 IEEE Std 1003.1-200x permits, but does not require, this limitation.

39399 Historically, “soft” errors in *vi* caused the terminal to be alerted, but no error message was  
39400 displayed. As a general rule, no error message was displayed for errors in command execution  
39401 in *vi*, when the error resulted from the user attempting an invalid or impossible action, or when  
39402 a searched-for object was not found. Examples of soft errors included **h** at the left margin,  
39403 **<control>-B** or **[[** at the beginning of the file, **2G** at the end of the file, and so on. In addition,  
39404 errors such as **%**, **[[**, **}]**, **)**, **N**, **n**, **f**, **F**, **t**, and **T** failing to find the searched-for object were soft as well.  
39405 Less consistently, **/** and **?** displayed an error message if the pattern was not found, **/**, **?**, **N**, and **n**  
39406 displayed an error message if no previous regular expression had been specified, and **;** did not  
39407 display an error message if no previous **f**, **F**, **t**, or **T** command had occurred. Also, behavior in  
39408 this area might reasonably be based on a runtime evaluation of the speed of a network  
39409 connection. Finally, some implementations have provided error messages for soft errors in  
39410 order to assist naive users, based on the value of a verbose edit option. IEEE Std 1003.1-200x  
39411 does not list specific errors for which an error message shall be displayed. Implementations  
39412 should conform to historical practice in the absence of any strong reason to diverge.

### 39413 **Page Backwards**

39414 The <control>-B and <control>-F commands historically considered it an error to attempt to  
 39415 page past the beginning or end of the file, whereas the <control>-D and <control>-U commands  
 39416 simply moved to the beginning or end of the file. For consistency, IEEE Std 1003.1-200x requires  
 39417 the latter behavior for all four commands. All four commands still consider it an error if the  
 39418 current line is at the beginning (<control>-B, <control>-U) or end (<control>-F, <control>-D) of  
 39419 the file. Historically, the <control>-B and <control>-F commands skip two lines in order to  
 39420 include overlapping lines when a single command is entered. This makes less sense in the  
 39421 presence of a *count*, as there will be, by definition, no overlapping lines. The actual calculation  
 39422 used by historical implementations of the *vi* editor for <control>-B was:

39423  $((\text{current first line}) - \text{count} \times (\text{window edit option})) + 2$

39424 and for <control>-F was:

39425  $((\text{current first line}) + \text{count} \times (\text{window edit option})) - 2$

39426 This calculation does not work well when intermixing commands with and without counts; for  
 39427 example, **3**<control>-F is not equivalent to entering the <control>-F command three times, and is  
 39428 not reversible by entering the <control>-B command three times. For consistency with other *vi*  
 39429 commands that take counts, IEEE Std 1003.1-200x requires a different calculation.

### 39430 **Scroll Forward**

39431 The 4BSD and System V implementations of *vi* differed on the initial value used by the **scroll**  
 39432 command. 4BSD used:

39433  $((\text{window edit option}) + 1) / 2$

39434 while System V used the value of the **scroll** edit option. The System V version is specified by  
 39435 IEEE Std 1003.1-200x because the standard developers believed that it was more intuitive and  
 39436 permitted the user a method of setting the scroll value initially without also setting the number  
 39437 of lines that are displayed.

### 39438 **Scroll Forward by Line**

39439 Historically, the <control>-E and <control>-Y commands considered it an error if the last and  
 39440 first lines, respectively, were already on the screen. IEEE Std 1003.1-200x requires conformance  
 39441 to historical practice. Historically, the <control>-E and <control>-Y commands had no effect in  
 39442 open mode. For simplicity and consistency of specification, IEEE Std 1003.1-200x requires that  
 39443 they behave as usual, albeit with a single line screen.

### 39444 **Clear and Redisplay**

39445 The historical <control>-L command refreshed the screen exactly as it was supposed to be  
 39446 currently displayed, replacing any '@' characters for lines that had been deleted but not  
 39447 updated on the screen with refreshed '@' characters. The intent of the <control>-L command is  
 39448 to refresh when the screen has been accidentally overwritten; for example, by a **write** command  
 39449 from another user, or modem noise.

39450 **Redraw Screen**

39451 The historical <control>-R command redisplayed only when necessary to update lines that had  
 39452 been deleted but not updated on the screen and that were flagged with '@' characters. There is  
 39453 no requirement that the screen be in any way refreshed if no lines of this form are currently  
 39454 displayed. IEEE Std 1003.1-200x permits implementations to extend this command to refresh  
 39455 lines on the screen flagged with '@' characters because they are too long to be displayed in the  
 39456 current framework; however, the current line and column need not be modified.

39457 **Search for tagstring**

39458 Historically, the first non-<blank> at or after the cursor was the first character, and all  
 39459 subsequent characters that were word characters, up to the end of the line, were included. For  
 39460 example, with the cursor on the leading space or on the '#' character in the text "#bar@", the  
 39461 tag was "#bar". On the character 'b' it was "bar", and on the 'a', it was "ar".  
 39462 IEEE Std 1003.1-200x requires this behavior.

39463 **Replace Text with Results from Shell Command**

39464 Historically, the <, >, and ! commands considered most cursor motions other than line-oriented  
 39465 motions an error; for example, the command >/foo<CR> succeeded, while the command >I  
 39466 failed, even though the text region described by the two commands might be identical. For  
 39467 consistency, all three commands only consider entire lines and not partial lines, and the region is  
 39468 defined as any line that contains a character that was specified by the motion.

39469 **Move to Matching Character**

39470 Other matching characters have been left implementation-defined in order to allow extensions  
 39471 such as matching '<' and '>' for searching HTML, or #ifdef, #else, and #endif for searching C  
 39472 source.

39473 **Repeat Substitution**

39474 IEEE Std 1003.1-200x requires that any c and g flags specified to the previous substitute  
 39475 command be ignored; however, the r flag may still apply, if supported by the implementation.

39476 **Return to Previous (Context or Section)**

39477 The [[, ]], (, ), {, and } commands are all affected by "section boundaries", but in some historical  
 39478 implementations not all of the commands recognize the same section boundaries. This is a bug,  
 39479 not a feature, and a unique section-boundary algorithm was not described for each command.  
 39480 One special case that is preserved is that the sentence command moves to the end of the last line  
 39481 of the edit buffer while the other commands go to the beginning, in order to preserve the  
 39482 traditional character cut semantics of the sentence command. Historically, vi section boundaries  
 39483 at the beginning and end of the edit buffer were the first non-<blank> on the first and last lines  
 39484 of the edit buffer if one exists; otherwise, the last character of the first and last lines of the edit  
 39485 buffer if one exists. To increase consistency with other section locations, this has been simplified  
 39486 by IEEE Std 1003.1-200x to the first character of the first and last lines of the edit buffer, or the  
 39487 first and the last lines of the edit buffer if they are empty.

39488 Sentence boundaries were problematic in the historical vi. They were not only the boundaries as  
 39489 defined for the section and paragraph commands, but they were the first non-<blank> that  
 39490 occurred after those boundaries, as well. Historically, the vi section commands were  
 39491 documented as taking an optional window size as a count preceding the command. This was not  
 39492 implemented in historical versions, so IEEE Std 1003.1-200x requires that the count repeat the  
 39493 command, for consistency with other vi commands.

39494 **Repeat**

39495 Historically, mapped commands other than text input commands could not be repeated using  
39496 the **period** command. IEEE Std 1003.1-200x requires conformance to historical practice.

39497 The restrictions on the interpretation of special characters (for example, <control>-H) in the  
39498 repetition of text input mode commands is intended to match historical practice. For example,  
39499 given the input sequence:

```
39500 iab<control>-H<control>-H<control>-Hdef<escape>
```

39501 the user should be informed of an error when the sequence is first entered, but not during a  
39502 command repetition. The character <control>-T is specifically exempted from this restriction.  
39503 Historical implementations of *vi* ignored <control>-T characters that were input in the original  
39504 command during command repetition. IEEE Std 1003.1-200x prohibits this behavior.

39505 **Find Regular Expression**

39506 Historically, commands did not affect the line searched to or from if the motion command was a  
39507 search (*/*, *?*, **N**, **n**) and the final position was the start/end of the line. There were some special  
39508 cases and *vi* was not consistent. IEEE Std 1003.1-200x does not permit this behavior, for  
39509 consistency. Historical implementations permitted, but were unable to handle searches as  
39510 motion commands that wrapped (that is, due to the edit option **wrapsan**) to the original  
39511 location. IEEE Std 1003.1-200x requires that this behavior be treated as an error.

39512 Historically, the syntax `"/RE/0"` was used to force the command to cut text in line mode.  
39513 IEEE Std 1003.1-200x requires conformance to historical practice.

39514 Historically, in open mode, a **z** specified to a search command redisplayed the current line  
39515 instead of displaying the current screen with the current line highlighted. For consistency and  
39516 simplicity of specification, IEEE Std 1003.1-200x does not permit this behavior.

39517 Historically, trailing **z** commands were permitted and ignored if entered as part of a search used  
39518 as a motion command. For consistency and simplicity of specification, IEEE Std 1003.1-200x does  
39519 not permit this behavior.

39520 **Execute an ex Command**

39521 Historically, *vi* implementations restricted the commands that could be entered on the colon  
39522 command line (for example, **append** and **change**), and some other commands were known to  
39523 cause them to fail catastrophically. For consistency, IEEE Std 1003.1-200x does not permit these  
39524 restrictions. When executing an *ex* command by entering `:`, it is not possible to enter a <newline>  
39525 as part of the command because it is considered the end of the command. A different approach  
39526 is to enter *ex* command mode by using the *vi* **Q** command (and later resuming visual mode with  
39527 the *ex vi* command). In *ex* command mode, the single-line limitation does not exist. So, for  
39528 example, the following is valid:

```
39529 Q
39530 s/break here/break\
39531 here/
39532 vi
```

39533 IEEE Std 1003.1-200x requires that, if the *ex* command overwrites any part of the screen that  
39534 would be erased by a refresh, *vi* pauses for a character from the user. Historically, this character  
39535 could be any character; for example, a character input by the user before the message appeared,  
39536 or even a mapped character. This is probably a bug, but implementations that have tried to be  
39537 more rigorous by requiring that the user enter a specific character, or that the user enter a  
39538 character after the message was displayed, have been forced by user indignation back into

39539 historical behavior. IEEE Std 1003.1-200x requires conformance to historical practice.

#### 39540 **Shift Left (Right)**

39541 Refer to the Rationale for the ! and / commands. Historically, the < and > commands sometimes  
39542 moved the cursor to the first non-<blank> (for example if the command was repeated or with \_  
39543 as the motion command), and sometimes left it unchanged. IEEE Std 1003.1-200x does not  
39544 permit this inconsistency, requiring instead that the cursor always move to the first non-  
39545 <blank>. Historically, the < and > commands did not support buffer arguments, although some  
39546 implementations allow the specification of an optional buffer. This behavior is neither required  
39547 nor disallowed by IEEE Std 1003.1-200x.

#### 39548 **Execute**

39549 Historically, buffers could execute other buffers, and loops, infinite and otherwise, were  
39550 possible. IEEE Std 1003.1-200x requires conformance to historical practice. The *\*buffer* syntax of  
39551 *ex* is not required in *vi*, because it is not historical practice and has been used in some *vi*  
39552 implementations to support additional scripting languages.

#### 39553 **Reverse Case**

39554 Historically, the ~ command ignored any associated *count*, and acted only on the characters in  
39555 the current line. For consistency with other *vi* commands, IEEE Std 1003.1-200x requires that an  
39556 associated *count* act on the next *count* characters, and that the command move to subsequent  
39557 lines if warranted by *count*, to make it possible to modify large pieces of text in a reasonably  
39558 efficient manner. There exist *vi* implementations that optionally require an associated motion  
39559 command for the ~ command. Implementations supporting this functionality are encouraged to  
39560 base it on the **tildedop** edit option and handle the text regions and cursor positioning identically  
39561 to the **yank** command.

#### 39562 **Append**

39563 Historically, *counts* specified to the **A**, **a**, **I**, and **i** commands repeated the input of the first line  
39564 *count* times, and did not repeat the subsequent lines of the input text. IEEE Std 1003.1-200x  
39565 requires that the entire text input be repeated *count* times.

#### 39566 **Move Backward to Preceding Word**

39567 Historically, *vi* became confused if word commands were used as motion commands in empty  
39568 files. IEEE Std 1003.1-200x requires that this be an error. Historical implementations of *vi* had a  
39569 large number of bugs in the word movement commands, and they varied greatly in behavior in  
39570 the presence of empty lines, “words” made up of a single character, and lines containing only  
39571 <blank>s. For consistency and simplicity of specification, IEEE Std 1003.1-200x does not permit  
39572 this behavior.

#### 39573 **Change to End-of-Line**

39574 Some historical implementations of the **C** command did not behave as described by  
39575 IEEE Std 1003.1-200x when the **\$** key was remapped because they were implemented by pushing  
39576 the **\$** key onto the input queue and reprocessing it. IEEE Std 1003.1-200x does not permit this  
39577 behavior. Historically, the **C**, **S**, and **s** commands did not copy replaced text into the numeric  
39578 buffers. For consistency and simplicity of specification, IEEE Std 1003.1-200x requires that they  
39579 behave like their respective **c** commands in all respects.

**39580 Delete**

39581 Historically, lines in open mode that were deleted were scrolled up, and an @ glyph written over  
39582 the beginning of the line. In the case of terminals that are incapable of the necessary cursor  
39583 motions, the editor erased the deleted line from the screen. IEEE Std 1003.1-200x requires  
39584 conformance to historical practice; that is, if the terminal cannot display the '@' character, the  
39585 line cannot remain on the screen.

**39586 Delete to End-of-Line**

39587 Some historical implementations of the **D** command did not behave as described by  
39588 IEEE Std 1003.1-200x when the **\$** key was remapped because they were implemented by pushing  
39589 the **\$** key onto the input queue and reprocessing it. IEEE Std 1003.1-200x does not permit this  
39590 behavior.

**39591 Join**

39592 An historical oddity of *vi* is that the commands **J**, **1J**, and **2J** are all equivalent.  
39593 IEEE Std 1003.1-200x requires conformance to historical practice. The *vi* **J** command is specified  
39594 in terms of the *ex* **join** command with an *ex* command *count* value. The address correction for a  
39595 *count* that is past the end of the edit buffer is necessary for historical compatibility for both *ex*  
39596 and *vi*.

**39597 Mark Position**

39598 Historical practice is that only lowercase letters, plus '``' and ''', could be used to mark a  
39599 cursor position. IEEE Std 1003.1-200x requires conformance to historical practice, but encourages  
39600 implementations to support other characters as marks as well.

**39601 Repeat Regular Expression Find (Forward and Reverse)**

39602 Historically, the **N** and **n** commands could not be used as motion components for the **c**  
39603 command. With the exception of the **cN** command, which worked if the search crossed a line  
39604 boundary, the text region would be discarded, and the user would not be in text input mode. For  
39605 consistency and simplicity of specification, IEEE Std 1003.1-200x does not permit this behavior.

**39606 Insert Empty Line (Below and Above)**

39607 Historically, counts to the **O** and **o** commands were used as the number of physical lines to  
39608 open, if the terminal was dumb and the **slowopen** option was not set. This was intended to  
39609 minimize traffic over slow connections and repainting for dumb terminals. IEEE Std 1003.1-200x  
39610 does not permit this behavior, requiring that a *count* to the open command behave as for other  
39611 text input commands. This change to historical practice was made for consistency, and because a  
39612 superset of the functionality is provided by the **slowopen** edit option.

**39613 Put from Buffer (Following and Before)**

39614 Historically, *counts* to the **p** and **P** commands were ignored if the buffer was a line mode buffer,  
39615 but were (mostly) implemented as described in IEEE Std 1003.1-200x if the buffer was a  
39616 character mode buffer. Because implementations exist that do not have this limitation, and  
39617 because pasting lines multiple times is generally useful, IEEE Std 1003.1-200x requires that *count*  
39618 be supported for all **p** and **P** commands.

39619 Historical implementations of *vi* were widely known to have major problems in the **p** and **P**  
39620 commands, particularly when unusual regions of text were copied into the edit buffer. The  
39621 standard developers viewed these as bugs, and they are not permitted for consistency and

39622 simplicity of specification.

39623 Historically, a **P** or **p** command (or an *ex put* command executed from open or visual mode)  
 39624 executed in an empty file, left an empty line as the first line of the file. For consistency and  
 39625 simplicity of specification, IEEE Std 1003.1-200x does not permit this behavior.

### 39626 **Replace Character**

39627 Historically, the **r** command did not correctly handle the *erase* and *word erase* characters as  
 39628 arguments, nor did it handle an associated *count* greater than 1 with a <carriage-return>  
 39629 argument, for which it replaced *count* characters with a single <newline>. IEEE Std 1003.1-200x  
 39630 does not permit these inconsistencies.

39631 Historically, the **r** command permitted the <control>-V escaping of entered characters, such as  
 39632 <ESC> and the <carriage-return>; however, it required two leading <control>-V characters  
 39633 instead of one. IEEE Std 1003.1-200x requires that this be changed for consistency with the other  
 39634 text input commands of *vi*.

39635 Historically, it is an error to enter the **r** command if there are less than *count* characters at or after  
 39636 the cursor in the line. While a reasonable and unambiguous extension would be to permit the **r**  
 39637 command on empty lines, it would require that too large a *count* be adjusted to match the  
 39638 number of characters at or after the cursor for consistency, which is sufficiently different from  
 39639 historical practice to be avoided. IEEE Std 1003.1-200x requires conformance to historical  
 39640 practice.

### 39641 **Replace Characters**

39642 Historically, if there were **autoindent** characters in the line on which the **R** command was run,  
 39643 and **autoindent** was set, the first <newline> would be properly indented and no characters  
 39644 would be replaced by the <newline>. Each additional <newline> would replace *n* characters,  
 39645 where *n* was the number of characters that were needed to indent the rest of the line to the  
 39646 proper indentation level. This behavior is a bug and is not permitted by IEEE Std 1003.1-200x.

### 39647 **Undo**

39648 Historical practice for cursor positioning after undoing commands was mixed. In most cases,  
 39649 when undoing commands that affected a single line, the cursor was moved to the start of added  
 39650 or changed text, or immediately after deleted text. However, if the user had moved from the line  
 39651 being changed, the column was either set to the first non-<blank>, returned to the origin of the  
 39652 command, or remained unchanged. When undoing commands that affected multiple lines or  
 39653 entire lines, the cursor was moved to the first character in the first line restored. As an example  
 39654 of how inconsistent this was, a search, followed by an **o** text input command, followed by an  
 39655 **undo** would return the cursor to the location where the **o** command was entered, but a **cw**  
 39656 command followed by an **o** command followed by an **undo** would return the cursor to the first  
 39657 non-<blank> of the line. IEEE Std 1003.1-200x requires the most useful of these behaviors, and  
 39658 discards the least useful, in the interest of consistency and simplicity of specification.

39659

**Yank**

39660  
39661  
39662  
39663  
39664  
39665  
39666  
39667  
39668  
39669

Historically, the **yank** command did not move to the end of the motion if the motion was in the forward direction. It moved to the end of the motion if the motion was in the backward direction, except for the **\_** command, or for the **G** and **'** commands when the end of the motion was on the current line. This was further complicated by the fact that for a number of motion commands, the **yank** command moved the cursor but did not update the screen; for example, a subsequent command would move the cursor from the end of the motion, even though the cursor on the screen had not reflected the cursor movement for the **yank** command. IEEE Std 1003.1-200x requires that all **yank** commands associated with backward motions move the cursor to the end of the motion for consistency, and specifically, to make **'** commands as motions consistent with search patterns as motions.

39670

**Yank Current Line**

39671  
39672  
39673  
39674

Some historical implementations of the **Y** command did not behave as described by IEEE Std 1003.1-200x when the **'\_'** key was remapped because they were implemented by pushing the **'\_'** key onto the input queue and reprocessing it. IEEE Std 1003.1-200x does not permit this behavior.

39675

**Redraw Window**

39676  
39677  
39678  
39679  
39680  
39681

Historically, the **z** command always redrew the screen. This is permitted but not required by IEEE Std 1003.1-200x, because of the frequent use of the **z** command in macros such as **map n nz** for screen positioning, instead of its use to change the screen size. The standard developers believed that expanding or scrolling the screen offered a better interface for users. The ability to redraw the screen is preserved if the optional new window size is specified, and in the **<control>-L** and **<control>-R** commands.

39682  
39683  
39684

The semantics of **z^** are confusing at best. Historical practice is that the screen before the screen that ended with the specified line is displayed. IEEE Std 1003.1-200x requires conformance to historical practice.

39685  
39686  
39687  
39688  
39689

Historically, the **z** command would not display a partial line at the top or bottom of the screen. If the partial line would normally have been displayed at the bottom of the screen, the command worked, but the partial line was replaced with **'@'** characters. If the partial line would normally have been displayed at the top of the screen, the command would fail. For consistency and simplicity of specification, IEEE Std 1003.1-200x does not permit this behavior.

39690  
39691

Historically, the **z** command with a line specification of 1 ignored the command. For consistency and simplicity of specification, IEEE Std 1003.1-200x does not permit this behavior.

39692  
39693  
39694

Historically, the **z** command did not set the cursor column to the first non-**<blank>** for the character if the first screen was to be displayed, and was already displayed. For consistency and simplicity of specification, IEEE Std 1003.1-200x does not permit this behavior.

39695

**Input Mode Commands in vi**

39696  
39697  
39698  
39699  
39700

Historical implementations of **vi** did not permit the user to erase more than a single line of input, or to use normal erase characters such as *line erase*, *worderase*, and *erase* to erase **autoindent** characters. As there exist implementations of **vi** that do not have these limitations, both behaviors are permitted, but only historical practice is required. In the case of these extensions, **vi** is required to pause at the **autoindent** and previous line boundaries.

39701  
39702

Historical implementations of **vi** updated only the portion of the screen where the current cursor character was displayed. For example, consider the **vi** input keystrokes:

39703 iabcd<escape>0C<tab>

39704 Historically, the <tab> would overwrite the characters "abcd" when it was displayed. Other  
 39705 implementations replace only the 'a' character with the <tab>, and then push the rest of the  
 39706 characters ahead of the cursor. Both implementations have problems. The historical  
 39707 implementation is probably visually nicer for the above example; however, for the keystrokes:

39708 iabcd<ESC>0R<tab><ESC>

39709 the historical implementation results in the string "bcd" disappearing and then magically  
 39710 reappearing when the <ESC> character is entered. IEEE Std 1003.1-200x requires the former  
 39711 behavior when overwriting erase-columns; that is, overwriting characters that are no longer  
 39712 logically part of the edit buffer, and the latter behavior otherwise.

39713 Historical implementations of *vi* discarded the <control>-D and <control>-T characters when  
 39714 they were entered at places where their command functionality was not appropriate. IEEE Std 1003.1-200x  
 39715 requires that the <control>-T functionality always be available, and that  
 39716 <control>-D be treated as any other key when not operating on **autoindent** characters.

39717 **NUL**

39718 Some historical implementations of *vi* limited the number of characters entered using the NUL  
 39719 input character to 256 bytes. IEEE Std 1003.1-200x permits this limitation; however,  
 39720 implementations are encouraged to remove this limit.

39721 **<control>-D**

39722 See also Rationale for the input mode command <newline>. The hidden assumptions in the  
 39723 <control>-D command (and in the *vi* **autoindent** specification in general) is that <space>s take  
 39724 up a single column on the screen and that <tab>s are comprised of an integral number of  
 39725 <space>s.

39726 **<newline>**

39727 Implementations are permitted to rewrite **autoindent** characters in the line when <newline>,  
 39728 <carriage-return>, <control>-D, and <control>-T are entered, or when the **shift** commands are  
 39729 used, because historical implementations have both done so and found it necessary to do so. For  
 39730 example, a <control>-D when the cursor is preceded by a single <tab>, with **tabstop** set to 8, and  
 39731 **shiftwidth** set to 3, will result in the <tab> being replaced by several <space>s.

39732 **<control>-T**

39733 See also the Rationale for the input mode command <newline>. Historically, <control>-T only  
 39734 worked if no non-<blank>s had yet been input in the current input line. In addition, the  
 39735 characters inserted by <control>-T were treated as **autoindent** characters, and could not be  
 39736 erased using normal user erase characters. Because implementations exist that do not have  
 39737 these limitations, and as moving to a column boundary is generally useful, IEEE Std 1003.1-200x  
 39738 requires that both limitations be removed.

- 39739            **<control>-V**
- 39740            Historically, *vi* used  $\text{^V}$ , regardless of the value of the literal-next character of the terminal.  
39741            IEEE Std 1003.1-200x requires conformance to historical practice.
- 39742            The uses described for **<control>-V** can also be accomplished with **<control>-Q**, which is useful  
39743            on terminals that use **<control>-V** for the down-arrow function. However, most historical  
39744            implementations use **<control>-Q** for the *termios* START character, so the editor will generally  
39745            not receive the **<control>-Q** unless **stty ixon** mode is set to off. (In addition, some historical  
39746            implementations of *vi* explicitly set **ixon** mode to on, so it was difficult for the user to set it to  
39747            off.) Any of the command characters described in IEEE Std 1003.1-200x can be made ineffective  
39748            by their selection as *termios* control characters, using the *stty* utility or other methods described  
39749            in the System Interfaces volume of IEEE Std 1003.1-200x.
- 39750            **<ESC>**
- 39751            Historically, SIGINT alerted the terminal when used to end input mode. This behavior is  
39752            permitted, but not required, by IEEE Std 1003.1-200x.
- 39753            **FUTURE DIRECTIONS**
- 39754            None.
- 39755            **SEE ALSO**
- 39756            *ex*, *stty*
- 39757            **CHANGE HISTORY**
- 39758            First released in Issue 2.
- 39759            **Issue 5**
- 39760            FUTURE DIRECTIONS section added.
- 39761            **Issue 6**
- 39762            This utility is now marked as part of the User Portability Utilities option.
- 39763            The APPLICATION USAGE section is added.
- 39764            The obsolescent SYNOPSIS is removed.
- 39765            The following new requirements on POSIX implementations derive from alignment with the  
39766            Single UNIX Specification:
- 39767
  - The **reindent** command description is added.

39768            The *vi* utility has been extensively rewritten for alignment with the IEEE P1003.2b draft  
39769            standard.

39770            IEEE PASC Interpretations 1003.2 #57, #62, #63, #64, #78, and #188 are applied. |

39771            IEEE PASC Interpretation 1003.2 #207 is applied, clarifying the description of the **R** command in |  
39772            a manner similar to the descriptions of other text input mode commands such as **i**, **o**, and **O**. |

39773 **NAME**

39774           wait — await process completion

39775 **SYNOPSIS**39776           wait [*pid*...]39777 **DESCRIPTION**

39778           When an asynchronous list (see Section 2.9.3.1 (on page 2252)) is started by the shell, the process  
 39779           ID of the last command in each element of the asynchronous list shall become known in the  
 39780           current shell execution environment; see Section 2.12 (on page 2263).

39781           If the *wait* utility is invoked with no operands, it shall wait until all process IDs known to the  
 39782           invoking shell have terminated and exit with a zero exit status.

39783           If one or more *pid* operands are specified that represent known process IDs, the *wait* utility shall  
 39784           wait until all of them have terminated. If one or more *pid* operands are specified that represent  
 39785           unknown process IDs, *wait* shall treat them as if they were known process IDs that exited with  
 39786           exit status 127. The exit status returned by the *wait* utility shall be the exit status of the process  
 39787           requested by the last *pid* operand.

39788           The known process IDs are applicable only for invocations of *wait* in the current shell execution  
 39789           environment.

39790 **OPTIONS**

39791           None.

39792 **OPERANDS**

39793           The following operand shall be supported:

39794           *pid*           One of the following:

39795                           1. The unsigned decimal integer process ID of a command, for which the utility  
 39796                           is to wait for the termination.

39797                           2. A job control job ID (see the Base Definitions volume of IEEE Std 1003.1-200x,  
 39798                           Section 3.203, Job Control Job ID) that identifies a background process group  
 39799                           to be waited for. The job control job ID notation is applicable only for  
 39800                           invocations of *wait* in the current shell execution environment; see Section  
 39801                           2.12 (on page 2263). The exit status of *wait* shall be determined by the last  
 39802                           command in the pipeline.

39803                           **Note:**     The job control job ID type of *pid* is only available on systems supporting  
 39804                           the User Portability Utilities option.

39805 **STDIN**

39806           Not used.

39807 **INPUT FILES**

39808           None.

39809 **ENVIRONMENT VARIABLES**39810           The following environment variables shall affect the execution of *wait*:

39811           *LANG*           Provide a default value for the internationalization variables that are unset or null.  
 39812                           (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
 39813                           Internationalization Variables for the precedence of internationalization variables  
 39814                           used to determine the values of locale categories.)

39815           *LC\_ALL*        If set to a non-empty string value, override the values of all the other  
 39816                           internationalization variables.

- 39817 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 39818 characters (for example, single-byte as opposed to multi-byte characters in  
 39819 arguments).
- 39820 **LC\_MESSAGES**  
 39821 Determine the locale that should be used to affect the format and contents of  
 39822 diagnostic messages written to standard error.
- 39823 **XSI NLS\_PATH** Determine the location of message catalogs for the processing of **LC\_MESSAGES**.
- 39824 **ASYNCHRONOUS EVENTS**
- 39825 Default.
- 39826 **STDOUT**
- 39827 Not used.
- 39828 **STDERR**
- 39829 The standard error shall be used only for diagnostic messages.
- 39830 **OUTPUT FILES**
- 39831 None.
- 39832 **EXTENDED DESCRIPTION**
- 39833 None.
- 39834 **EXIT STATUS**
- 39835 If one or more operands were specified, all of them have terminated or were not known by the  
 39836 invoking shell, and the status of the last operand specified is known, then the exit status of *wait*  
 39837 shall be the exit status information of the command indicated by the last operand specified. If  
 39838 the process terminated abnormally due to the receipt of a signal, the exit status shall be greater  
 39839 than 128 and shall be distinct from the exit status generated by other signals, but the exact value  
 39840 is unspecified. (See the *kill -I* option.) Otherwise, the *wait* utility shall exit with one of the  
 39841 following values:
- 39842 0 The *wait* utility was invoked with no operands and all process IDs known by the  
 39843 invoking shell have terminated.
- 39844 1-126 The *wait* utility detected an error.
- 39845 127 The command identified by the last *pid* operand specified is unknown.
- 39846 **CONSEQUENCES OF ERRORS**
- 39847 Default.
- 39848 **APPLICATION USAGE**
- 39849 On most implementations, *wait* is a shell built-in. If it is called in a subshell or separate utility  
 39850 execution environment, such as one of the following:
- 39851 (wait)  
 39852 nohup wait ...  
 39853 find . -exec wait ... \;
- 39854 it returns immediately because there are no known process IDs to wait for in those  
 39855 environments.
- 39856 Historical implementations of interactive shells have discarded the exit status of terminated  
 39857 background processes before each shell prompt. Therefore, the status of background processes  
 39858 was usually lost unless it terminated while *wait* was waiting for it. This could be a serious  
 39859 problem when a job that was expected to run for a long time actually terminated quickly with a  
 39860 syntax or initialization error because the exit status returned was usually zero if the requested

39861 process ID was not found. This volume of IEEE Std 1003.1-200x requires the implementation to  
 39862 keep the status of terminated jobs available until the status is requested, so that scripts like:

```
39863 j1&
39864 p1=$!
39865 j2&
39866 wait $p1
39867 echo Job 1 exited with status $?
39868 wait $!
39869 echo Job 2 exited with status $?
```

39870 works without losing status on any of the jobs. The shell is allowed to discard the status of any  
 39871 process that it determines the application cannot get the process ID from the shell. It is also  
 39872 required to remember only {CHILD\_MAX} number of processes in this way. Since the only way  
 39873 to get the process ID from the shell is by using the '!' shell parameter, the shell is allowed to  
 39874 discard the status of an asynchronous list if "\$!" was not referenced before another  
 39875 asynchronous list was started. (This means that the shell only has to keep the status of the last  
 39876 asynchronous list started if the application did not reference "\$!". If the implementation of the  
 39877 shell is smart enough to determine that a reference to "\$!" was not saved anywhere that the  
 39878 application can retrieve it later, it can use this information to trim the list of saved information.  
 39879 Note also that a successful call to *wait* with no operands discards the exit status of all  
 39880 asynchronous lists.)

39881 If the exit status of *wait* is greater than 128, there is no way for the application to know if the  
 39882 waited-for process exited with that value or was killed by a signal. Since most utilities exit with  
 39883 small values, there is seldom any ambiguity. Even in the ambiguous cases, most applications  
 39884 just need to know that the asynchronous job failed; it does not matter whether it detected an  
 39885 error and failed or was killed and did not complete its job normally.

#### 39886 EXAMPLES

39887 Although the exact value used when a process is terminated by a signal is unspecified, if it is  
 39888 known that a signal terminated a process, a script can still reliably figure out which signal using  
 39889 *kill* as shown by the following script:

```
39890 sleep 1000&
39891 pid=$!
39892 kill -kill $pid
39893 wait $pid
39894 echo $pid was terminated by a SIG$(kill -l $?) signal.
```

39895 If the following sequence of commands is run in less than 31 seconds:

```
39896 sleep 257 | sleep 31 &
39897 jobs -l %%
```

39898 either of the following commands returns the exit status of the second *sleep* in the pipeline:

```
39899 wait <pid of sleep 31>
39900 wait %%
```

#### 39901 RATIONALE

39902 The description of *wait* does not refer to the *waitpid()* function from the System Interfaces  
 39903 volume of IEEE Std 1003.1-200x because that would needlessly overspecify this interface.  
 39904 However, the wording means that *wait* is required to wait for an explicit process when it is given  
 39905 an argument so that the status information of other processes is not consumed. Historical  
 39906 implementations use the *wait()* function defined in the System Interfaces volume of  
 39907 IEEE Std 1003.1-200x until *wait()* returns the requested process ID or finds that the requested

39908 process does not exist. Because this means that a shell script could not reliably get the status of  
39909 all background children if a second background job was ever started before the first job finished,  
39910 it is recommended that the *wait* utility use a method such as the functionality provided by the  
39911 *waitpid()* function.

39912 The ability to wait for multiple *pid* operands was adopted from the KornShell.

39913 This new functionality was added because it is needed to determine the exit status of any  
39914 asynchronous list accurately. The only compatibility problem that this change creates is for a  
39915 script like

```
39916 while sleep 60 do  
39917     job& echo Job started $(date) as $! done
```

39918 which causes the shell to monitor all of the jobs started until the script terminates or runs out of  
39919 memory. This would not be a problem if the loop did not reference "\$!" or if the script would  
39920 occasionally *wait* for jobs it started.

39921 **FUTURE DIRECTIONS**

39922 None.

39923 **SEE ALSO**

39924 *sh*, the System Interfaces volume of IEEE Std 1003.1-200x, *waitpid()*

39925 **CHANGE HISTORY**

39926 First released in Issue 2.

39927 **NAME**39928 `wc` — word, line, and byte or character count39929 **SYNOPSIS**39930 `wc [-c|-m][-lw][file...]`39931 **DESCRIPTION**39932 The `wc` utility shall read one or more input files and, by default, write the number of <newline>s,  
39933 words, and bytes contained in each input file to the standard output.39934 The utility also shall write a total count for all named files, if more than one input file is  
39935 specified.39936 The `wc` utility shall consider a *word* to be a non-zero-length string of characters delimited by  
39937 white space.39938 **OPTIONS**39939 The `wc` utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2,  
39940 Utility Syntax Guidelines.

39941 The following options shall be supported:

39942 `-c` Write to the standard output the number of bytes in each input file.39943 `-l` Write to the standard output the number of <newline>s in each input file.39944 `-m` Write to the standard output the number of characters in each input file.39945 `-w` Write to the standard output the number of words in each input file.39946 When any option is specified, `wc` shall report only the information requested by the specified  
39947 options.39948 **OPERANDS**

39949 The following operand shall be supported:

39950 *file* A pathname of an input file. If no *file* operands are specified, the standard input  
39951 shall be used.39952 **STDIN**39953 The standard input shall be used only if no *file* operands are specified. See the INPUT FILES  
39954 section.39955 **INPUT FILES**

39956 The input files may be of any type.

39957 **ENVIRONMENT VARIABLES**39958 The following environment variables shall affect the execution of `wc`:39959 *LANG* Provide a default value for the internationalization variables that are unset or null.  
39960 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
39961 Internationalization Variables for the precedence of internationalization variables  
39962 used to determine the values of locale categories.)39963 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
39964 internationalization variables.39965 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
39966 characters (for example, single-byte as opposed to multi-byte characters in  
39967 arguments and input files) and which characters are defined as white space  
39968 characters.

- 39969 **LC\_MESSAGES**  
 39970 Determine the locale that should be used to affect the format and contents of  
 39971 diagnostic messages written to standard error and informative messages written to  
 39972 standard output.
- 39973 XSI **NLSPATH** Determine the location of message catalogs for the processing of **LC\_MESSAGES**.
- 39974 **ASYNCHRONOUS EVENTS**  
 39975 Default.
- 39976 **STDOUT**  
 39977 By default, the standard output shall contain an entry for each input file of the form:  
 39978 "%d %d %d %s\n", <newlines>, <words>, <bytes>, <file>  
 39979 If the **-m** option is specified, the number of characters shall replace the <bytes> field in this  
 39980 format.  
 39981 If any options are specified and the **-l** option is not specified, the number of <newline>s shall  
 39982 not be written.  
 39983 If any options are specified and the **-w** option is not specified, the number of words shall not be  
 39984 written.  
 39985 If any options are specified and neither **-c** nor **-m** is specified, the number of bytes or characters  
 39986 shall not be written.  
 39987 If no input *file* operands are specified, no name shall be written and no <blank>s preceding the  
 39988 pathname shall be written.  
 39989 If more than one input *file* operand is specified, an additional line shall be written, of the same  
 39990 format as the other lines, except that the word **total** (in the POSIX locale) shall be written instead  
 39991 of a pathname and the total of each column shall be written as appropriate. Such an additional  
 39992 line, if any, is written at the end of the output.
- 39993 **STDERR**  
 39994 The standard error shall be used only for diagnostic messages. |
- 39995 **OUTPUT FILES**  
 39996 None.
- 39997 **EXTENDED DESCRIPTION**  
 39998 None.
- 39999 **EXIT STATUS**  
 40000 The following exit values shall be returned:  
 40001 0 Successful completion.  
 40002 >0 An error occurred.
- 40003 **CONSEQUENCES OF ERRORS**  
 40004 Default.

**40005 APPLICATION USAGE**

40006           The **-m** option is not a switch, but an option at the same level as **-c**. Thus, to produce the full  
40007           default output with character counts instead of bytes, the command required is:

40008           wc -mlw

**40009 EXAMPLES**

40010           None.

**40011 RATIONALE**

40012           The output file format pseudo-*printf()* string differs from the System V version of *wc*: |

40013           "%7d%7d%7d %s\n"

40014           which produces possibly ambiguous and unparseable results for very large files, as it assumes no  
40015           number shall exceed six digits.

40016           Some historical implementations use only <space>, <tab>, and <newline> as word separators.  
40017           The equivalent of the ISO C standard *isspace()* function is more appropriate.

40018           The **-c** option stands for “character” count, even though it counts bytes. This stems from the  
40019           sometimes erroneous historical view that bytes and characters are the same size. Due to  
40020           international requirements, the **-m** option (reminiscent of “multi-byte”) was added to obtain  
40021           actual character counts.

40022           Early proposals only specified the results when input files were text files. The current  
40023           specification more closely matches historical practice. (Bytes, words, and <newline>s are  
40024           counted separately and the results are written when an end-of-file is detected.)

40025           Historical implementations of the *wc* utility only accepted one argument to specify the options  
40026           **-c**, **-l**, and **-w**. Some of them also had multiple occurrences of an option cause the  
40027           corresponding count to be written multiple times and had the order of specification of the  
40028           options affect the order of the fields on output, but did not document either of these. Because  
40029           common usage either specifies no options or only one option, and because none of this was  
40030           documented, the changes required by this volume of IEEE Std 1003.1-200x should not break  
40031           many historical applications (and do not break any historical conforming applications). |

**40032 FUTURE DIRECTIONS**

40033           None.

**40034 SEE ALSO**

40035           *cksum*

**40036 CHANGE HISTORY**

40037           First released in Issue 2.

40038 **NAME**40039            what — identify SCCS files (**DEVELOPMENT**)40040 **SYNOPSIS**40041 XSI        what [-s] *file...*

40042

40043 **DESCRIPTION**40044            The *what* utility shall search the given files for all occurrences of the pattern that *get* (see *get* (on page 2675)) substitutes for the %Z% keyword ("@( # ") and shall write to standard output what follows until the first occurrence of one of the following:

40047            "    &gt;    newline    \    NUL

40048 **OPTIONS**40049            The *what* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2, Utility Syntax Guidelines.

40051            The following option shall be supported:

40052            -s            Quit after finding the first occurrence of the pattern in each file.

40053 **OPERANDS**

40054            The following operands shall be supported:

40055            *file*        A pathname of a file to search.40056 **STDIN**

40057            Not used.

40058 **INPUT FILES**

40059            The input files shall be of any file type.

40060 **ENVIRONMENT VARIABLES**40061            The following environment variables shall affect the execution of *what*:40062            *LANG*        Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)40066            *LC\_ALL*     If set to a non-empty string value, override the values of all the other internationalization variables.40068            *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).40071            *LC\_MESSAGES*

40072                        Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

40074            *NLSPATH*   Determine the location of message catalogs for the processing of *LC\_MESSAGES*.40075 **ASYNCHRONOUS EVENTS**

40076            Default.

40077 **STDOUT**40078            The standard output shall consist of the following for each *file* operand:

40079            "%s:\n\t%s\n", &lt;pathname&gt;, &lt;identification string&gt;

40080 **STDERR**

40081           The standard error shall be used only for diagnostic messages.

40082 **OUTPUT FILES**

40083           None.

40084 **EXTENDED DESCRIPTION**

40085           None.

40086 **EXIT STATUS**

40087           The following exit values shall be returned:

40088           0   Any matches were found.

40089           1   Otherwise.

40090 **CONSEQUENCES OF ERRORS**

40091           Default.

40092 **APPLICATION USAGE**

40093           The *what* utility is intended to be used in conjunction with the SCCS command *get*, which automatically inserts identifying information, but it can also be used where the information is inserted by any other means.

40096           When the string "@(#)" is included in a library routine in a shared library, it might not be found in an **a.out** file using that library routine.

40098 **EXAMPLES**

40099           If the C-language program in file **f.c** contains:

```
40100           char ident[] = "@(#)identification information";
```

40101           and **f.c** is compiled to yield **f.o** and **a.out**, then the command:

```
40102           what f.c f.o a.out
```

40103           writes:

```
40104           f.c:
40105                identification information
```

```
40106                ...
```

```
40107           f.o:
40108                identification information
```

```
40109                ...
```

```
40110           a.out:
40111                identification information
```

```
40112                ...
```

40113 **RATIONALE**

40114           None.

40115 **FUTURE DIRECTIONS**

40116           None.

40117 **SEE ALSO**

40118           *get*

40119 **CHANGE HISTORY**

40120           First released in Issue 2.

## 40121 NAME

40122 who — display who is on the system

## 40123 SYNOPSIS

40124 UP who [-mTu]

40125

40126 XSI who [-mu]-s[-bHlprt][file]

40127 who [-mTu][-abdHlprt][file]

40128 who -q [file]

40129 who am i

40130 who am I

40131

## 40132 DESCRIPTION

40133 The *who* utility shall list various pieces of information about accessible users. The domain of  
40134 accessibility is implementation-defined.40135 XSI Based on the options given, *who* can also list the user's name, terminal line, login time, elapsed  
40136 time since activity occurred on the line, and the process ID of the command interpreter for each  
40137 current system user.

## 40138 OPTIONS

40139 The *who* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section  
40140 12.2, Utility Syntax Guidelines.40141 The following options shall be supported. The metavariables, such as *<line>*, refer to fields  
40142 described in the STDOUT section.40143 XSI **-a** Process the implementation-defined database or named file with the **-b**, **-d**, **-l**, **-p**,  
40144 **-r**, **-t**, **-T** and **-u** options turned on.40145 XSI **-b** Write the time and date of the last reboot.40146 XSI **-d** Write a list of all processes that have expired and not been respawned by the *init* |  
40147 system process. The *<exit>* field shall appear for dead processes and contain the |  
40148 termination and exit values of the dead process. This can be useful in determining |  
40149 why a process terminated. |40150 XSI **-H** Write column headings above the regular output.40151 XSI **-l** (The letter ell.) List only those lines on which the system is waiting for someone to |  
40152 login. The *<name>* field shall be **LOGIN** in such cases. Other fields shall be the |  
40153 same as for user entries except that the *<state>* field does not exist. |40154 **-m** Output only information about the current terminal.40155 XSI **-p** List any other process that is currently active and has been previously spawned by  
40156 *init*.40157 XSI **-q** (Quick.) List only the names and the number of users currently logged on. When |  
40158 this option is used, all other options shall be ignored. |40159 XSI **-r** Write the current *run-level* of the *init* process.40160 XSI **-s** List only the *<name>*, *<line>*, and *<time>* fields. This is the default case.40161 XSI **-t** Indicate the last change to the system clock.

40162        **-T**        Show the state of each terminal, as described in the STDOUT section.

40163        **-u**        Write “idle time” for each displayed user in addition to any other information. The  
 40164        idle time is the time since any activity occurred on the user’s terminal. The method  
 40165 XSI        of determining this is unspecified. This option shall list only those users who are  
 40166        currently logged in. The *<name>* is the user’s login name. The *<line>* is the name of  
 40167        the line as found in the directory */dev*. The *<time>* is the time that the user logged  
 40168        in. The *<activity>* is the number of hours and minutes since activity last occurred  
 40169        on that particular line. A dot indicates that the terminal has seen activity in the last  
 40170        minute and is therefore “current”. If more than twenty-four hours have elapsed or  
 40171        the line has not been used since boot time, the entry shall be marked *<old>*. This  
 40172        field is useful when trying to determine whether a person is working at the  
 40173        terminal or not. The *<pid>* is the process ID of the user’s login process.

#### 40174 OPERANDS

40175 XSI        The following operands shall be supported:

40176        **am i, am I**    In the POSIX locale, limit the output to describing the invoking user, equivalent to  
 40177        the **-m** option. The **am** and **i** or **I** must be separate arguments.

40178        **file**        Specify a pathname of a file to substitute for the implementation-defined database  
 40179        of logged-on users that *who* uses by default.

#### 40180 STDIN

40181        Not used.

#### 40182 INPUT FILES

40183        None.

#### 40184 ENVIRONMENT VARIABLES

40185        The following environment variables shall affect the execution of *who*:

40186        **LANG**        Provide a default value for the internationalization variables that are unset or null.  
 40187        (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
 40188        Internationalization Variables for the precedence of internationalization variables  
 40189        used to determine the values of locale categories.)

40190        **LC\_ALL**       If set to a non-empty string value, override the values of all the other  
 40191        internationalization variables.

40192        **LC\_CTYPE**    Determine the locale for the interpretation of sequences of bytes of text data as  
 40193        characters (for example, single-byte as opposed to multi-byte characters in  
 40194        arguments).

40195        **LC\_MESSAGES**

40196        Determine the locale that should be used to affect the format and contents of  
 40197        diagnostic messages written to standard error.

40198        **LC\_TIME**     Determine the locale used for the format and contents of the date and time strings.

40199 XSI        **NLSPATH**        Determine the location of message catalogs for the processing of **LC\_MESSAGES**.

40200        **TZ**         Determine the timezone used when writing date and time information. If **TZ** is  
 40201        unset or null, an unspecified default timezone shall be used.

#### 40202 ASYNCHRONOUS EVENTS

40203        Default.

**40204 STDOUT**

40205 The *who* utility shall write its default format to the standard output in an implementation-  
40206 defined format, subject only to the requirement of containing the information described above.

40207 XSI OF XSI-conformant systems shall write the default information to the standard output in the  
40208 following general format:

```
40209 <name>[<state>]<line><time>[<activity>][<pid>][<comment>][<exit>]
```

40210 The following format shall be used for the **-T** option:

```
40211 "%s %c %s %s\n" <name>, <terminal state>, <terminal name>,  
40212 <time of login>
```

40213 where *<terminal state>* is one of the following characters:

- 40214 + The terminal allows write access to other users.
- 40215 - The terminal denies write access to other users.
- 40216 ? The terminal write-access state cannot be determined.

40217 In the POSIX locale, the *<time of login>* shall be equivalent in format to the output of:

```
40218 date +"%b %e %H:%M"
```

40219 If the **-u** option is used with **-T**, the idle time shall be added to the end of the previous format in  
40220 an unspecified format.

**40221 STDERR**

40222 The standard error shall be used only for diagnostic messages.

**40223 OUTPUT FILES**

40224 None.

**40225 EXTENDED DESCRIPTION**

40226 None.

**40227 EXIT STATUS**

40228 The following exit values shall be returned:

- 40229 0 Successful completion.
- 40230 >0 An error occurred.

**40231 CONSEQUENCES OF ERRORS**

40232 Default.

**40233 APPLICATION USAGE**

40234 The name *init* used for the system process is the most commonly used on historical systems, but  
40235 it may vary.

40236 The “domain of accessibility” referred to is a broad concept that permits interpretation either on  
40237 a very secure basis or even to allow a network-wide implementation like the historical *rwho*.

**40238 EXAMPLES**

40239 None.

**40240 RATIONALE**

40241 Due to differences between historical implementations, the base options provided were a  
40242 compromise to allow users to work with those functions. The standard developers also  
40243 considered removing all the options, but felt that these options offered users valuable  
40244 functionality. Additional options to match historical systems are available on XSI-conformant

- 40245 systems.
- 40246 It is recognized that the *who* command may be of limited usefulness, especially in a multi-level  
40247 secure environment. The standard developers considered, however, that having some standard  
40248 method of determining the “accessibility” of other users would aid user portability.
- 40249 No format was specified for the default *who* output for systems not supporting the XSI  
40250 Extension. In such a user-oriented command, designed only for human use, this was not  
40251 considered to be a deficiency.
- 40252 The format of the terminal name is unspecified, but the descriptions of *ps*, *talk*, and *write* require  
40253 that they use the same format. |
- 40254 It is acceptable for an implementation to produce no output for an invocation of *who mil*. |
- 40255 **FUTURE DIRECTIONS**
- 40256 None.
- 40257 **SEE ALSO**
- 40258 *msg*
- 40259 **CHANGE HISTORY**
- 40260 First released in Issue 2.
- 40261 **Issue 6**
- 40262 This utility is now marked as part of the User Portability Utilities option.
- 40263 The *TZ* entry is added to the ENVIRONMENT VARIABLES section.

## 40264 NAME

40265 write — write to another user

## 40266 SYNOPSIS

40267 UP write *user\_name* [*terminal*]

40268

## 40269 DESCRIPTION

40270 The *write* utility shall read lines from the user's standard input and write them to the terminal of  
40271 another user. When first invoked, it shall write the message:40272 **Message from** *sender-login-id* (*sending-terminal*) [*date*]...40273 to *user\_name*. When it has successfully completed the connection, the sender's terminal shall be  
40274 alerted twice to indicate that what the sender is typing is being written to the recipient's  
40275 terminal.

40276 If the recipient wants to reply, this can be accomplished by typing:

40277 write *sender-login-id* [*sending-terminal*]40278 upon receipt of the initial message. Whenever a line of input as delimited by a NL, EOF, or EOL  
40279 special character (see the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General  
40280 Terminal Interface) is accumulated while in canonical input mode, the accumulated data shall be  
40281 written on the other user's terminal. Characters shall be processed as follows:

- 40282 • Typing <alert> shall write the alert character to the recipient's terminal.
- 40283 • Typing the erase and kill characters shall affect the sender's terminal in the manner described  
40284 by the **termios** interface in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11,  
40285 General Terminal Interface.
- 40286 • Typing the interrupt or end-of-file characters shall cause *write* to write an appropriate  
40287 message ("EOT\n" in the POSIX locale) to the recipient's terminal and exit.
- 40288 • Typing characters from *LC\_CTYPE* classifications **print** or **space** shall cause those characters  
40289 to be sent to the recipient's terminal.
- 40290 • When and only when the *stty iexten* local mode is enabled, the existence and processing of  
40291 additional special control characters and multi-byte or single-byte functions is  
40292 implementation-defined.
- 40293 • Typing other non-printable characters shall cause implementation-defined sequences of  
40294 printable characters to be written to the recipient's terminal.

40295 To write to a user who is logged in more than once, the *terminal* argument can be used to indicate  
40296 which terminal to write to; otherwise, the recipient's terminal is selected in an implementation-  
40297 defined manner and an informational message is written to the sender's standard output,  
40298 indicating which terminal was chosen.40299 Permission to be a recipient of a *write* message can be denied or granted by use of the *mesg*  
40300 utility. However, a user's privilege may further constrain the domain of accessibility of other  
40301 users' terminals. The *write* utility shall fail when the user lacks the appropriate privileges to  
40302 perform the requested action.

## 40303 OPTIONS

40304 None.

40305 **OPERANDS**

40306 The following operands shall be supported:

40307 *user\_name* Login name of the person to whom the message shall be written. The application  
40308 shall ensure that this operand is of the form returned by the *who* utility.

40309 *terminal* Terminal identification in the same format provided by the *who* utility.

40310 **STDIN**

40311 Lines to be copied to the recipient's terminal is read from standard input.

40312 **INPUT FILES**

40313 None.

40314 **ENVIRONMENT VARIABLES**

40315 The following environment variables shall affect the execution of *write*:

40316 *LANG* Provide a default value for the internationalization variables that are unset or null.  
40317 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
40318 Internationalization Variables for the precedence of internationalization variables  
40319 used to determine the values of locale categories.)

40320 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
40321 internationalization variables.

40322 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
40323 characters (for example, single-byte as opposed to multi-byte characters in  
40324 arguments and input files). If the recipient's locale does not use an *LC\_CTYPE*  
40325 equivalent to the sender's, the results are undefined.

40326 *LC\_MESSAGES*

40327 Determine the locale that should be used to affect the format and contents of  
40328 diagnostic messages written to standard error and informative messages written to  
40329 standard output.

40330 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

40331 **ASYNCHRONOUS EVENTS**

40332 If an interrupt signal is received, *write* shall write an appropriate message on the recipient's  
40333 terminal and exits with a status of zero. It shall take the standard action for all other signals.

40334 **STDOUT**

40335 An informational message shall be written to standard output if a recipient is logged in more  
40336 than once.

40337 **STDERR**

40338 The standard error shall be used only for diagnostic messages.

40339 **OUTPUT FILES**

40340 The recipient's terminal is used for output.

40341 **EXTENDED DESCRIPTION**

40342 None.

40343 **EXIT STATUS**

40344 The following exit values shall be returned:

40345 0 Successful completion.

40346 >0 The addressed user is not logged on or the addressed user denies permission.

40347 **CONSEQUENCES OF ERRORS**

40348 Default.

40349 **APPLICATION USAGE**40350 The *talk* utility is considered by some users to be a more usable utility on full-screen terminals.40351 **EXAMPLES**

40352 None.

40353 **RATIONALE**

40354 The *write* utility was included in this volume of IEEE Std 1003.1-200x since it can be  
40355 implemented on all terminal types. The standard developers considered the *talk* utility, which  
40356 cannot be implemented on certain terminals, to be a “better” communications interface. Both of  
40357 these programs are in widespread use on historical implementations. Therefore, the standard  
40358 developers decided that both utilities should be specified.

40359 The format of the terminal name is unspecified, but the descriptions of *ps*, *talk*, *who*, and *write*  
40360 require that they all use or accept the same format.

40361 **FUTURE DIRECTIONS**

40362 None.

40363 **SEE ALSO**

40364 *mesg*, *talk*, *who*, the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General  
40365 Terminal Interface

40366 **CHANGE HISTORY**

40367 First released in Issue 2.

40368 **Issue 5**

40369 FUTURE DIRECTIONS section added.

40370 **Issue 6**

40371 This utility is now marked as part of the User Portability Utilities option.

40372 The normative text is reworded to avoid use of the term “must” for application requirements.

## 40373 NAME

40374 xargs — construct argument lists and invoke utility

## 40375 SYNOPSIS

```
40376 xsi xargs [-t][-p][-E eofstr][-I replstr][-L number][-n number [-x]]
40377 [-s size][utility [argument...]]
```

## 40378 DESCRIPTION

40379 The *xargs* utility shall construct a command line consisting of the *utility* and *argument* operands  
 40380 specified followed by as many arguments read in sequence from standard input as fit in length  
 40381 and number constraints specified by the options. The *xargs* utility shall then invoke the  
 40382 constructed command line and wait for its completion. This sequence shall be repeated until one  
 40383 of the following occurs:

- 40384 • An end-of-file condition is detected on standard input.
- 40385 • The logical end-of-file string (see the **-E eofstr** option) is found on standard input after  
 40386 double-quote processing, apostrophe processing, and backslash escape processing (see next  
 40387 paragraph).
- 40388 • An invocation of a constructed command line returns an exit status of 255.

40389 The application shall ensure that arguments in the standard input are separated by unquoted  
 40390 <blank>s, unescaped <blank>s or <newline>s. A string of zero or more non-double-quote ( ' ' )  
 40391 characters and non-<newline>s can be quoted by enclosing them in double-quotes. A string of  
 40392 zero or more non-apostrophe ( ' \ ' ' ) characters and non-<newline>s can be quoted by enclosing  
 40393 them in apostrophes. Any unquoted character can be escaped by preceding it with a backslash.  
 40394 The utility named by *utility* shall be executed one or more times until the end-of-file is reached  
 40395 or the logical end-of file string is found. The results are unspecified if the utility named by *utility*  
 40396 attempts to read from its standard input.

40397 The generated command line length shall be the sum of the size in bytes of the utility name and  
 40398 each argument treated as strings, including a null byte terminator for each of these strings. The  
 40399 *xargs* utility shall limit the command line length such that when the command line is invoked,  
 40400 the combined argument and environment lists (see the *exec* family of functions in the System  
 40401 Interfaces volume of IEEE Std 1003.1-200x) shall not exceed {ARG\_MAX}-2 048 bytes. Within  
 40402 this constraint, if neither the **-n** nor the **-s** option is specified, the default command line length  
 40403 shall be at least {LINE\_MAX}.

## 40404 OPTIONS

40405 The *xargs* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section  
 40406 12.2, Utility Syntax Guidelines.

40407 The following options shall be supported:

40408 **-E eofstr** Use *eofstr* as the logical end-of-file string. If **-E** is not specified, it is unspecified  
 40409 whether the logical end-of-file string is the underscore character ( ' \_ ' ) or the end-  
 40410 of-file string capability is disabled. When *eofstr* is the null string, the logical end-  
 40411 of-file string capability shall be disabled and underscore characters shall be taken  
 40412 literally.

40413 xsi **-I replstr** Insert mode: *utility* is executed for each line from standard input, taking the entire  
 40414 line as a single argument, inserting it in *arguments* for each occurrence of *replstr*. A  
 40415 maximum of five arguments in *arguments* can each contain one or more instances  
 40416 of *replstr*. Any <blank>s at the beginning of each line shall be ignored.  
 40417 Constructed arguments cannot grow larger than 255 bytes. Option **-x** shall be  
 40418 forced on.

- 40419 XSI **-L number** The *utility* shall be executed for each non-empty *number* lines of arguments from  
 40420 standard input. The last invocation of *utility* shall be with fewer lines of arguments  
 40421 if fewer than *number* remain. A line is considered to end with the first <newline>  
 40422 unless the last character of the line is a <blank>; a trailing <blank> signals  
 40423 continuation to the next non-empty line, inclusive. The **-L** and **-n** options are  
 40424 mutually-exclusive; the last one specified shall take effect.
- 40425 **-n number** Invoke *utility* using as many standard input arguments as possible, up to *number* (a  
 40426 positive decimal integer) arguments maximum. Fewer arguments shall be used if:
- The command line length accumulated exceeds the size specified by the **-s**  
 40427 option (or {LINE\_MAX} if there is no **-s** option).
  - The last iteration has fewer than but not zero, operands remaining.  
 40429
- 40430 **-p** Prompt mode: the user is asked whether to execute *utility* at each invocation. Trace  
 40431 mode (**-t**) is turned on to write the command instance to be executed, followed by  
 40432 a prompt to standard error. An affirmative response read from */dev/tty* shall  
 40433 execute the command; otherwise, that particular invocation of *utility* shall be  
 40434 skipped.
- 40435 **-s size** Invoke *utility* using as many standard input arguments as possible yielding a  
 40436 command line length less than *size* (a positive decimal integer) bytes. Fewer  
 40437 arguments shall be used if:
- The total number of arguments exceeds that specified by the **-n** option.  
 40438
  - The total number of lines exceeds that specified by the **-L** option.  
 40439 XSI
  - End-of-file is encountered on standard input before *size* bytes are accumulated.  
 40440
- 40441 Values of *size* up to at least {LINE\_MAX} bytes shall be supported, provided that  
 40442 the constraints specified in the DESCRIPTION are met. It shall not be considered  
 40443 an error if a value larger than that supported by the implementation or exceeding  
 40444 the constraints specified in the DESCRIPTION is given; *xargs* shall use the largest  
 40445 value it supports within the constraints.
- 40446 **-t** Enable trace mode. Each generated command line shall be written to standard  
 40447 error just prior to invocation.
- 40448 **-x** Terminate if a command line containing *number* arguments (see the **-n** option  
 40449 XSI above) or *number* lines (see the **-L** option above) will not fit in the implied or  
 40450 specified size (see the **-s** option above).

#### 40451 OPERANDS

40452 The following operands shall be supported:

- 40453 *utility* The name of the utility to be invoked, found by search path using the *PATH*  
 40454 environment variable, described in the Base Definitions volume of  
 40455 IEEE Std 1003.1-200x, Chapter 8, Environment Variables. If *utility* is omitted, the  
 40456 default shall be the *echo* utility. If the *utility* operand names any of the special  
 40457 built-in utilities in Section 2.14 (on page 2266), the results are undefined.
- 40458 *argument* An initial option or operand for the invocation of *utility*.

#### 40459 STDIN

40460 The standard input shall be a text file. The results are unspecified if an end-of-file condition is  
 40461 detected immediately following an escaped <newline>.

40462 **INPUT FILES**

40463           The file `/dev/tty` shall be used to read responses required by the `-p` option. |

40464 **ENVIRONMENT VARIABLES**

40465           The following environment variables shall affect the execution of *xargs*:

40466           **LANG**       Provide a default value for the internationalization variables that are unset or null.  
40467                       (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
40468                       Internationalization Variables for the precedence of internationalization variables  
40469                       used to determine the values of locale categories.)

40470           **LC\_ALL**     If set to a non-empty string value, override the values of all the other  
40471                       internationalization variables.

40472           **LC\_COLLATE**

40473                       Determine the locale for the behavior of ranges, equivalence classes and multi-  
40474                       character collating elements used in the extended regular expression defined for  
40475                       the **yesexpr** locale keyword in the *LC\_MESSAGES* category.

40476           **LC\_CTYPE**   Determine the locale for the interpretation of sequences of bytes of text data as  
40477                       characters (for example, single-byte as opposed to multi-byte characters in  
40478                       arguments and input files) and the behavior of character classes used in the  
40479                       extended regular expression defined for the **yesexpr** locale keyword in the  
40480                       *LC\_MESSAGES* category.

40481           **LC\_MESSAGES**

40482                       Determine the locale for the processing of affirmative responses and that should be  
40483                       used to affect the format and contents of diagnostic messages written to standard  
40484                       error.

40485 **XSI**           **NLS\_PATH**   Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

40486           **PATH**       Determine the location of *utility*, as described in the Base Definitions volume of  
40487                       IEEE Std 1003.1-200x, Chapter 8, Environment Variables.

40488 **ASYNCHRONOUS EVENTS**

40489           Default.

40490 **STDOUT**

40491           Not used.

40492 **STDERR**

40493           The standard error shall be used for diagnostic messages and the `-t` and `-p` options. If the `-t` |  
40494           option is specified, the *utility* and its constructed argument list shall be written to standard error,  
40495           as it will be invoked, prior to invocation. If `-p` is specified, a prompt of the following format  
40496           shall be written (in the POSIX locale):

40497           " ? . . . "

40498           at the end of the line of the output from `-t`.

40499 **OUTPUT FILES**

40500           None.

40501 **EXTENDED DESCRIPTION**

40502           None.

40503 **EXIT STATUS**

40504 The following exit values shall be returned:

- 40505           0    All invocations of *utility* returned exit status zero.
- 40506           1-125   A command line meeting the specified requirements could not be assembled, one or more of the invocations of *utility* returned a non-zero exit status, or some other error occurred.
- 40507
- 40508
- 40509           126    The utility specified by *utility* was found but could not be invoked.
- 40510           127    The utility specified by *utility* could not be found.

40511 **CONSEQUENCES OF ERRORS**

40512           If a command line meeting the specified requirements cannot be assembled, the utility cannot be invoked, an invocation of the utility is terminated by a signal, or an invocation of the utility exits with exit status 255, the *xargs* utility shall write a diagnostic message and exit without processing any remaining input.

40513

40514

40515

40516 **APPLICATION USAGE**

40517           The 255 exit status allows a utility being used by *xargs* to tell *xargs* to terminate if it knows no further invocations using the current data stream will succeed. Thus, *utility* should explicitly *exit* with an appropriate value to avoid accidentally returning with 255.

40518

40519

40520           Note that input is parsed as lines; <blank>s separate arguments. If *xargs* is used to bundle output of commands like *find dir -print* or *ls* into commands to be executed, unexpected results are likely if any filenames contain any <blank>s or <newline>s. This can be fixed by using *find* to call a script that converts each file found into a quoted string that is then piped to *xargs*. Note that the quoting rules used by *xargs* are not the same as in the shell. They were not made consistent here because existing applications depend on the current rules and the shell syntax is not fully compatible with it. An easy rule that can be used to transform any string into a quoted form that *xargs* interprets correctly is to precede each character in the string with a backslash.

40521

40522

40523

40524

40525

40526

40527

40528           On implementations with a large value for {ARG\_MAX}, *xargs* may produce command lines longer than {LINE\_MAX}. For invocation of utilities, this is not a problem. If *xargs* is being used to create a text file, users should explicitly set the maximum command line length with the *-s* option.

40529

40530

40531

40532           The *command*, *env*, *nice*, *nohup*, *time*, and *xargs* utilities have been specified to use exit code 127 if an error occurs so that applications can distinguish “failure to find a utility” from “invoked utility exited with an error indication”. The value 127 was chosen because it is not commonly used for other meanings; most utilities use small values for “normal error conditions” and the values above 128 can be confused with termination due to receipt of a signal. The value 126 was chosen in a similar manner to indicate that the utility could be found, but not invoked. Some scripts produce meaningful error messages differentiating the 126 and 127 cases. The distinction between exit codes 126 and 127 is based on KornShell practice that uses 127 when all attempts to *exec* the utility fail with [ENOENT], and uses 126 when any attempt to *exec* the utility fails for any other reason.

40533

40534

40535

40536

40537

40538

40539

40540

40541

40542 **EXAMPLES**

- 40543           1. The following command combines the output of the parenthesised commands onto one line, which is then written to the end-of-file **log**:
- 40544
- 40545            (logname; date; printf "%s\n" "\$0 \$\*") | xargs >>log
- 40546           2. The following command invokes *diff* with successive pairs of arguments originally typed as command line arguments (assuming there are no embedded <blank>s in the elements of the original argument list):
- 40547
- 40548

40549 `printf "%s\n" "$*" | xargs -n 2 -x diff`

40550 3. In the following commands, the user is asked which files in the current directory are to be  
40551 archived. The files are archived into **arch**; *a*, one at a time, or *b*, many at a time.

40552 `a. ls | xargs -p -L 1 ar -r arch`

40553 `b. ls | xargs -p -L 1 | xargs ar -r arch`

40554 4. The following executes with successive pairs of arguments originally typed as command  
40555 line arguments:

40556 `echo $* | xargs -n 2 diff`

40557 5. On XSI-conformant systems, the following moves all files from directory **\$1** to directory **\$2**,  
40558 and echoes each move command just before doing it:

40559 `ls $1 | xargs -I {} -t mv $1/{ } $2/{ }`

#### 40560 RATIONALE

40561 The *xargs* utility was usually found only in System V-based systems; BSD systems included an  
40562 *apply* utility that provided functionality similar to *xargs -n number*. The SVID lists *xargs* as a  
40563 software development extension. This volume of IEEE Std 1003.1-200x does not share the view  
40564 that it is used only for development, and therefore it is not optional.

40565 The classic application of the *xargs* utility is in conjunction with the *find* utility to reduce the  
40566 number of processes launched by a simplistic use of the *find-exec* combination. The *xargs* utility  
40567 is also used to enforce an upper limit on memory required to launch a process. With this basis in  
40568 mind, this volume of IEEE Std 1003.1-200x selected only the minimal features required.

40569 Although the 255 exit status is mostly an accident of historical implementations, it allows a  
40570 utility being used by *xargs* to tell *xargs* to terminate if it knows no further invocations using the  
40571 current data stream shall succeed. Any non-zero exit status from a utility falls into the 1-125  
40572 range when *xargs* exits. There is no statement of how the various non-zero utility exit status  
40573 codes are accumulated by *xargs*. The value could be the addition of all codes, their highest  
40574 value, the last one received, or a single value such as 1. Since no algorithm is arguably better  
40575 than the others, and since many of the standard utilities say little more (portably) than  
40576 “pass/fail”, no new algorithm was invented.

40577 Several other *xargs* options were withdrawn because simple alternatives already exist within this  
40578 volume of IEEE Std 1003.1-200x. For example, the *-i replstr* option can be just as efficiently  
40579 performed using a shell *for* loop. Since *xargs* calls an *exec* function with each input line, the *-i*  
40580 option does not usually exploit the grouping capabilities of *xargs*.

40581 The requirement that *xargs* never produce command lines such that invocation of *utility* is  
40582 within 2 048 bytes of hitting the POSIX *exec* {ARG\_MAX} limitations is intended to guarantee  
40583 that the invoked utility has room to modify its environment variables and command line  
40584 arguments and still be able to invoke another utility. Note that the minimum {ARG\_MAX}  
40585 allowed by the System Interfaces volume of IEEE Std 1003.1-200x is 4 096 bytes and the  
40586 minimum value allowed by the this volume of IEEE Std 1003.1-200x is 2 048 bytes; therefore, the  
40587 2 048 bytes difference seems reasonable. Note, however, that *xargs* may never be able to invoke a  
40588 utility if the environment passed in to *xargs* comes close to using {ARG\_MAX} bytes.

40589 The version of *xargs* required by this volume of IEEE Std 1003.1-200x is required to wait for the  
40590 completion of the invoked command before invoking another command. This was done because  
40591 historical scripts using *xargs* assumed sequential execution. Implementations wanting to provide  
40592 parallel operation of the invoked utilities are encouraged to add an option enabling parallel  
40593 invocation, but should still wait for termination of all of the children before *xargs* terminates  
40594 normally.

40595 The `-e` option was omitted from the ISO POSIX-2:1993 standard in the belief that the `eofstr`  
40596 option-argument was recognized only when it was on a line by itself and before quote and  
40597 escape processing were performed, and that the logical end-of-file processing was only enabled  
40598 if a `-e` option was specified. In that case, a simple `sed` script could be used to duplicate the `-e`  
40599 functionality. Further investigation revealed that:

40600     • The logical end-of-file string was checked for after quote and escape processing, making a `sed`  
40601       script that provided equivalent functionality much more difficult to write.

40602     • The default was to perform logical end-of-file processing with an underscore as the logical  
40603       end-of-file string.

40604 To correct this misunderstanding, the `-E eofstr` option was adopted from the X/Open Portability  
40605 Guide. Users should note that the description of the `-E` option matches historical documentation  
40606 of the `-e` option (which was not adopted because it did not support the Utility Syntax  
40607 Guidelines), by saying that if `eofstr` is the null string, logical end-of-file processing is disabled.  
40608 Historical implementations of `xargs` actually did not disable logical end-of-file processing; they  
40609 treated a null argument found in the input as a logical end-of-file string. (A null `string` argument  
40610 could be generated using single or double quotes ( ' ' or " "). Since this behavior was not  
40611 documented historically, it is considered to be a bug.

#### 40612 **FUTURE DIRECTIONS**

40613     None.

#### 40614 **SEE ALSO**

40615     *echo*

#### 40616 **CHANGE HISTORY**

40617     First released in Issue 2.

#### 40618 **Issue 5**

40619     Second FUTURE DIRECTION added.

#### 40620 **Issue 6**

40621     The obsolescent `-e`, `-i`, and `-l` options are removed.

40622     The following new requirements on POSIX implementations derive from alignment with the  
40623     Single UNIX Specification:

40624     • The `-p` option is added.

40625     • In the INPUT FILES section, the file `/dev/tty` is used to read responses required by the `-p`  
40626       option.

40627     • The STDERR section is updated to describe the `-p` option.

40628     The description of the `-E` option is aligned with the ISO POSIX-2: 1993 standard.

40629     The normative text is reworded to avoid use of the term “must” for application requirements.

## 40630 NAME

40631 yacc — yet another compiler compiler (DEVELOPMENT)

## 40632 SYNOPSIS

40633 CD yacc [-dltv][-b *file\_prefix*][-p *sym\_prefix*] *grammar*

40634

## 40635 DESCRIPTION

40636 The *yacc* utility shall read a description of a context-free grammar in *file* and write C source code,  
 40637 conforming to the ISO C standard, to a code file, and optionally header information into a  
 40638 header file, in the current directory. The C code shall define a function and related routines and  
 40639 macros for an automaton that executes a parsing algorithm meeting the requirements in  
 40640 **Algorithms** (on page 3272).

40641 The form and meaning of the grammar are described in the EXTENDED DESCRIPTION section.

40642 The C source code and header file shall be produced in a form suitable as input for the C  
 40643 compiler (see *c99* (on page 2413)).

## 40644 OPTIONS

40645 The *yacc* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section  
 40646 12.2, Utility Syntax Guidelines.

40647 The following options shall be supported:

40648 **-b *file\_prefix*** Use *file\_prefix* instead of *y* as the prefix for all output filenames. The code file  
 40649 *y.tab.c*, the header file *y.tab.h* (created when **-d** is specified), and the description  
 40650 file *y.output* (created when **-v** is specified), shall be changed to *file\_prefix.tab.c*,  
 40651 *file\_prefix.tab.h*, and *file\_prefix.output*, respectively.

40652 **-d** Write the header file; by default only the code file is written. The **#define**  
 40653 statements that associate the token codes assigned by *yacc* with the user-declared  
 40654 token names. This allows source files other than *y.tab.c* to access the token codes.

40655 **-l** Produce a code file that does not contain any **#line** constructs. If this option is not  
 40656 present, it is unspecified whether the code file or header file contains **#line**  
 40657 directives. This should only be used after the grammar and the associated actions  
 40658 are fully debugged.

40659 **-p *sym\_prefix*** Use *sym\_prefix* instead of *yy* as the prefix for all external names produced by *yacc*.  
 40660 The names affected shall include the functions *yyparse()*, *yylex()*, and *yyerror()*,  
 40661 and the variables *yyval*, *yychar*, and *yydebug*. (In the remainder of this section, the  
 40662 six symbols cited are referenced using their default names only as a notational  
 40663 convenience.) Local names may also be affected by the **-p** option; however, the **-p**  
 40664 option shall not affect **#define** symbols generated by *yacc*.

40665 **-t** Modify conditional compilation directives to permit compilation of debugging  
 40666 code in the code file. Runtime debugging statements shall always be contained in  
 40667 the code file, but by default conditional compilation directives prevent their  
 40668 compilation.

40669 **-v** Write a file containing a description of the parser and a report of conflicts  
 40670 generated by ambiguities in the grammar.

## 40671 OPERANDS

40672 The following operand is required:

40673 *grammar* A pathname of a file containing instructions, hereafter called *grammar*, for which a  
 40674 parser is to be created. The format for the grammar is described in the EXTENDED

40675 DESCRIPTION section.

40676 **STDIN**

40677 Not used.

40678 **INPUT FILES**

40679 The file *grammar* shall be a text file formatted as specified in the EXTENDED DESCRIPTION

40680 section.

40681 **ENVIRONMENT VARIABLES**

40682 The following environment variables shall affect the execution of *yacc*:

40683 *LANG* Provide a default value for the internationalization variables that are unset or null.

40684 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,

40685 Internationalization Variables for the precedence of internationalization variables

40686 used to determine the values of locale categories.)

40687 *LC\_ALL* If set to a non-empty string value, override the values of all the other

40688 internationalization variables.

40689 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as

40690 characters (for example, single-byte as opposed to multi-byte characters in

40691 arguments and input files).

40692 *LC\_MESSAGES*

40693 Determine the locale that should be used to affect the format and contents of

40694 diagnostic messages written to standard error.

40695 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

40696 The *LANG* and *LC\_\** variables affect the execution of the *yacc* utility as stated. The *main()*

40697 function defined in **Yacc Library** (on page 3272) shall call:

40698 `setlocale(LC_ALL, " ")`

40699 and thus, the program generated by *yacc* also shall be affected by the contents of these variables

40700 at runtime.

40701 **ASYNCHRONOUS EVENTS**

40702 Default.

40703 **STDOUT**

40704 Not used.

40705 **STDERR**

40706 If shift/reduce or reduce/reduce conflicts are detected in *grammar*, *yacc* shall write a report of |

40707 those conflicts to the standard error in an unspecified format. |

40708 Standard error shall also be used for diagnostic messages. |

40709 **OUTPUT FILES**

40710 The code file, the header file, and the description file shall be text files. All are described in the

40711 following sections.

40712 **Code File**

40713 This file shall contain the C source code for the `yyparse()` function. It shall contain code for the  
 40714 various semantic actions with macro substitution performed on them as described in the  
 40715 EXTENDED DESCRIPTION section. It also shall contain a copy of the `#define` statements in the  
 40716 header file. If a `%union` declaration is used, the declaration for `YYSTYPE` shall be also included  
 40717 in this file.

40718 **Header File**

40719 The header file shall contain `#define` statements that associate the token numbers with the token  
 40720 names. This allows source files other than the code file to access the token codes. If a `%union`  
 40721 declaration is used, the declaration for `YYSTYPE` and an `extern YYSTYPE yylval` declaration shall  
 40722 be also included in this file.

40723 **Description File**

40724 The description file shall be a text file containing a description of the state machine  
 40725 corresponding to the parser, using an unspecified format. Limits for internal tables (see **Limits**  
 40726 (on page 3273)) shall also be reported, in an implementation-defined manner. (Some  
 40727 implementations may use dynamic allocation techniques and have no specific limit values to  
 40728 report.)

40729 **EXTENDED DESCRIPTION**

40730 The `yacc` command accepts a language that is used to define a grammar for a target language to  
 40731 be parsed by the tables and code generated by `yacc`. The language accepted by `yacc` as a  
 40732 grammar for the target language is described below using the `yacc` input language itself.

40733 The input *grammar* includes rules describing the input structure of the target language and code  
 40734 to be invoked when these rules are recognized to provide the associated semantic action. The  
 40735 code to be executed shall appear as bodies of text that are intended to be C-language code. The  
 40736 C-language inclusions are presumed to form a correct function when processed by `yacc` into its  
 40737 output files. The code included in this way shall be executed during the recognition of the target  
 40738 language.

40739 Given a grammar, the `yacc` utility generates the files described in the OUTPUT FILES section.  
 40740 The code file can be compiled and linked using `c99`. If the declaration and programs sections of  
 40741 the grammar file did not include definitions of `main()`, `yylex()`, and `yyerror()`, the compiled  
 40742 output requires linking with externally supplied version of those functions. Default versions of  
 40743 `main()` and `yyerror()` are supplied in the `yacc` library and can be linked in by using the `-ly`  
 40744 operand to `c99`. The `yacc` library interfaces need not support interfaces with other than the  
 40745 default `yy` symbol prefix. The application provides the lexical analyzer function, `yylex()`; the `lex`  
 40746 utility is specifically designed to generate such a routine.

40747 **Input Language**

40748 The application shall ensure that every specification file consists of three sections in order:  
 40749 *declarations*, *grammar rules*, and *programs*, separated by double percent signs ("`%%`"). The  
 40750 declarations and programs sections can be empty. If the latter is empty, the preceding "`%%`"  
 40751 mark separating it from the rules section can be omitted.

40752 The input is free form text following the structure of the grammar defined below.

40753 **Lexical Structure of the Grammar**

40754 The <blank>s, <newline>s, and <form-feed>s shall be ignored, except that the application shall  
 40755 ensure that they do not appear in names or multi-character reserved symbols. Comments shall  
 40756 be enclosed in `"/ * . . . */`, and can appear wherever a name is valid.

40757 Names are of arbitrary length, made up of letters, periods ('.'), underscores ('\_'), and non-  
 40758 initial digits. Uppercase and lowercase letters are distinct. Conforming applications shall not  
 40759 use names beginning in `yy` or `YY` since the `yacc` parser uses such names. Many of the names  
 40760 appear in the final output of `yacc`, and thus they should be chosen to conform with any  
 40761 additional rules created by the C compiler to be used. In particular they appear in `#define`  
 40762 statements.

40763 A literal shall consist of a single character enclosed in single-quotes ('\''). All of the escape  
 40764 sequences supported for character constants by the ISO C standard shall be supported by `yacc`.

40765 The relationship with the lexical analyzer is discussed in detail below.

40766 The application shall ensure that the NUL character is not used in grammar rules or literals.

40767 **Declarations Section**

40768 The declarations section is used to define the symbols used to define the target language and  
 40769 their relationship with each other. In particular, much of the additional information required to  
 40770 resolve ambiguities in the context-free grammar for the target language is provided here.

40771 Usually `yacc` assigns the relationship between the symbolic names it generates and their  
 40772 underlying numeric value. The declarations section makes it possible to control the assignment  
 40773 of these values.

40774 It is also possible to keep semantic information associated with the tokens currently on the parse  
 40775 stack in a user-defined C-language **union**, if the members of the union are associated with the  
 40776 various names in the grammar. The declarations section provides for this as well.

40777 The first group of declarators below all take a list of names as arguments. That list can optionally  
 40778 be preceded by the name of a C union member (called a *tag* below) appearing within '`<`' and  
 40779 '`>`'. (As an exception to the typographical conventions of the rest of this volume of  
 40780 IEEE Std 1003.1-200x, in this case `<tag>` does not represent a metavariable, but the literal angle  
 40781 bracket characters surrounding a symbol.) The use of *tag* specifies that the tokens named on this  
 40782 line shall be of the same C type as the union member referenced by *tag*. This is discussed in  
 40783 more detail below.

40784 For lists used to define tokens, the first appearance of a given token can be followed by a  
 40785 positive integer (as a string of decimal digits). If this is done, the underlying value assigned to it  
 40786 for lexical purposes shall be taken to be that number.

40787 The following declares *name* to be a token:

```
40788 token [<tag>] name [number][name [number]]...
```

40789 If *tag* is present, the C type for all tokens on this line shall be declared to be the type referenced  
 40790 by *tag*. If a positive integer, *number*, follows a *name*, that value shall be assigned to the token.

40791 The following declares *name* to be a token, and assigns precedence to it:

```
40792 %left [<tag>] name [number][name [number]]...
```

```
40793 %right [<tag>] name [number][name [number]]...
```

40794 One or more lines, each beginning with one of these symbols, can appear in this section. All  
 40795 tokens on the same line have the same precedence level and associativity; the lines are in order

40796 of increasing precedence or binding strength. **%left** denotes that the operators on that line are  
 40797 left associative, and **%right** similarly denotes right associative operators. If *tag* is present, it shall  
 40798 declare a C type for *names* as described for **%token**.

40799 The following declares *name* to be a token, and indicates that this cannot be used associatively:

```
40800 %nonassoc [<tag>] name [number][name [number]]...
```

40801 If the parser encounters associative use of this token it reports an error. If *tag* is present, it shall  
 40802 declare a C type for *names* as described for **%token**.

40803 The following declares that union member *names* are non-terminals, and thus it is required to  
 40804 have a *tag* field at its beginning:

```
40805 %type <tag> name...
```

40806 Because it deals with non-terminals only, assigning a token number or using a literal is also  
 40807 prohibited. If this construct is present, *yacc* shall perform type checking; if this construct is not  
 40808 present, the parse stack shall hold only the **int** type.

40809 Every name used in *grammar* not defined by a **%token**, **%left**, **%right**, or **%nonassoc** declaration  
 40810 is assumed to represent a non-terminal symbol. The *yacc* utility shall report an error for any  
 40811 non-terminal symbol that does not appear on the left side of at least one grammar rule.

40812 Once the type, precedence, or token number of a name is specified, it shall not be changed. If the  
 40813 first declaration of a token does not assign a token number, *yacc* shall assign a token number.  
 40814 Once this assignment is made, the token number shall not be changed by explicit assignment.

40815 The following declarators do not follow the previous pattern.

40816 The following declares the non-terminal *name* to be the *start symbol*, which represents the largest,  
 40817 most general structure described by the grammar rules:

```
40818 %start name
```

40819 By default, it is the left-hand side of the first grammar rule; this default can be overridden with  
 40820 this declaration.

40821 The following declares the *yacc* value stack to be a union of the various types of values desired:

```
40822 %union { body of union (in C) }
```

40823 By default, the values returned by actions (see below) and the lexical analyzer shall be of type  
 40824 **int**. The *yacc* utility keeps track of types, and it shall insert corresponding union member names  
 40825 in order to perform strict type checking of the resulting parser.

40826 Alternatively, given that at least one *<tag>* construct is used, the union can be declared in a  
 40827 header file (which shall be included in the declarations section by using an **#include** construct  
 40828 within **%{** and **%}**), and a **typedef** used to define the symbol YYSTYPE to represent this union.  
 40829 The effect of **%union** is to provide the declaration of YYSTYPE directly from the *yacc* input.

40830 C-language declarations and definitions can appear in the declarations section, enclosed by the  
 40831 following marks:

```
40832 %{ ... %}
```

40833 These statements shall be copied into the code file, and have global scope within it so that they  
 40834 can be used in the rules and program sections.

40835 The application shall ensure that the declarations section is terminated by the token **%%**.

40836 **Grammar Rules in yacc**

40837 The rules section defines the context-free grammar to be accepted by the function *yacc* generates,  
 40838 and associates with those rules C-language actions and additional precedence information. The  
 40839 grammar is described below, and a formal definition follows.

40840 The rules section is comprised of one or more grammar rules. A grammar rule has the form:

40841 A : BODY ;

40842 The symbol **A** represents a non-terminal name, and **BODY** represents a sequence of zero or  
 40843 more *names*, *literals*, and *semantic actions* that can then be followed by optional *precedence rules*.  
 40844 Only the names and literals participate in the formation of the grammar; the semantic actions  
 40845 and precedence rules are used in other ways. The colon and the semicolon are *yacc* punctuation.  
 40846 If there are several successive grammar rules with the same left-hand side, the vertical bar '|' can  
 40847 be used to avoid rewriting the left-hand side; in this case the semicolon appears only after  
 40848 the last rule. The BODY part can be empty (or empty of names and literals) to indicate that the  
 40849 non-terminal symbol matches the empty string.

40850 The *yacc* utility assigns a unique number to each rule. Rules using the vertical bar notation are  
 40851 distinct rules. The number assigned to the rule appears in the description file.

40852 The elements comprising a BODY are:

40853 *name, literal* These form the rules of the grammar: *name* is either a *token* or a *non-terminal*; *literal*  
 40854 stands for itself (less the lexically required quotation marks).

40855 *semantic action*

40856 With each grammar rule, the user can associate actions to be performed each time  
 40857 the rule is recognized in the input process. (Note that the word "action" can also  
 40858 refer to the actions of the parser—shift, reduce, and so on.)

40859 These actions can return values and can obtain the values returned by previous  
 40860 actions. These values are kept in objects of type YYSTYPE (see **%union**). The  
 40861 result value of the action shall be kept on the parse stack with the left-hand side of  
 40862 the rule, to be accessed by other reductions as part of their right-hand side. By  
 40863 using the *<tag>* information provided in the declarations section, the code  
 40864 generated by *yacc* can be strictly type checked and contain arbitrary information. In  
 40865 addition, the lexical analyzer can provide the same kinds of values for tokens, if  
 40866 desired.

40867 An action is an arbitrary C statement and as such can do input or output, call  
 40868 subprograms and alter external variables. An action is one or more C statements  
 40869 enclosed in curly braces '{' and '}'.

40870 Certain pseudo-variables can be used in the action. These are macros for access to  
 40871 data structures known internally to *yacc*.

40872 \$\$ The value of the action can be set by assigning it to \$\$ . If type  
 40873 checking is enabled and the type of the value to be assigned cannot  
 40874 be determined, a diagnostic message may be generated.

40875 \$*number* This refers to the value returned by the component specified by the  
 40876 token *number* in the right side of a rule, reading from left to right;  
 40877 *number* can be zero or negative. If it is, it refers to the data associated  
 40878 with the name on the parser's stack preceding the leftmost symbol of  
 40879 the current rule. (That is, "\$0" refers to the name immediately  
 40880 preceding the leftmost name in the current rule, to be found on the  
 40881 parser's stack and "\$-1" refers to the symbol to its left.) If *number*

40882 refers to an element past the current point in the rule, or beyond the  
 40883 bottom of the stack, the result is undefined. If type checking is  
 40884 enabled and the type of the value to be assigned cannot be  
 40885 determined, a diagnostic message may be generated.

40886 `$<tag>number`

40887 These correspond exactly to the corresponding symbols without the  
 40888 *tag* inclusion, but allow for strict type checking (and preclude  
 40889 unwanted type conversions). The effect is that the macro is expanded  
 40890 to use *tag* to select an element from the YYSTYPE union (using  
 40891 *dataname.tag*). This is particularly useful if *number* is not positive.

40892 `$<tag>$`

40893 This imposes on the reference the type of the union member  
 40894 referenced by *tag*. This construction is applicable when a reference  
 40895 to a left context value occurs in the grammar, and provides yacc with  
 a means for selecting a type.

40896 Actions can occur anywhere in a rule (not just at the end); an action can access  
 40897 values returned by actions to its left, and in turn the value it returns can be  
 40898 accessed by actions to its right. An action appearing in the middle of a rule shall be  
 40899 equivalent to replacing the action with a new non-terminal symbol and adding an  
 40900 empty rule with that non-terminal symbol on the left-hand side. The semantic  
 40901 action associated with the new rule shall be equivalent to the original action. The  
 40902 use of actions within rules might introduce conflicts that would not otherwise  
 40903 exist.

40904 By default, the value of a rule shall be the value of the first element in it. If the first  
 40905 element does not have a type (particularly in the case of a literal) and type  
 40906 checking is turned on by `%type` an error message shall result.

40907 *precedence* The keyword `%prec` can be used to change the precedence level associated with a  
 40908 particular grammar rule. Examples of this are in cases where a unary and binary  
 40909 operator have the same symbolic representation, but need to be given different  
 40910 precedences, or where the handling of an ambiguous if-else construction is  
 40911 necessary. The reserved symbol `%prec` can appear immediately after the body of  
 40912 the grammar rule and can be followed by a token name or a literal. It shall cause  
 40913 the precedence of the grammar rule to become that of the following token name or  
 40914 literal. The action for the rule as a whole can follow `%prec`.

40915 If a program section follows, the application shall ensure that the grammar rules are terminated  
 40916 by `%%`.

#### 40917 **Programs Section**

40918 The *programs* section can include the definition of the lexical analyzer `yylex()`, and any other  
 40919 functions, for example those used in the actions specified in the grammar rules. It is unspecified  
 40920 whether the programs section precedes or follows the semantic actions in the output file;  
 40921 therefore, if the application contains any macro definitions and declarations intended to apply to  
 40922 the code in the semantic actions, it shall place them within "`%{ . . . %}`" in the declarations  
 40923 section.

40924       **Input Grammar**

40925       The following input to *yacc* yields a parser for the input to *yacc*. This formal syntax takes  
40926       precedence over the preceding text syntax description.

40927       The lexical structure is defined less precisely; **Lexical Structure of the Grammar** (on page 3264)  
40928       defines most terms. The correspondence between the previous terms and the tokens below is as  
40929       follows.

40930       **IDENTIFIER**     This corresponds to the concept of *name*, given previously. It also includes  
40931       literals as defined previously.

40932       **C\_IDENTIFIER**   This is a name, and additionally it is known to be followed by a colon. A literal  
40933       cannot yield this token.

40934       **NUMBER**         A string of digits (a non-negative decimal integer).

40935       **TYPE, LEFT, MARK, LCURL, RCURL**

40936       These correspond directly to **%type**, **%left**, **%%**, **%{**, and **%}**.

40937       **{...}**            This indicates C-language source code, with the possible inclusion of '\$'  
40938       macros as discussed previously.

40939       /\* Grammar for the input to yacc. \*/  
40940       /\* Basic entries. \*/  
40941       /\* The following are recognized by the lexical analyzer. \*/

40942       %token     IDENTIFIER         /\* Includes identifiers and literals \*/  
40943       %token     C\_IDENTIFIER        /\* identifier (but not literal)  
40944                                        followed by a :. \*/  
40945       %token     NUMBER             /\* [0-9][0-9]\* \*/

40946       /\* Reserved words : %type=>TYPE %left=>LEFT, and so on \*/

40947       %token     LEFT RIGHT NONASSOC TOKEN PREC TYPE START UNION

40948       %token     MARK                /\* The %% mark. \*/  
40949       %token     LCURL               /\* The %{ mark. \*/  
40950       %token     RCURL               /\* The %} mark. \*/

40951       /\* 8-bit character literals stand for themselves; \*/  
40952       /\* tokens have to be defined for multi-byte characters. \*/

40953       %start     spec

40954       %%

40955       spec     : defs MARK rules tail  
40956                ;  
40957       tail     : MARK  
40958                {  
40959                /\* In this action, set up the rest of the file. \*/  
40960                }  
40961                | /\* Empty; the second MARK is optional. \*/  
40962                ;  
40963       defs     : /\* Empty. \*/  
40964                |        defs def  
40965                ;  
40966       def     : START IDENTIFIER  
40967                |        UNION

```

40968     {
40969     /* Copy union definition to output. */
40970     }
40971     |    LCURL
40972     {
40973     /* Copy C code to output file. */
40974     }
40975     |    RCURL
40976     |    rword tag nlist
40977     ;
40978     rword : TOKEN
40979     |    LEFT
40980     |    RIGHT
40981     |    NONASSOC
40982     |    TYPE
40983     ;
40984     tag  : /* Empty: union tag ID optional. */
40985     |    '<' IDENTIFIER '>'
40986     ;
40987     nlist : nmno
40988     |    nlist nmno
40989     ;
40990     nmno  : IDENTIFIER          /* Note: literal invalid with % type. */
40991     |    IDENTIFIER NUMBER    /* Note: invalid with % type. */
40992     ;

40993     /* Rule section */

40994     rules : C_IDENTIFIER rbody prec
40995     |    rules rule
40996     ;
40997     rule  : C_IDENTIFIER rbody prec
40998     |    '|' rbody prec
40999     ;
41000     rbody : /* empty */
41001     |    rbody IDENTIFIER
41002     |    rbody act
41003     ;
41004     act   : '{'
41005     |    {
41006     |        /* Copy action, translate $$, and so on. */
41007     |    }
41008     |    '}'
41009     ;
41010     prec  : /* Empty */
41011     |    PREC IDENTIFIER
41012     |    PREC IDENTIFIER act
41013     |    prec ';'
41014     ;

```

41015 **Conflicts**

41016 The parser produced for an input grammar may contain states in which conflicts occur. The  
41017 conflicts occur because the grammar is not LALR(1). An ambiguous grammar always contains at  
41018 least one LALR(1) conflict. The yacc utility shall resolve all conflicts, using either default rules or  
41019 user-specified precedence rules.

41020 Conflicts are either shift/reduce conflicts or reduce/reduce conflicts. A shift/reduce conflict is  
41021 where, for a given state and lookahead symbol, both a shift action and a reduce action are  
41022 possible. A reduce/reduce conflict is where, for a given state and lookahead symbol, reductions  
41023 by two different rules are possible.

41024 The rules below describe how to specify what actions to take when a conflict occurs. Not all  
41025 shift/reduce conflicts can be successfully resolved this way because the conflict may be due to  
41026 something other than ambiguity, so incautious use of these facilities can cause the language  
41027 accepted by the parser to be much different from that which was intended. The description file  
41028 shall contain sufficient information to understand the cause of the conflict. Where ambiguity is  
41029 the reason either the default or explicit rules should be adequate to produce a working parser.

41030 The declared precedences and associativities (see **Declarations Section** (on page 3264)) are used  
41031 to resolve parsing conflicts as follows:

- 41032 1. A precedence and associativity is associated with each grammar rule; it is the precedence  
41033 and associativity of the last token or literal in the body of the rule. If the **%prec** keyword is  
41034 used, it overrides this default. Some grammar rules might not have both precedence and  
41035 associativity.
- 41036 2. If there is a shift/reduce conflict, and both the grammar rule and the input symbol have  
41037 precedence and associativity associated with them, then the conflict is resolved in favor of  
41038 the action (shift or reduce) associated with the higher precedence. If the precedences are  
41039 the same, then the associativity is used; left associative implies reduce, right associative  
41040 implies shift, and non-associative implies an error in the string being parsed.
- 41041 3. When there is a shift/reduce conflict that cannot be resolved by rule 2, the shift is done.  
41042 Conflicts resolved this way are counted in the diagnostic output described in **Error**  
41043 **Handling**.
- 41044 4. When there is a reduce/reduce conflict, a reduction is done by the grammar rule that  
41045 occurs earlier in the input sequence. Conflicts resolved this way are counted in the  
41046 diagnostic output described in **Error Handling**.

41047 Conflicts resolved by precedence or associativity shall not be counted in the shift/reduce and  
41048 reduce/reduce conflicts reported by yacc on either standard error or in the description file.

41049 **Error Handling**

41050 The token **error** shall be reserved for error handling. The name **error** can be used in grammar  
41051 rules. It indicates places where the parser can recover from a syntax error. The default value of  
41052 **error** shall be 256. Its value can be changed using a **%token** declaration. The lexical analyzer  
41053 should not return the value of **error**.

41054 The parser shall detect a syntax error when it is in a state where the action associated with the  
41055 lookahead symbol is **error**. A semantic action can cause the parser to initiate error handling by  
41056 executing the macro YYERROR. When YYERROR is executed, the semantic action passes  
41057 control back to the parser. YYERROR cannot be used outside of semantic actions.

41058 When the parser detects a syntax error, it normally calls `yyerror()` with the character string  
41059 "syntax error" as its argument. The call shall not be made if the parser is still recovering

41060 from a previous error when the error is detected. The parser is considered to be recovering from  
 41061 a previous error until the parser has shifted over at least three normal input symbols since the  
 41062 last error was detected or a semantic action has executed the macro *yyerror*. The parser shall not  
 41063 call *yyerror*() when YYERROR is executed.

41064 The macro function YYRECOVERING shall return 1 if a syntax error has been detected and the  
 41065 parser has not yet fully recovered from it. Otherwise, zero shall be returned.

41066 When a syntax error is detected by the parser, the parser shall check if a previous syntax error  
 41067 has been detected. If a previous error was detected, and if no normal input symbols have been  
 41068 shifted since the preceding error was detected, the parser checks if the lookahead symbol is an  
 41069 endmarker (see **Interface to the Lexical Analyzer**). If it is, the parser shall return with a non-  
 41070 zero value. Otherwise, the lookahead symbol shall be discarded and normal parsing shall  
 41071 resume.

41072 When YYERROR is executed or when the parser detects a syntax error and no previous error has  
 41073 been detected, or at least one normal input symbol has been shifted since the previous error was  
 41074 detected, the parser shall pop back one state at a time until the parse stack is empty or the  
 41075 current state allows a shift over **error**. If the parser empties the parse stack, it shall return with a  
 41076 non-zero value. Otherwise, it shall shift over **error** and then resume normal parsing. If the parser  
 41077 reads a lookahead symbol before the error was detected, that symbol shall still be the lookahead  
 41078 symbol when parsing is resumed.

41079 The macro *yyerror* in a semantic action shall cause the parser to act as if it has fully recovered  
 41080 from any previous errors. The macro *yyclearin* shall cause the parser to discard the current  
 41081 lookahead token. If the current lookahead token has not yet been read, *yyclearin* shall have no  
 41082 effect.

41083 The macro YYACCEPT shall cause the parser to return with the value zero. The macro  
 41084 YYABORT shall cause the parser to return with a non-zero value.

#### 41085 **Interface to the Lexical Analyzer**

41086 The *yylex*() function is an integer-valued function that returns a *token number* representing the  
 41087 kind of token read. If there is a value associated with the token returned by *yylex*() (see the  
 41088 discussion of *tag* above), it shall be assigned to the external variable *yylval*.

41089 If the parser and *yylex*() do not agree on these token numbers, reliable communication between |  
 41090 them cannot occur. For (single-byte character) literals, the token is simply the numeric value of |  
 41091 the character in the current character set. The numbers for other tokens can either be chosen by |  
 41092 *yacc*, or chosen by the user. In either case, the **#define** construct of C is used to allow *yylex*() to  
 41093 return these numbers symbolically. The **#define** statements are put into the code file, and the  
 41094 header file if that file is requested. The set of characters permitted by *yacc* in an identifier is larger  
 41095 than that permitted by C. Token names found to contain such characters shall not be included in  
 41096 the **#define** declarations.

41097 If the token numbers are chosen by *yacc*, the tokens other than literals shall be assigned numbers  
 41098 greater than 256, although no order is implied. A token can be explicitly assigned a number by  
 41099 following its first appearance in the declarations section with a number. Names and literals not  
 41100 defined this way retain their default definition. All token numbers assigned by *yacc* shall be  
 41101 unique and distinct from the token numbers used for literals and user-assigned tokens. If  
 41102 duplicate token numbers cause conflicts in parser generation, *yacc* shall report an error;  
 41103 otherwise, it is unspecified whether the token assignment is accepted or an error is reported.

41104 The end of the input is marked by a special token called the *endmarker*, which has a token  
 41105 number that is zero or negative. (These values are invalid for any other token.) All lexical  
 41106 analyzers shall return zero or negative as a token number upon reaching the end of their input. If

41107 the tokens up to, but excluding, the endmarker form a structure that matches the start symbol,  
41108 the parser shall accept the input. If the endmarker is seen in any other context, it shall be  
41109 considered an error.

### 41110 **Completing the Program**

41111 In addition to *yyparse()* and *yylex()*, the functions *yyerror()* and *main()* are required to make a  
41112 complete program. The application can supply *main()* and *yyerror()*, or those routines can be  
41113 obtained from the yacc library.

### 41114 **Yacc Library**

41115 The following functions shall appear only in the yacc library accessible through the `-ly` operand |  
41116 to *c99*; they can therefore be redefined by a conforming application: |

#### 41117 **int main(void)**

41118 This function shall call *yyparse()* and exit with an unspecified value. Other actions within  
41119 this function are unspecified.

#### 41120 **int yyerror(const char \*s)**

41121 This function shall write the NUL-terminated argument to standard error, followed by a  
41122 `<newline>`.

41123 The order of the `-ly` and `-ll` operands given to *c99* is significant; the application shall either  
41124 provide its own *main()* function or ensure that `-ly` precedes `-ll`.

### 41125 **Debugging the Parser**

41126 The parser generated by yacc shall have diagnostic facilities in it that can be optionally enabled  
41127 at either compile time or at runtime (if enabled at compile time). The compilation of the runtime  
41128 debugging code is under the control of YYDEBUG, a preprocessor symbol. If YYDEBUG has a  
41129 non-zero value, the debugging code shall be included. If its value is zero, the code shall not be  
41130 included.

41131 In parsers where the debugging code has been included, the external **int** *yydebug* can be used to  
41132 turn debugging on (with a non-zero value) and off (zero value) at runtime. The initial value of  
41133 *yydebug* shall be zero.

41134 When `-t` is specified, the code file shall be built such that, if YYDEBUG is not already defined at  
41135 compilation time (using the *c99* `-D YYDEBUG` option, for example), YYDEBUG shall be set  
41136 explicitly to 1. When `-t` is not specified, the code file shall be built such that, if YYDEBUG is not  
41137 already defined, it shall be set explicitly to zero.

41138 The format of the debugging output is unspecified but includes at least enough information to  
41139 determine the shift and reduce actions, and the input symbols. It also provides information  
41140 about error recovery.

### 41141 **Algorithms**

41142 The parser constructed by yacc implements an LALR(1) parsing algorithm as documented in the  
41143 literature. It is unspecified whether the parser is table-driven or direct-coded.

41144 A parser generated by yacc shall never request an input symbol from *yylex()* while in a state  
41145 where the only actions other than the error action are reductions by a single rule.

41146 The literature of parsing theory defines these concepts.

41147 **Limits**

41148 The `yacc` utility may have several internal tables. The minimum maximums for these tables are  
 41149 shown in the following table. The exact meaning of these values is implementation-defined. The  
 41150 implementation shall define the relationship between these values and between them and any  
 41151 error messages that the implementation may generate should it run out of space for any internal  
 41152 structure. An implementation may combine groups of these resources into a single pool as long  
 41153 as the total available to the user does not fall below the sum of the sizes specified by this section.

41154 **Table 4-22** Internal Limits in `yacc`

Limit	Minimum Maximum	Description
{NTERMS}	126	Number of tokens.
{NNONTERM}	200	Number of non-terminals.
{NPROD}	300	Number of rules.
{NSTATES}	600	Number of states.
{MEMSIZE}	5 200	Length of rules. The total length, in names (tokens and non-terminals), of all the rules of the grammar. The left-hand side is counted for each rule, even if it is not explicitly repeated, as specified in <b>Grammar Rules in yacc</b> (on page 3266).
{ACTSIZE}	4 000	Number of actions. “Actions” here (and in the description file) refer to parser actions (shift, reduce, and so on) not to semantic actions defined in <b>Grammar Rules in yacc</b> (on page 3266).

41172 **EXIT STATUS**

41173 The following exit values shall be returned:

41174 0 Successful completion.

41175 >0 An error occurred.

41176 **CONSEQUENCES OF ERRORS**

41177 If any errors are encountered, the run is aborted and `yacc` exits with a non-zero status. Partial  
 41178 code files and header files may be produced. The summary information in the description  
 41179 file always shall be produced if the `-v` flag is present.

41180 **APPLICATION USAGE**

41181 Historical implementations experience name conflicts on the names `yacc.tmp`, `yacc.acts`,  
 41182 `yacc.debug`, `y.tab.c`, `y.tab.h`, and `y.output` if more than one copy of `yacc` is running in a single  
 41183 directory at one time. The `-b` option was added to overcome this problem. The related problem  
 41184 of allowing multiple `yacc` parsers to be placed in the same file was addressed by adding a `-p`  
 41185 option to override the previously hard-coded `yy` variable prefix.

41186 The description of the `-p` option specifies the minimal set of function and variable names that  
 41187 cause conflict when multiple parsers are linked together. `YYSTYPE` does not need to be changed.  
 41188 Instead, the programmer can use `-b` to give the header files for different parsers different names,  
 41189 and then the file with the `yylex()` for a given parser can include the header for that parser.  
 41190 Names such as `yyclearerr` do not need to be changed because they are used only in the actions;  
 41191 they do not have linkage. It is possible that an implementation has other names, either internal  
 41192 ones for implementing things such as `yyclearerr`, or providing non-standard features that it  
 41193 wants to change with `-p`.

41194 Unary operators that are the same token as a binary operator in general need their precedence  
 41195 adjusted. This is handled by the `%prec` advisory symbol associated with the particular grammar  
 41196 rule defining that unary operator. (See **Grammar Rules in yacc** (on page 3266).) Applications  
 41197 are not required to use this operator for unary operators, but the grammars that do not require it  
 41198 are rare.

#### 41199 EXAMPLES

41200 Access to the `yacc` library is obtained with library search operands to `c99`. To use the `yacc` library  
 41201 `main()`:

```
41202 c99 y.tab.c -l y
```

41203 Both the `lex` library and the `yacc` library contain `main()`. To access the `yacc main()`:

```
41204 c99 y.tab.c lex.yy.c -l y -l l
```

41205 This ensures that the `yacc` library is searched first, so that its `main()` is used.

41206 The historical `yacc` libraries have contained two simple functions that are normally coded by the  
 41207 application programmer. These functions are similar to the following code:

```
41208 #include <locale.h>
41209 int main(void)
41210 {
41211     extern int yyparse();
41212     setlocale(LC_ALL, "");
41213     /* If the following parser is one created by lex, the
41214        application must be careful to ensure that LC_CTYPE
41215        and LC_COLLATE are set to the POSIX locale. */
41216     (void) yyparse();
41217     return (0);
41218 }
41219 #include <stdio.h>
41220 int yyerror(const char *msg)
41221 {
41222     (void) fprintf(stderr, "%s\n", msg);
41223     return (0);
41224 }
```

#### 41225 RATIONALE

41226 The references in **Referenced Documents** (on page xx) may be helpful in constructing the parser  
 41227 generator. The referenced DeRemer and Pennello article (along with the works it references)  
 41228 describes a technique to generate parsers that conform to this volume of IEEE Std 1003.1-200x.  
 41229 Work in this area continues to be done, so implementors should consult current literature before  
 41230 doing any new implementations. The original Knuth article is the theoretical basis for this kind  
 41231 of parser, but the tables it generates are impractically large for reasonable grammars and should  
 41232 not be used. The “equivalent to” wording is intentional to assure that the best tables that are  
 41233 LALR(1) can be generated.

41234 There has been confusion between the class of grammars, the algorithms needed to generate  
 41235 parsers, and the algorithms needed to parse the languages. They are all reasonably orthogonal.  
 41236 In particular, a parser generator that accepts the full range of LR(1) grammars need not generate  
 41237 a table any more complex than one that accepts SLR(1) (a relatively weak class of LR grammars)  
 41238 for a grammar that happens to be SLR(1). Such an implementation need not recognize the case,  
 41239 either; table compression can yield the SLR(1) table (or one even smaller than that) without

41240 recognizing that the grammar is SLR(1). The speed of an LR(1) parser for any class is dependent  
 41241 more upon the table representation and compression (or the code generation if a direct parser is  
 41242 generated) than upon the class of grammar that the table generator handles.

41243 The speed of the parser generator is somewhat dependent upon the class of grammar it handles.  
 41244 However, the original Knuth article algorithms for constructing LR parsers was judged by its  
 41245 author to be impractically slow at that time. Although full LR is more complex than LALR(1), as  
 41246 computer speeds and algorithms improve, the difference (in terms of acceptable wall-clock  
 41247 execution time) is becoming less significant.

41248 Potential authors are cautioned that the referenced DeRemer and Pennello article previously  
 41249 cited identifies a bug (an over-simplification of the computation of LALR(1) lookahead sets) in  
 41250 some of the LALR(1) algorithm statements that preceded it to publication. They should take the  
 41251 time to seek out that paper, as well as current relevant work, particularly Aho's.

41252 The **-b** option was added to provide a portable method for permitting yacc to work on multiple  
 41253 separate parsers in the same directory. If a directory contains more than one yacc grammar, and  
 41254 both grammars are constructed at the same time (by, for example, a parallel *make* program),  
 41255 conflict results. While the solution is not historical practice, it corrects a known deficiency in  
 41256 historical implementations. Corresponding changes were made to all sections that referenced  
 41257 the filenames **y.tab.c** (now "the code file"), **y.tab.h** (now "the header file"), and **y.output** (now  
 41258 "the description file").

41259 The grammar for yacc input is based on System V documentation. The textual description shows  
 41260 there that the ' ; ' is required at the end of the rule. The grammar and the implementation do not  
 41261 require this. (The use of **C\_IDENTIFIER** causes a reduce to occur in the right place.)

41262 Also, in that implementation, the constructs such as **%token** can be terminated by a semicolon,  
 41263 but this is not permitted by the grammar. The keywords such as **%token** can also appear in  
 41264 uppercase, which is again not discussed. In most places where '**%**' is used, '**\**' can be  
 41265 substituted, and there are alternate spellings for some of the symbols (for example, **%LEFT** can  
 41266 be "**%<**" or even "**\<**").

41267 Historically, **<tag>** can contain any characters except '**>**', including white space, in the  
 41268 implementation. However, since the *tag* must reference a ISO C standard union member, in  
 41269 practice conforming implementations need to support only the set of characters for ISO C  
 41270 standard identifiers in this context.

41271 Some historical implementations are known to accept actions that are terminated by a period.  
 41272 Historical implementations often allow '**\$**' in names. A conforming implementation does not  
 41273 need to support either of these behaviors.

41274 Deciding when to use **%prec** illustrates the difficulty in specifying the behavior of yacc. There  
 41275 may be situations in which the *grammar* is not, strictly speaking, in error, and yet yacc cannot  
 41276 interpret it unambiguously. The resolution of ambiguities in the grammar can in many instances  
 41277 be resolved by providing additional information, such as using **%type** or **%union** declarations. It  
 41278 is often easier and it usually yields a smaller parser to take this alternative when it is  
 41279 appropriate.

41280 The size and execution time of a program produced without the runtime debugging code is  
 41281 usually smaller and slightly faster in historical implementations.

41282 Statistics messages from several historical implementations include the following types of  
 41283 information:

41284 *n*/512 terminals, *n*/300 non-terminals  
 41285 *n*/600 grammar rules, *n*/1500 states  
 41286 *n* shift/reduce, *n* reduce/reduce conflicts reported

41287  $n/350$  working sets used  
41288 Memory: states, etc.  $n/15\,000$ , parser  $n/15\,000$   
41289  $n/600$  distinct lookahead sets  
41290  $n$  extra closures  
41291  $n$  shift entries,  $n$  exceptions  
41292  $n$  goto entries  
41293  $n$  entries saved by goto default  
41294 Optimizer space used: input  $n/15\,000$ , output  $n/15\,000$   
41295  $n$  table entries,  $n$  zero  
41296 Maximum spread:  $n$ , Maximum offset:  $n$

41297 The report of internal tables in the description file is left implementation-defined because all  
41298 aspects of these limits are also implementation-defined. Some implementations may use  
41299 dynamic allocation techniques and have no specific limit values to report.

41300 The format of the **y.output** file is not given because specification of the format was not seen to  
41301 enhance applications portability. The listing is primarily intended to help human users  
41302 understand and debug the parser; use of **y.output** by a conforming application script would be  
41303 unusual. Furthermore, implementations have not produced consistent output and no popular  
41304 format was apparent. The format selected by the implementation should be human-readable, in  
41305 addition to the requirement that it be a text file.

41306 Standard error reports are not specifically described because they are seldom of use to  
41307 conforming applications and there was no reason to restrict implementations.

41308 Some implementations recognize " $=\{$ " as equivalent to ' $\{$ ' because it appears in historical  
41309 documentation. This construction was recognized and documented as obsolete as long ago as  
41310 1978, in the referenced *Yacc: Yet Another Compiler-Compiler*. This volume of IEEE Std 1003.1-200x  
41311 chose to leave it as obsolete and omit it.

41312 Multi-byte characters should be recognized by the lexical analyzer and returned as tokens. They  
41313 should not be returned as multi-byte character literals. The token **error** that is used for error  
41314 recovery is normally assigned the value 256 in the historical implementation. Thus, the token  
41315 value 256, which used in many multi-byte character sets, is not available for use as the value of a  
41316 user-defined token.

41317 **FUTURE DIRECTIONS**  
41318 None.

41319 **SEE ALSO**  
41320 *c99, lex*

41321 **CHANGE HISTORY**  
41322 First released in Issue 2.

41323 **Issue 5**  
41324 **FUTURE DIRECTIONS** section added.

41325 **Issue 6**  
41326 This utility is now marked as part of the C-Language Development Utilities option.  
41327 Minor changes have been added to align with the IEEE P1003.2b draft standard.  
41328 The normative text is reworded to avoid use of the term "must" for application requirements.  
41329 IEEE PASC Interpretation 1003.2 #177 is applied, changing the comment on **RCURL** from the }%  
41330 token to the %}.

41331 **NAME**

41332 zcat — expand and concatenate data

41333 **SYNOPSIS**41334 XSI zcat [*file...*]

41335

41336 **DESCRIPTION**

41337 The *zcat* utility shall write to standard output the uncompressed form of files that have been  
 41338 compressed using the *compress* utility. It is the equivalent of *uncompress -c*. Input files are not  
 41339 affected.

41340 **OPTIONS**

41341 None.

41342 **OPERANDS**

41343 The following operand shall be supported:

41344 *file* The pathname of a file previously processed by the *compress* utility. If *file* already  
 41345 has the *.Z* suffix specified, it is used as submitted. Otherwise, the *.Z* suffix is  
 41346 appended to the filename prior to processing.

41347 **STDIN**41348 The standard input shall be used only if no *file* operands are specified, or if a *file* operand is '- '.41349 **INPUT FILES**41350 Input files shall be compressed files that are in the format produced by the *compress* utility.41351 **ENVIRONMENT VARIABLES**41352 The following environment variables shall affect the execution of *zcat*:

41353 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 41354 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,  
 41355 Internationalization Variables for the precedence of internationalization variables  
 41356 used to determine the values of locale categories.)

41357 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 41358 internationalization variables.

41359 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 41360 characters (for example, single-byte as opposed to multi-byte characters in  
 41361 arguments).

41362 *LC\_MESSAGES*

41363 Determine the locale that should be used to affect the format and contents of  
 41364 diagnostic messages written to standard error.

41365 *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

41366 **ASYNCHRONOUS EVENTS**

41367 Default.

41368 **STDOUT**

41369 The compressed files given as input shall be written on standard output in their uncompressed  
 41370 form.

41371 **STDERR**

41372 The standard error shall be used only for diagnostic messages. |

41373 **OUTPUT FILES**

41374 None.

41375 **EXTENDED DESCRIPTION**

41376 None.

41377 **EXIT STATUS**

41378 The following exit values shall be returned:

41379 0 Successful completion.

41380 &gt;0 An error occurred.

41381 **CONSEQUENCES OF ERRORS**

41382 Default.

41383 **APPLICATION USAGE**

41384 None.

41385 **EXAMPLES**

41386 None.

41387 **RATIONALE**

41388 None.

41389 **FUTURE DIRECTIONS**

41390 None.

41391 **SEE ALSO**41392 *compress, uncompress*41393 **CHANGE HISTORY**

41394 First released in Issue 4.

# Index

1

2	<control>-V.....	2568	background work	
3	<control>-W .....	2569	at .....	2346
4	_POSIX_VDISABLE.....	3090	batch.....	2392
5	Account_Name .....	2310	bg.....	2410
6	actions equivalent to functions .....	2207	crontab.....	2476
7	admin.....	<b>2328</b>	fg.....	2643
8	ADV.....	<b>2210</b>	jobs .....	2720
9	AIO .....	<b>2210</b>	nice .....	2863
10	alias.....	<b>2333</b>	nohup.....	2875
11	alias substitution .....	2234	renice.....	3018
12	AND lists.....	2252	BAR.....	<b>2210</b>
13	AND-OR list .....	2251	basename.....	<b>2389</b>
14	appending redirected output .....	2245	batch.....	<b>2392</b>
15	ar .....	<b>2336</b>	batch administration.....	2306
16	archives		batch authorization.....	2305
17	ar command.....	2336	batch client-server interaction .....	2303
18	ARG_MAX.....	3259	batch environment	
19	arithmetic expansion .....	2243	services .....	2303
20	arithmetic language		utilities.....	2303
21	bc.....	2395	Batch Job Abort.....	2305, 2316
22	array identifiers.....	2400	batch job creation .....	2304
23	asa.....	<b>2343</b>	Batch Job Execution.....	2305, 2308
24	asynchronous lists.....	2252	Batch Job Exit .....	2305, 2315
25	at .....	<b>2346</b>	batch job identifier .....	2324
26	at-job.....	2346	Batch Job Message Request .....	2318
27	automatic storage class .....	2404	Batch Job Routing .....	2304, 2315
28	awk .....	<b>2355</b>	batch job states.....	2307
29	actions.....	2366	Batch Job Status Request.....	2319
30	arithmetic functions.....	2368	batch job tracking .....	2304
31	escape sequences.....	2364	batch notification.....	2306
32	expression patterns.....	2366	batch queue.....	2303
33	expressions .....	2358	Batch Queue Status Request .....	2321
34	functions.....	2368	Batch Server Restart.....	2316
35	grammar.....	2372	batch services .....	2306
36	input/output and general functions .....	2370	bc.....	<b>2395</b>
37	lexical conventions.....	2378	grammar .....	2396
38	output statements .....	2367	lexical conventions.....	2398
39	overall program structure .....	2357	operations .....	2400
40	pattern ranges .....	2366	operators .....	2400
41	patterns.....	2365	bcc (mailer blind carbon copy) .....	2807
42	regular expressions.....	2363	BC_BASE_MAX .....	2219
43	special patterns.....	2365	BC_DIM_MAX.....	2219
44	string functions.....	2369	BC_SCALE_MAX.....	2219
45	user-defined functions .....	2371	BC_STRING_MAX .....	2219, 2398
46	variables and special variables.....	2361	BE.....	<b>2211</b>
47			bg .....	<b>2410</b>

48	binary primaries .....	3106	compilers	
49	break.....	<b>2267</b>	c99.....	2413
50	break special built-in .....	2267	fort77.....	2663
51	builtin.....	2463	yacc.....	3261
52	c99.....	<b>2413</b>	compound commands.....	2253
53	external symbols.....	2417	compound-list.....	2251
54	standard libraries .....	2416	compress .....	<b>2465</b>
55	cal.....	<b>2422</b>	compression	
56	can.....	2201	compress .....	2465
57	carriage-control characters.....	2343	uncompress .....	3149
58	case conditional construct .....	2254	zcat .....	3277
59	cat.....	<b>2424</b>	concurrent execution of processes.....	2203
60	cc (mailer carbon copy) .....	2807	configuration values .....	2683
61	CD.....	<b>2211</b>	conforming application.....	2341
62	cd.....	<b>2428</b>	consequences of shell errors.....	2247
63	cflow.....	<b>2432</b>	continue .....	<b>2271</b>
64	changing the current working directory .....	2207	control characters .....	3087
65	character counting.....	3243	controlling terminal .....	2203
66	charmap		Coordinated Universal Time (UTC).....	2501
67	with localedef.....	2757	copy files commands	
68	writing names with locale .....	2752	cp .....	2468
69	charmap file.....	2756, 3090	dd.....	2503
70	CHAR_BIT.....	2549	ln.....	2748
71	Checkpoint.....	2310	mv.....	2854
72	checksums		pax.....	2900
73	cksum.....	2448	cp.....	<b>2468</b>
74	chgrp .....	<b>2435</b>	cpio format.....	2920
75	chmod .....	<b>2438</b>	CPT.....	<b>2211</b>
76	grammar.....	2441	CPU time.....	3115
77	chown.....	<b>2444</b>	cron daemon.....	2479
78	cksum.....	<b>2448</b>	crontab.....	<b>2476</b>
79	cmp.....	<b>2453</b>	CS.....	<b>2211</b>
80	codeset conversion.....	2705	csplit.....	<b>2480</b>
81	tr.....	3124	ctags.....	<b>2484</b>
82	COLL_WEIGHTS_MAX.....	2219	current working directory .....	2203
83	colon.....	<b>2269</b>	cut .....	<b>2489</b>
84	colon special built-in .....	2269, 2271	CX.....	<b>2211</b>
85	comm .....	<b>2456</b>	cxref.....	<b>2493</b>
86	command .....	<b>2459</b>	data keywords.....	2946
87	command mode.....	2537	date.....	<b>2496</b>
88	command search and execution.....	2249	conversion specifications .....	2496
89	command substitution .....	2242	modified conversion specifications.....	2497
90	communications commands		dd.....	<b>2503</b>
91	mailx .....	2785	default queue.....	2320
92	talk.....	3098	deferred batch services.....	2308
93	uucp .....	3163	Delete Batch Job Request .....	2317
94	uudecode.....	3167	delta.....	<b>2512</b>
95	uuencode.....	3170	destination .....	2325
96	uustat .....	3175	df.....	<b>2516</b>
97	uux.....	3178	diff.....	<b>2520</b>
98	write .....	3252	binary output format.....	2522

## Index

99	default output format.....	2522	regular expressions.....	2538
100	directory comparison format.....	2521	shell escape command.....	2547
101	–c or –C output format.....	2523	substitute command.....	2545
102	–e output format.....	2523	undo command.....	2546
103	–f output format.....	2523	write command.....	2547
104	directory commands		edit buffer.....	2556, 3185
105	cd.....	2428	edit line.....	3053
106	pwd.....	2957	editors	
107	directory lister.....	2770	ed.....	2537
108	dirname.....	<b>2527</b>	ex.....	2556
109	disk space commands		sed.....	3039
110	df.....	2516	vi.....	3185
111	du.....	2530	ED_FILE_MAX.....	2549
112	ulimit.....	3138	ED_LINE_MAX.....	2549
113	documentation.....	2830	effective group ID.....	2203
114	dot.....	<b>2273</b>	effective user ID.....	2203
115	dot special built-in.....	2273	Eighth Edition UNIX.....	2463
116	double-quotes.....	2232	env.....	<b>2553</b>
117	du.....	<b>2530</b>	EPERM.....	2473
118	duplicating an input file descriptor.....	2246	Error_Path.....	2311
119	duplicating an output file descriptor.....	2247	escape character (backslash).....	2232
120	echo.....	<b>2534</b>	escape sequences	
121	ed.....	<b>2537</b>	awk.....	2364
122	addresses.....	2539	gencat.....	2673
123	append command.....	2541	lex.....	2739
124	change command.....	2541	establish the locale.....	2207
125	commands.....	2540	eval.....	<b>2275</b>
126	copy command.....	2546	eval special built-in.....	2275
127	delete command.....	2542	ex.....	<b>2556</b>
128	edit command.....	2542	<backslash>.....	2568
129	edit without checking command.....	2542	<control>-D command.....	2591
130	filename command.....	2542	<newline>.....	2568
131	global command.....	2542	abbreviate command.....	2571
132	global non-matched command.....	2547	addressing.....	2562
133	help command.....	2543	adjust window command.....	2589
134	help-mode command.....	2543	append command.....	2572
135	insert command.....	2543	args command.....	2572
136	interactive global command.....	2543	autoindent option.....	2593
137	interactive global not-matched command.....	2547	autoprint option.....	2594
138	join command.....	2544	autowrite option.....	2594
139	line number command.....	2547	beautify option.....	2594
140	list command.....	2544	change command.....	2572
141	mark command.....	2544	chdir command.....	2572
142	move command.....	2544	command descriptions.....	2569
143	null command.....	2548	copy command.....	2573
144	number command.....	2544	delete command.....	2573
145	print command.....	2545	directory option.....	2594
146	prompt command.....	2545	edcompatible option.....	2594
147	quit command.....	2545	edit command.....	2573
148	quit without checking command.....	2545	edit options.....	2593
149	read command.....	2545	errorbells option.....	2595

150	escape command.....	2590	tag command .....	2585
151	execute command.....	2592	taglength option.....	2598
152	exrc command.....	2595	tags command.....	2599
153	file command.....	2574	term command.....	2599
154	global command.....	2574	terse command .....	2599
155	ignorecase option.....	2595	unabbrev command.....	2586
156	initialization.....	2559	undo command.....	2586
157	input editing.....	2567	unmap command.....	2586
158	insert command.....	2575	version command.....	2586
159	join command .....	2575	visual command.....	2587
160	list command.....	2576, 2595	warn command.....	2599
161	magic command.....	2595	window command.....	2599
162	map command.....	2577	wrapmargin option.....	2600
163	mark command.....	2578	wrapscan option.....	2600
164	mesg command.....	2595	write command.....	2587
165	move command.....	2579	write line number command .....	2591
166	next command.....	2579	writeln option .....	2600
167	number command.....	2580	xit command .....	2588
168	number option.....	2596	yank command .....	2588
169	open command.....	2580	exec.....	<b>2277</b>
170	paragraphs option.....	2596	Fam.....	2876
171	preserve.....	2556	exec family .....	2463, 2861, 3259
172	preserve command .....	2580	exec special built-in.....	2277
173	print command.....	2580	Execution_Time .....	2312
174	prompt command .....	2596	EXINIT.....	2556
175	put command.....	2581	exit .....	<b>2279</b>
176	quit command.....	2581	exit special built-in.....	2279
177	read command.....	2581	exit status and errors .....	2247
178	readonly command.....	2596	exit status for commands.....	2248
179	recover command .....	2582	expand .....	<b>2627</b>
180	redraw command.....	2597	export.....	<b>2281</b>
181	regular expressions.....	2592	export special built-in.....	2281
182	remap command .....	2597	expr.....	<b>2630</b>
183	replacement strings.....	2592	matching expression.....	2632
184	report command.....	2597	string operand.....	2632
185	rewind command.....	2582	expression argument .....	2367
186	scroll command.....	2567, 2597	expression list.....	2367
187	sections command .....	2597	EXPR_NEST_MAX.....	2219
188	set command.....	2583	extended regular expression.....	2355, 2363
189	shell command.....	2583	.....	2472, 2655, 2694, 2738, 2856, 2909
190	shell option.....	2598	.....	3023, 3257
191	shift left command.....	2591	extension	
192	shift right command.....	2591	CX.....	2211
193	shiftwidth option.....	2598	OH.....	2213
194	showmatch option.....	2598	XSI .....	2217
195	showmode command.....	2598	false.....	<b>2635</b>
196	slowopen command.....	2598	fc .....	<b>2637</b>
197	source command .....	2583	FD.....	<b>2211</b>
198	substitute command.....	2584	fg.....	<b>2643</b>
199	suspend command.....	2585	field splitting .....	2243
200	tabstop option.....	2598	FIFO special files .....	2840

## Index

201	file .....	<b>2645</b>	dd.....	2503
202	file access permissions .....	2204	expand.....	2627
203	file comparisons		fold.....	2660
204	cmp.....	2453	head.....	2702
205	comm.....	2456	iconv.....	2705
206	diff.....	2520	more.....	2842
207	uniq.....	3157	nl.....	2866
208	file contents.....	2206	paste.....	2886
209	file conversion		pax.....	2900
210	cut.....	2489	pr.....	2935
211	dd.....	2503	read.....	3015
212	expand.....	2627	sed.....	3039
213	fold.....	2660	tail.....	3095
214	head.....	2702	tee.....	3102
215	join.....	2724	tr.....	3124
216	od.....	2878	uncompress.....	3149
217	paste.....	2886	unexpand.....	3152
218	patch.....	2890	zcat.....	3277
219	sort.....	3068	find.....	<b>2652</b>
220	strings.....	3077	fold.....	<b>2660</b>
221	tail.....	3095	for loop.....	2253
222	tr.....	3124	fort77.....	<b>2663</b>
223	tsort.....	3132	external symbols.....	2666
224	unexpand.....	3152	standard libraries.....	2665
225	uniq.....	3157	FR.....	<b>2211</b>
226	uudecode.....	3167	FSC.....	<b>2211</b>
227	uuencode.....	3170	function definition command.....	2256
228	file creation.....	2204	function identifiers.....	2400
229	file descriptor.....	2203, 2244	fuser.....	<b>2669</b>
230	file mode creation mask.....	2203	g-file.....	2512
231	file permission commands		gencat.....	<b>2672</b>
232	chgrp.....	2435	escape sequences.....	2673
233	chmod.....	2438	generated file.....	2512
234	chown.....	2444	get.....	<b>2675</b>
235	umask.....	3140	getconf.....	<b>2683</b>
236	file read.....	2204	getopts.....	<b>2689</b>
237	file removal.....	2206	global storage class.....	2404
238	file searching		GNU make.....	2826
239	grep.....	2694	grep.....	<b>2694</b>
240	file time values.....	2206	grouping commands.....	2253
241	file tree commands		hash.....	<b>2699</b>
242	diff.....	2520	head.....	<b>2702</b>
243	find.....	2652	here-document.....	2246
244	ls.....	2770	history command	
245	mkdir.....	2837	fc.....	2637
246	rmdir.....	3029	Hold Batch Job Request.....	2318
247	file write.....	2204	Hold_Types.....	2312
248	filters		HOME.....	2572
249	asa.....	2343	hunk.....	2892
250	awk.....	2355	iconv.....	<b>2705</b>
251	compress.....	2465	id.....	<b>2708</b>

252	if conditional construct .....	2254	command escapes .....	2803
253	implementation-defined .....	2201	commands .....	2795
254	inference rule .....	2809	copy messages .....	2796
255	input field separator .....	3050	declare aliases .....	2795
256	input mode.....	2537	declare alternatives.....	2796
257	IP6.....	<b>2212</b>	delete aliases.....	2802
258	ipcrm .....	<b>2712</b>	delete messages .....	2796
259	ipcs.....	<b>2714</b>	delete messages and display.....	2797
260	jobs.....	<b>2720</b>	direct messages to mbox.....	2799
261	Job_Owner .....	2312	discard header fields.....	2796
262	join .....	<b>2724</b>	display beginning of messages.....	2802
263	Join_Path .....	2312	display current message number.....	2803
264	Keep_Files.....	2312	display header summary .....	2798
265	keyword-value pairs.....	2325	display list of folders .....	2798
266	kill .....	<b>2729</b>	display message.....	2800
267	legacy .....	2201	display message size .....	2801
268	lex.....	<b>2734</b>	echo a string .....	2797
269	actions.....	2740	edit message.....	2797, 2802
270	definitions.....	2736	execute commands conditionally .....	2799
271	escape sequences.....	2739	exit .....	2797
272	regular expressions .....	2738	follow up specified messages.....	2798
273	rules.....	2737	help.....	2798
274	table sizes .....	2737	hold messages.....	2798
275	user subroutines .....	2738	internal variables.....	2792
276	lex, translation table.....	2744	invoke a shell.....	2801
277	libraries		invoke shell command .....	2803
278	ar command.....	2336	list available commands .....	2799
279	LIMIT .....	2218	mail a message.....	2799
280	line counting.....	3243	null command.....	2803
281	LINE_MAX.....	2219, 2356, 2549-2550	pipe message.....	2799
282	.....	2557, 2806, 3047, 3160	process next specified message.....	2799
283	link.....	<b>2746</b>	quit .....	2800
284	lists.....	2251	read mailx commands from a file .....	2801
285	AND-OR .....	2251	receive mode .....	2785
286	compound-list.....	2251	reply to a message.....	2800
287	ln .....	<b>2748</b>	reply to a message list .....	2800
288	locale .....	<b>2752</b>	retain header fields .....	2801
289	localedef .....	<b>2757</b>	save messages .....	2801
290	Locate Batch Job Request.....	2319	scroll header display.....	2803
291	locking file.....	2442	send mode.....	2785
292	logger .....	<b>2761</b>	set variables.....	2801
293	logname .....	<b>2763</b>	start-up .....	2792
294	lp .....	<b>2765</b>	touch messages.....	2802
295	LR(1) grammars .....	3275	undelete messages .....	2802
296	ls.....	<b>2770</b>	unset variables .....	2802
297	m4 .....	<b>2778</b>	write messages to a file .....	2802
298	macro processor.....	2778	Mail_Points .....	2313
299	magic file .....	2650	Mail_Users .....	2313
300	mailx.....	<b>2785</b>	make.....	<b>2809</b>
301	change current directory.....	2796	default rules.....	2819
302	change folder.....	2797	inference rules.....	2816

## Index

303	internal macros .....	2818	MPR.....	2213
304	libraries.....	2817	MSG.....	2213
305	macros .....	2815	mv.....	2854
306	makefile execution.....	2813	MX.....	2213
307	makefile syntax.....	2812	NAME_MAX.....	2340, 2926
308	target rules.....	2813	newgrp.....	2859
309	make, GNU version .....	2826	NGROUPS_MAX .....	2862
310	man.....	<b>2830</b>	nice.....	<b>2863</b>
311	may.....	2202	Ninth Edition UNIX.....	2407, 2536, 2944
312	MC1.....	<b>2212</b>	nl.....	<b>2866</b>
313	MC2.....	<b>2212</b>	nm.....	<b>2870</b>
314	mesg.....	<b>2834</b>	noclobber option.....	2899
315	message catalog generation .....	2672	nohup.....	<b>2875</b>
316	MF.....	<b>2212</b>	non-printable.....	2549, 3046, 3101
317	MIL-STD-1753.....	2667	OB.....	<b>2213</b>
318	Minimum_Cpu_Interval.....	2310	object files.....	2870
319	mkdir.....	<b>2837</b>	od.....	<b>2878</b>
320	mkfifo.....	<b>2840</b>	OF.....	<b>2213</b>
321	ML.....	<b>2212</b>	OH.....	<b>2213</b>
322	MLR.....	<b>2212</b>	open file descriptors for reading and writing	2247
323	Modify Batch Job Request .....	2319	open mode .....	2556
324	MON.....	<b>2213</b>	option	
325	more .....	<b>2842</b>	ADV.....	2210
326	discard and refresh .....	2848	AIO.....	2210
327	display position.....	2851	BAR.....	2210
328	examine new file.....	2850	BE.....	2211
329	examine next file .....	2850	CD.....	2211
330	examine previous file .....	2850	CPT.....	2211
331	go to beginning of file.....	2848	CS.....	2211
332	go to end-of-file .....	2848	FD.....	2211
333	go to tag.....	2850	FR.....	2211
334	help.....	2847	FSC.....	2211
335	invoke editor .....	2850	IP6.....	2212
336	mark position.....	2849	MC1.....	2212
337	quit .....	2851	MC2.....	2212
338	refresh the screen.....	2848	MF.....	2212
339	repeat search .....	2849	ML.....	2212
340	repeat search in reverse .....	2850	MLR.....	2212
341	return to mark.....	2849	MON.....	2213
342	return to previous position .....	2849	MPR.....	2213
343	scroll backward one half screenful.....	2848	MSG.....	2213
344	scroll backward one line.....	2847	MX.....	2213
345	scroll backward one screenful.....	2847	PIO.....	2213
346	scroll forward one half screenful .....	2848	PS.....	2214
347	scroll forward one line .....	2847	RS.....	2214
348	scroll forward one screenful .....	2847	RTS.....	2214
349	search backward for pattern.....	2849	SD.....	2214
350	search forward for pattern .....	2849	SEM.....	2214
351	skip forward one line.....	2848	SHM.....	2214
352	motion command.....	3054	SIO.....	2214
353	Move Batch Job Request .....	2320	SPI.....	2214

354	SPN.....	2215	pax.....	<b>2900</b>
355	SS.....	2215	archive character set encoding/decoding...	2931
356	TCT.....	2215	cpio file data.....	2922
357	TEF.....	2215	cpio filename.....	2922
358	THR.....	2215	cpio header.....	2920
359	TMO.....	2215	cpio interchange format.....	2920
360	TMR.....	2215	cpio special entries.....	2922
361	TPI.....	2216	extended header.....	2913
362	TPP.....	2216	extended header file times.....	2916
363	TPS.....	2216	extended header keyword precedence.....	2916
364	TRC.....	2216	list mode format specifications.....	2908
365	TRI.....	2216	ustar format.....	2916
366	TRL.....	2216	ustar interchange format.....	2916
367	TSA.....	2216	PIO.....	<b>2213</b>
368	TSF.....	2216	pipelines.....	2250
369	TSH.....	2217	portable character set.....	2815
370	TSP.....	2217	positional parameters.....	2235
371	TSS.....	2217	POSIX2_BC_BASE_MAX.....	2218-2219
372	TYM.....	2217	POSIX2_BC_DIM_MAX.....	2218-2219
373	UP.....	2217	POSIX2_BC_SCALE_MAX.....	2218-2219
374	XSR.....	2217	POSIX2_BC_STRING_MAX.....	2218-2219
375	OR lists.....	2253	POSIX2_COLL_WEIGHTS_MAX.....	2218-2219
376	ordinary identifiers.....	2400	POSIX2_EXPR_NEST_MAX.....	2218-2219
377	Output_Path.....	2313	POSIX2_LINE_MAX.....	2218, 2220
378	paginators		POSIX2_RE_DUP_MAX.....	2218, 2220
379	more.....	2842	POSIX2_SYMLINKS.....	2220
380	parameter expansion.....	2239	POSIX2_VERSION.....	2218
381	parameters and variables.....	2235	pr.....	<b>2935</b>
382	paste.....	<b>2886</b>	print-related commands	
383	patch.....	<b>2890</b>	fold.....	2660
384	filename determination.....	2893	lp.....	2765
385	patch application.....	2893	pr.....	2935
386	patchfile format.....	2892	printf.....	<b>2940</b>
387	pathchk.....	<b>2896</b>	Priority.....	2314
388	pathname expansion.....	2244	privileges.....	2835, 2865
389	pathname manipulation		process attributes.....	2203
390	basename.....	2389	process group ID.....	2203
391	dirname.....	2527	process ID.....	2203
392	pathchk.....	2896	process status report.....	2950
393	pathname resolution.....	2207	prs.....	<b>2945</b>
394	PATH_MAX.....	2219, 2932, 3025	PS.....	<b>2214</b>
395	pattern matching.....	2652, 2909, 3164, 3180	ps.....	<b>2950</b>
396	definition.....	2264	public locale.....	2752
397	in case statements.....	2254	pwd.....	<b>2957</b>
398	in shell variables.....	2240	qalter.....	<b>2959</b>
399	pattern matching notation.....	2264, 2658, 2925	qdel.....	<b>2968</b>
400	pattern scanning and processing language		qhold.....	<b>2971</b>
401	at.....	2355	qmove.....	<b>2974</b>
402	patterns matching a single character.....	2264	qmsg.....	<b>2977</b>
403	patterns matching multiple characters.....	2264	qrerun.....	<b>2980</b>
404	patterns used for filename expansion.....	2265	qrls.....	<b>2982</b>

## Index

405	qselect .....	<b>2985</b>	unset .....	3155
406	qsig .....	<b>2994</b>	val .....	3182
407	qstat .....	<b>2997</b>	what .....	3246
408	qsub .....	<b>3002</b>	SD .....	<b>2214</b>
409	Queue Batch Job Request.....	2320	search pattern.....	2484
410	quote removal .....	2244	sed.....	<b>3039</b>
411	quoting.....	2232	addresses.....	3041
412	read.....	<b>3015</b>	editing commands .....	3041
413	readonly.....	<b>2283</b>	regular expressions.....	3041
414	readonly special built-in .....	2283	Select Batch Jobs Request .....	2322
415	real group ID .....	2203	SEM .....	<b>2214</b>
416	real user ID.....	2203	sequential lists.....	2252
417	redirecting input.....	2245	Server Shutdown Request .....	2322
418	redirecting output .....	2245	Server Status Request .....	2323
419	redirection.....	2244	session membership.....	2203
420	regular expressions .....	2363, 2472, 2538	set.....	<b>2287</b>
421	.....	2562, 2592, 2632, 2655, 2694, 2738, 2846	set special built-in.....	2287
422	.....	2856, 2866, 2906, 3023, 3041, 3202, 3205	set-group-ID.....	2203, 2474
423	.....	3257	set-user-ID.....	2203, 2443, 2474
424	related to shell patterns.....	2264	set-user-ID scripts .....	3062
425	relational database operator .....	2724	sh.....	<b>3048</b>
426	Release Batch Job Request.....	2321	command history list.....	3052
427	remove directories.....	3029	command line editing .....	3052
428	remove files .....	3022	vi line editing command mode .....	3054
429	renice.....	<b>3018</b>	vi line editing insert mode.....	3053
430	requested batch services .....	2317	vi-mode command line editing.....	3053
431	Rerun Batch Job Request.....	2322	shall .....	2202
432	Rerunable .....	2314	shell command language.....	2231
433	reserved words .....	2235	alias substitution .....	2234
434	Resource_List .....	2314	appending redirected output.....	2245
435	return.....	<b>2285</b>	arithmetic expansion .....	2243
436	return special built-in .....	2285	command substitution.....	2242
437	RE_DUP_MAX.....	2219	compound commands .....	2253
438	rm.....	<b>3022</b>	consequences of shell errors .....	2247
439	rm del.....	<b>3027</b>	double-quotes .....	2232
440	rm dir .....	<b>3029</b>	duplicating an input file descriptor.....	2246
441	root directory.....	2203	duplicating an output file descriptor.....	2247
442	RS .....	<b>2214</b>	escape character (backslash).....	2232
443	RTS .....	<b>2214</b>	exit status and errors .....	2247
444	sact.....	<b>3031</b>	exit status for commands .....	2248
445	saved set-group-ID .....	2203	field splitting .....	2243
446	saved set-user-ID.....	2203	function definition command.....	2256
447	sccs.....	<b>3034</b>	grammar.....	2257
448	SCCS commands		here-document.....	2246
449	admin.....	2328	introduction.....	2231
450	delta.....	2512	lists .....	2251
451	get.....	2675	open file descriptors for reading and writing.....	2247
452	prs.....	2945	parameter expansion.....	2239
453	rm del.....	3027	parameters and variables .....	2235
454	sact.....	3031	pathname expansion .....	2244
455	sccs .....	3034		

456	pattern matching notation.....	2264	sort.....	<b>3068</b>
457	patterns matching a single character .....	2264	special built-in .....	2463, 2865
458	patterns matching multiple characters.....	2264	.....	2877, 2958, 3062, 3115, 3131
459	patterns used for filename expansion.....	2265	special built-in utilities.....	2266
460	pipelines.....	2250	break .....	2267, 2271
461	positional parameters.....	2235	characteristics.....	2266
462	quote removal.....	2244	colon.....	2269
463	quoting .....	2232	dot.....	2273
464	redirecting input.....	2245	eval.....	2275
465	redirecting output .....	2245	exec.....	2277
466	redirection.....	2244	exit.....	2279
467	reserved words .....	2235	export.....	2281
468	shell commands.....	2248	readonly .....	2283
469	shell execution environment.....	2263	return .....	2285
470	shell grammar lexical conventions.....	2257	set.....	2287
471	shell grammar rules .....	2257	shift.....	2293
472	shell variables.....	2236	times.....	2295
473	signals and error handling .....	2262	trap .....	2297
474	simple commands .....	2248	unset.....	2300
475	single-quotes .....	2232	special parameters.....	2235
476	special built-in utilities.....	2266	special targets.....	2814
477	special parameters .....	2235	SPI.....	<b>2214</b>
478	tilde expansion.....	2239	split.....	<b>3074</b>
479	token recognition .....	2233	split files	
480	word expansions .....	2238	csplit.....	2480
481	shell commands .....	2248	split.....	3074
482	shell execution environment.....	2263	SPN.....	<b>2215</b>
483	.....	2334, 3017, 3142	spoofing.....	2284
484	shell grammar .....	2257	SS.....	<b>2215</b>
485	shell grammar lexical conventions .....	2257	standard error.....	2244
486	shell grammar rules .....	2257	standard input.....	2244
487	shell introduction .....	2231	standard output .....	2244
488	shell variables.....	2236	strings.....	<b>3077</b>
489	Shell_Path_List.....	2314	strip.....	<b>3080</b>
490	shift.....	<b>2293</b>	stty.....	<b>3082</b>
491	shift special built-in.....	2293	combination modes .....	3087
492	SHM .....	<b>2214</b>	control modes .....	3082
493	should .....	2202	input modes.....	3083
494	SIGCONT.....	2559	local modes.....	3085
495	SIGHUP.....	2538, 2559, 3185, 3226	output modes.....	3084
496	SIGINT.....	2538, 2558, 3238	special control character assignments .....	3086
497	Signal Batch Job Request.....	2323	st_gid.....	2341
498	signal processes .....	2729	st_mode .....	2341
499	signals and error handling.....	2262	st_mtime.....	2341
500	SIGQUIT.....	2538	st_size.....	2341
501	SIGTERM .....	2559	st_uid.....	2341
502	simple commands .....	2248	superuser.....	2641, 2776, 2924
503	single-quotes .....	2232	supplementary group IDs .....	2203
504	SIO .....	<b>2214</b>	system configuration values .....	2683
505	sleep.....	<b>3065</b>	system name.....	3146
506	SLR(1) grammars.....	3275	tabs .....	<b>3091</b>

## Index

507	tag file creation.....	2484	unalias.....	3144
508	tail.....	3095	uname.....	3146
509	talk.....	3098	unary primaries.....	3106
510	tar format.....	2916	uncompress.....	3149
511	target queue.....	2320	undefined.....	2202
512	target rule.....	2809	unexpand.....	3152
513	TCT.....	2215	unset.....	3155
514	tee.....	3102	uniq.....	3157
515	TEF.....	2215	unlink.....	3161
516	terminal characteristics		unset.....	2300
517	stty.....	3082	unset special built-in.....	2300
518	tabs.....	3091	unspecified.....	2202
519	tput.....	3121	until loop.....	2255
520	tty.....	3134	UP.....	2217
521	terminate processes.....	2729	user identity	
522	terminology.....	2201	id.....	2708
523	test.....	3105	logname.....	2763
524	THR.....	2215	newgrp.....	2859
525	tilde expansion.....	2239	who.....	3248
526	time.....	3113	User_List.....	2315
527	times.....	2295	ustar format.....	2916
528	times special built-in.....	2295	utility option parsing.....	2689
529	TMO.....	2215	uucp.....	3163
530	TMPDIR.....	2904	uudecode.....	3167
531	TMR.....	2215	uencode.....	3170
532	token recognition.....	2233	uustat.....	3175
533	touch.....	3117	uux.....	3178
534	TPI.....	2216	val.....	3182
535	TPP.....	2216	Variable_List.....	2315
536	TPS.....	2216	vi.....	3185
537	tput.....	3121	<ESC>.....	3225
538	tr.....	3124	append.....	3206
539	Track Batch Job Request.....	2323	change.....	3207
540	trap.....	2297	change to end-of-line.....	3207
541	trap special built-in.....	2297	clear and redisplay.....	3193
542	TRC.....	2216	command descriptions.....	3186
543	TRI.....	2216	control-D.....	3222
544	TRL.....	2216	control-H.....	3222
545	trojan horse.....	2776	control-T.....	3224
546	true.....	3130	control-U.....	3224
547	TSA.....	2216	control-V.....	3224
548	TSF.....	2216	current and line above.....	3200
549	TSH.....	2217	delete.....	3208
550	tsort.....	3132	delete character.....	3217-3218
551	TSP.....	2217	delete to end-of-line.....	3208
552	TSS.....	2217	display information.....	3192
553	tty.....	3134	edit the alternate file.....	3194
554	TYM.....	2217	enter ex mode.....	3214
555	type.....	3136	execute.....	3205
556	ulimit.....	3138	execute an ex command.....	3204
557	umask.....	3140	exit.....	3220

558	find character .....	3209	undo .....	3216
559	find regular expression .....	3202	undo current line .....	3216
560	Initialization .....	3186	yank.....	3218
561	input mode commands .....	3220	yank current line.....	3218
562	insert.....	3210-3211	visual mode .....	2556
563	insert empty line .....	3212-3213	wait.....	<b>3239</b>
564	join.....	3211	warning	
565	mark position .....	3211	OB.....	2213
566	move back .....	3200, 3206	OF.....	2213
567	move cursor .....	3192, 3195-3196, 3215	wc.....	<b>3243</b>
568	move down.....	3193	what.....	<b>3246</b>
569	move forward .....	3200	while loop .....	2255
570	move to bigword.....	3209, 3217	who.....	<b>3248</b>
571	move to bottom of screen.....	3211	word counting.....	3243
572	move to first character in line.....	3203	word expansions .....	2238
573	move to first non-<blank>.....	3199	write .....	<b>3252</b>
574	move to line.....	3210	xargs .....	<b>3255</b>
575	move to matching character .....	3196	XSI.....	<b>2217</b>
576	move to middle of screen .....	3212	XSR .....	<b>2217</b>
577	move to next section.....	3199	yacc.....	<b>3261</b>
578	move to specific column.....	3201	algorithms.....	3272
579	move to top of screen.....	3210	code file .....	3263
580	move to word .....	3209, 3216	completing the program .....	3272
581	move up .....	3193	conflicts .....	3270
582	newline .....	3223	debugging the parser.....	3272
583	nul .....	3222, 3225	declarations section .....	3264
584	page backwards.....	3191	description file .....	3263
585	page forward.....	3192	error handling .....	3270
586	put from buffer .....	3213	grammar rules.....	3266
587	redraw screen.....	3194	header file .....	3263
588	redraw window .....	3219	input grammar.....	3268
589	regular expression.....	3205	input language.....	3263
590	repeat .....	3201	interface to the lexical analyzer.....	3271
591	repeat find .....	3204, 3212	lexical structure of the grammar.....	3264
592	repeat substitution.....	3197	library.....	3272
593	replace character.....	3214-3215	limits .....	3273
594	replace text with command .....	3195	programs section.....	3267
595	return to previous context.....	3197-3198	zcat.....	<b>3277</b>
596	return to previous section .....	3198		
597	reverse case.....	3205		
598	reverse find character.....	3201		
599	scroll backward.....	3194		
600	scroll backward by line .....	3194		
601	scroll forward.....	3191		
602	scroll forward by line .....	3191		
603	search for tagstring .....	3195		
604	shift left.....	3204		
605	shift right.....	3204		
606	substitute character.....	3215		
607	substitute lines.....	3215		
608	terminate command or input mode.....	3194		

1 / *Rationale (Informative)* |

2 **Part A:** |  
3 **Base Definitions** |

4 *The Open Group* |  
5 *The Institute of Electrical and Electronics Engineers, Inc.* |



# Rationale for Base Definitions

7

## 8 A.1 Introduction

### 9 A.1.1 Scope

10 IEEE Std 1003.1-200x is one of a family of standards known as POSIX. The family of standards  
11 extends to many topics; IEEE Std 1003.1-200x is known as POSIX.1 and consists of both  
12 operating system interfaces and shell and utilities. IEEE Std 1003.1-200x is technically identical  
13 to The Open Group Base Specifications, Issue 6, which comprise the core volumes of the Single  
14 UNIX Specification, Version 3.

#### 15 Scope of IEEE Std 1003.1-200x

16 The (paraphrased) goals of this development were to produce a single common revision to the  
17 overlapping POSIX.1 and POSIX.2 standards, and the Single UNIX Specification, Version 2. As  
18 such, the scope of the revision includes the scopes of the original documents merged.

19 Since the revision includes merging the Base volumes of the Single UNIX Specification, many  
20 features that were previously not *adopted* into earlier revisions of POSIX.1 and POSIX.2 are now  
21 included in IEEE Std 1003.1-200x. In most cases, these additions are part of the XSI extension; in  
22 other cases the standard developers decided that now was the time to migrate these to the base  
23 standard.

24 The Single UNIX Specification programming environment provides a broad-based functional set  
25 of interfaces to support the porting of existing UNIX applications and the development of new  
26 applications. The environment also supports a rich set of tools for application development.

27 The majority of the obsolescent material from the existing POSIX.1 and POSIX.2 standards, and  
28 material marked LEGACY from The Open Group's Base specifications, has been removed in this  
29 revision. New members of the Legacy Option Group have been added, reflecting the advance in  
30 understanding of what is required.

31 The following IEEE Standards have been added to the base documents in this revision:

- 32 • IEEE Std 1003.1d-1999
- 33 • IEEE Std 1003.1j-2000
- 34 • IEEE Std 1003.1q-2000
- 35 • IEEE P1003.1a draft standard
- 36 • IEEE Std 1003.2d-1994
- 37 • IEEE P1003.2b draft standard
- 38 • Selected parts of IEEE Std 1003.1g-2000

39 Only selected parts of IEEE Std 1003.1g-2000 were included. This was because there is much  
40 duplication between the XNS, Issue 5.2 specification (another base document) and the material  
41 from IEEE Std 1003.1g-2000, the former document being aligned with the latest networking  
42 specifications for IPv6. Only the following sections of IEEE Std 1003.1g-2000 were considered for  
43 inclusion:

- 44 • General terms related to sockets (2.2.2)
- 45 • Socket concepts (5.1 through 5.3 inclusive)
- 46 • The *pselect()* function (6.2.2.1 and 6.2.3)
- 47 • The `<sys/select.h>` header (6.2)

48 The following were requirements on IEEE Std 1003.1-200x:

- 49 • Backward-compatibility

50 It was agreed that there should be no breakage of functionality in the existing base  
 51 documents. This requirement was tempered by changes introduced due to interpretations  
 52 and corrigenda on the base documents, and any changes introduced in the  
 53 ISO/IEC 9899:1999 standard (C Language).

- 54 • Architecture and n-bit neutral

55 The common standard should not make any implicit assumptions about the system  
 56 architecture or size of data types; for example, previously some 32-bit implicit assumptions  
 57 had crept into the standards.

- 58 • Extensibility

59 It should be possible to extend the common standard without breaking backward-  
 60 compatibility. For example, the name space should be reserved and structured to avoid  
 61 duplication of names between the standard and extensions to it.

## 62 **POSIX.1 and the ISO C standard**

63 Previous revisions of POSIX.1 built upon the ISO C standard by reference only. This revision  
 64 takes a different approach.

65 The standard developers believed it essential for a programmer to have a single complete  
 66 reference place, but recognized that deference to the formal standard had to be addressed for the  
 67 duplicate interface definitions between the ISO C standard and the Single UNIX Specification. |

68 It was agreed that where an interface has a version in the ISO C standard, the DESCRIPTION  
 69 section should describe the relationship to the ISO C standard and markings should be added as  
 70 appropriate to show where the ISO C standard has been extended in the text.

71 The following block of text was added to each reference page affected:

72 *The functionality described on this reference page is aligned with the ISO C standard. Any conflict*  
 73 *between the requirements described here and the ISO C standard is unintentional. This volume of*  
 74 *IEEE Std 1003.1-200x defers to the ISO C standard.*

75 and each page was parsed for additions beyond the ISO C standard (that is, including both  
 76 POSIX and UNIX extensions), and these extensions are marked as CX extensions (for C  
 77 Extensions). |

**FIPS Requirements**

The Federal Information Processing Standards (FIPS) are a series of U.S. government procurement standards managed and maintained on behalf of the U.S. Department of Commerce by the National Institute of Standards and Technology (NIST).

The following restrictions have been made in this version of IEEE Std 1003.1 in order to align with FIPS 151-2 requirements:

- The implementation supports `_POSIX_CHOWN_RESTRICTED`.
- The limit `{NGROUPS_MAX}` is now greater than or equal to 8.
- The implementation supports the setting of the group ID of a file (when it is created) to that of the parent directory.
- The implementation supports `_POSIX_SAVED_IDS`.
- The implementation supports `_POSIX_VDISABLE`.
- The implementation supports `_POSIX_JOB_CONTROL`.
- The implementation supports `_POSIX_NO_TRUNC`.
- The `read()` function returns the number of bytes read when interrupted by a signal and does not return `-1`.
- The `write()` function returns the number of bytes written when interrupted by a signal and does not return `-1`.
- In the environment for the login shell, the environment variables `LOGNAME` and `HOME` are defined and have the properties described in IEEE Std 1003.1-200x.
- The value of `{CHILD_MAX}` is now greater than or equal to 25.
- The value of `{OPEN_MAX}` is now greater than or equal to 20.
- The implementation supports the functionality associated with the symbols `CS7`, `CS8`, `CSTOPB`, `PARODD`, and `PARENB` defined in `<termios.h>`.

**A.1.2 Conformance**

See Section A.2 (on page 3299).

**A.1.3 Normative References**

There is no additional rationale provided for this section.

**A.1.4 Terminology**

The meanings specified in IEEE Std 1003.1-200x for the words *shall*, *should*, and *may* are mandated by ISO/IEC directives.

In the Rationale (Informative) volume of IEEE Std 1003.1-200x, the words *shall*, *should*, and *may* are sometimes used to illustrate similar usages in IEEE Std 1003.1-200x. However, the rationale itself does not specify anything regarding implementations or applications.

**conformance document**

As a practical matter, the conformance document is effectively part of the system documentation. Conformance documents are distinguished by IEEE Std 1003.1-200x so that they can be referred to distinctly.

**116 implementation-defined**

117 This definition is analogous to that of the ISO C standard and, together with *undefined* and  
118 *unspecified*, provides a range of specification of freedom allowed to the interface  
119 implementor.

**120 may**

121 The use of *may* has been limited as much as possible, due both to confusion stemming from  
122 its ordinary English meaning and to objections regarding the desirability of having as few  
123 options as possible and those as clearly specified as possible.

124 The usage of *can* and *may* were selected to contrast optional application behavior (*can*)  
125 against optional implementation behavior (*may*).

**126 shall**

127 Declarative sentences are sometimes used in IEEE Std 1003.1-200x as if they included the  
128 word *shall*, and facilities thus specified are no less required. For example, the two  
129 statements:

130 1. The *foo()* function shall return zero.

131 2. The *foo()* function returns zero.

132 are meant to be exactly equivalent.

**133 should**

134 In IEEE Std 1003.1-200x, the word *should* does not usually apply to the implementation, but  
135 rather to the application. Thus, the important words regarding implementations are *shall*,  
136 which indicates requirements, and *may*, which indicates options.

**137 obsolescent**

138 The term *obsolescent* means “do not use this feature in new applications”. The obsolescence  
139 concept is not an ideal solution, but was used as a method of increasing consensus: many  
140 more objections would be heard from the user community if some of these historical  
141 features were suddenly withdrawn without the grace period obsolescence implies. The  
142 phrase “may be considered for withdrawal in future revisions” implies that the result of  
143 that consideration might in fact keep those features indefinitely if the predominance of  
144 applications do not migrate away from them quickly.

**145 legacy**

146 The term *legacy* was added for compatibility with the Single UNIX Specification. It means  
147 “this feature is historic and optional; do not use this feature in new applications. There are  
148 alternate interfaces that are more suitable.”. It is used exclusively for XSI extensions, and  
149 includes facilities that were mandatory in previous versions of the base document but are  
150 optional in this revision. This is a way to “sunset” the usage of certain functions.  
151 Application writers should not rely on the existence of these facilities in new applications,  
152 but should follow the migration path detailed in the APPLICATION USAGE sections of the  
153 relevant pages.

154 The terms *legacy* and *obsolescent* are different: a feature marked LEGACY is not  
155 recommended for new work and need not be present on an implementation (if the XSI  
156 Legacy Option Group is not supported). A feature noted as obsolescent is supported by all  
157 implementations, but may be removed in a future revision; new applications should not use  
158 these features.

**159 system documentation**

160 The system documentation should normally describe the whole of the implementation,  
161 including any extensions provided by the implementation. Such documents normally  
162 contain information at least as detailed as the specifications in IEEE Std 1003.1-200x. Few

163 requirements are made on the system documentation, but the term is needed to avoid a  
 164 dangling pointer where the conformance document is permitted to point to the system  
 165 documentation.

166 **undefined**

167 See *implementation-defined*.

168 **unspecified**

169 See *implementation-defined*.

170 The definitions for *unspecified* and *undefined* appear nearly identical at first examination, but  
 171 are not. The term *unspecified* means that a conforming program may deal with the  
 172 unspecified behavior, and it should not care what the outcome is. The term *undefined* says  
 173 that a conforming program should not do it because no definition is provided for what it  
 174 does (and implicitly it would care what the outcome was if it tried it). It is important to  
 175 remember, however, that if the syntax permits the statement at all, it must have some  
 176 outcome in a real implementation.

177 Thus, the terms *undefined* and *unspecified* apply to the way the application should think  
 178 about the feature. In terms of the implementation, it is always “defined”—there is always  
 179 some result, even if it is an error. The implementation is free to choose the behavior it  
 180 prefers.

181 This also implies that an implementation, or another standard, could specify or define the  
 182 result in a useful fashion. The terms apply to IEEE Std 1003.1-200x specifically.

183 The term *implementation-defined* implies requirements for documentation that are not  
 184 required for *undefined* (or *unspecified*). Where there is no need for a conforming program to  
 185 know the definition, the term *undefined* is used, even though *implementation-defined* could  
 186 also have been used in this context. There could be a fourth term, specifying “this standard  
 187 does not say what this does; it is acceptable to define it in an implementation, but it does not  
 188 need to be documented”, and *undefined* would then be used very rarely for the few things  
 189 for which any definition is not useful. In particular, *implementation-defined* is used where it  
 190 is believed that certain classes of application will need to know such details to determine  
 191 whether the application can be successfully ported to the implementation. Such  
 192 applications are not always strictly portable, but nevertheless are common and useful; often  
 193 the requirements met by the application cannot be met without dealing with the issues  
 194 implied by “*implementation-defined*”.

195 In many places IEEE Std 1003.1-200x is silent about the behavior of some possible construct.  
 196 For example, a variable may be defined for a specified range of values and behaviors are  
 197 described for those values; nothing is said about what happens if the variable has any other  
 198 value. That kind of silence can imply an error in the standard, but it may also imply that the  
 199 standard was intentionally silent and that any behavior is permitted. There is a natural  
 200 tendency to infer that if the standard is silent, a behavior is prohibited. That is not the intent.  
 201 Silence is intended to be equivalent to the term *unspecified*.

202 The term *application* is not defined in IEEE Std 1003.1-200x; it is assumed to be a part of  
 203 general computer science terminology.

204 Three terms used within IEEE Std 1003.1-200x overlap in meaning: “macro”, “symbolic name”, |  
 205 and “symbolic constant”. |

206 **macro** |

207 This usually describes a C preprocessor symbol, the result of the **#define** operator, with or |  
 208 without an argument. It may also be used to describe similar mechanisms in editors and |  
 209 text processors. |

210 **symbolic name** |  
 211 This can also refer to a C preprocessor symbol (without arguments), but is also used to refer |  
 212 to the names for characters in character sets. In addition, it is sometimes used to refer to |  
 213 host names and even filenames. |

214 **symbolic constant** |  
 215 This also refers to a C preprocessor symbol (also without arguments). |

216 In most cases, the difference in semantic content is negligible to nonexistent. Readers should not |  
 217 attempt to read any meaning into the various usages of these terms. |

## 218 A.1.5 Portability

219 To aid the identification of options within IEEE Std 1003.1-200x, a notation consisting of margin |  
 220 codes and shading is used. This is based on the notation used in previous revisions of The Open |  
 221 Group's Base specifications.

222 The benefits of this approach is a reduction in the number of *if* statements within the running |  
 223 text, that makes the text easier to read, and also an identification to the programmer that they |  
 224 need to ensure that their target platforms support the underlying options. For example, if |  
 225 functionality is marked with THR in the margin, it will be available on all systems supporting |  
 226 the Threads option, but may not be available on some others.

### 227 A.1.5.1 Codes

228 This section includes codes for options defined in the Base Definitions volume of |  
 229 IEEE Std 1003.1-200x, Section 2.1.6, Options, and the following additional codes for other |  
 230 purposes:

231 CX This margin code is used to denote extensions beyond the ISO C standard. For |  
 232 interfaces that are duplicated between IEEE Std 1003.1-200x and the ISO C standard, a |  
 233 CX introduction block describes the nature of the duplication, with any extensions |  
 234 appropriately CX marked and shaded. |

235 Where an interface is added to an ISO C standard header, within the header the |  
 236 interface has an appropriate margin marker and shading (for example, CX, XSI, TSF, |  
 237 and so on) and the same marking appears on the reference page in the SYNOPSIS |  
 238 section. This enables a programmer to easily identify that the interface is extending an |  
 239 ISO C standard header. |

240 MX This margin code is used to denote IEC 60559: 1989 standard floating-point extensions.

241 OB This margin code is used to denote obsolescent behavior and thus flag a possible future |  
 242 application portability warning.

243 OH The Single UNIX Specification has historically tried to reduce the number of headers an |  
 244 application has had to include when using a particular interface. Sometimes this was |  
 245 fewer than the base standard, and hence a notation is used to flag which headers are |  
 246 optional if you are using a system supporting the XSI extension. |

247 XSI This code is used to denote interfaces and facilities within interfaces only required on |  
 248 systems supporting the XSI extension. This is introduced to support the Single UNIX |  
 249 Specification.

250 XSR This code is used to denote interfaces and facilities within interfaces only required on |  
 251 systems supporting STREAMS. This is introduced to support the Single UNIX |  
 252 Specification, although it is defined in a way so that it can standalone from the XSI |  
 253 notation.

254 A.1.5.2 *Margin Code Notation*

255 Since some features may depend on one or more options, or require more than one options, a  
256 notation is used. Where a feature requires support of a single option, a single margin code will  
257 occur in the margin. If it depends on two options and both are required, then the codes will  
258 appear with a <space> separator. If either of two options are required then a logical OR is  
259 denoted using the ' | ' symbol. If more than two codes are used, a special notation is used.

260 **A.2 Conformance**

261 The terms *profile* and *profiling* are used throughout this section.

262 A profile of a standard or standards is a codified set of option selections, such that by being  
263 conformant to a profile, particular classes of users are specifically supported.

264 These conformance definitions are descended from those in the ISO POSIX-1: 1996 standard, but  
265 with changes for the following:

- 266 • The addition of profiling options, allowing larger profiles of options such as the XSI |  
267 extension used by the Single UNIX Specification. In effect, it has profiled itself (that is, |  
268 created a self-profile). |
- 269 • The addition of a definition of subprofiling considerations, to allow smaller profiles of |  
270 options. |
- 271 • The addition of a hierarchy of super-options for XSI; these were formerly known as *Feature* |  
272 *Groups* in The Open Group System Interfaces and Headers, Issue 5 specification. |
- 273 • Options from the ISO POSIX-2: 1993 standard are also now included as IEEE Std 1003.1-200x |  
274 merges the functionality from it.

275 **A.2.1 Implementation Conformance**

276 These definitions allow application developers to know what to depend on in an  
277 implementation.

278 There is no definition of a *strictly conforming implementation*; that would be an implementation  
279 that provides *only* those facilities specified by POSIX.1 with no extensions whatsoever. This is  
280 because no actual operating system implementation can exist without system administration  
281 and initialization facilities that are beyond the scope of POSIX.1.

282 A.2.1.1 *Requirements*

283 The word “support” is used in certain instances, rather than “provide”, in order to allow an  
284 implementation that has no resident software development facilities, but that supports the  
285 execution of a *Strictly Conforming POSIX.1 Application*, to be a *conforming implementation*.

286 A.2.1.2 *Documentation*

287 The conformance documentation is required to use the same numbering scheme as POSIX.1 for  
288 purposes of cross-referencing. All options that an implementation chooses shall be reflected in  
289 <limits.h> and <unistd.h>.

290 Note that the use of “may” in terms of where conformance documents record where  
291 implementations may vary, implies that it is not required to describe those features identified as  
292 undefined or unspecified.

293 Other aspects of systems must be evaluated by purchasers for suitability. Many systems  
294 incorporate buffering facilities, maintaining updated data in volatile storage and transferring  
295 such updates to non-volatile storage asynchronously. Various exception conditions, such as a  
296 power failure or a system crash, can cause this data to be lost. The data may be associated with a  
297 file that is still open, with one that has been closed, with a directory, or with any other internal  
298 system data structures associated with permanent storage. This data can be lost, in whole or  
299 part, so that only careful inspection of file contents could determine that an update did not  
300 occur.

301 Also, interrelated file activities, where multiple files and/or directories are updated, or where  
302 space is allocated or released in the file system structures, can leave inconsistencies in the  
303 relationship between data in the various files and directories, or in the file system itself. Such  
304 inconsistencies can break applications that expect updates to occur in a specific sequence, so that  
305 updates in one place correspond with related updates in another place.

306 For example, if a user creates a file, places information in the file, and then records this action in  
307 another file, a system or power failure at this point followed by restart may result in a state in  
308 which the record of the action is permanently recorded, but the file created (or some of its  
309 information) has been lost. The consequences of this to the user may be undesirable. For a user  
310 on such a system, the only safe action may be to require the system administrator to have a  
311 policy that requires, after any system or power failure, that the entire file system must be  
312 restored from the most recent backup copy (causing all intervening work to be lost).

313 The characteristics of each implementation will vary in this respect and may or may not meet the  
314 requirements of a given application or user. Enforcement of such requirements is beyond the  
315 scope of POSIX.1. It is up to the purchaser to determine what facilities are provided in an  
316 implementation that affect the exposure to possible data or sequence loss, and also what  
317 underlying implementation techniques and/or facilities are provided that reduce or limit such  
318 loss or its consequences.

### 319 A.2.1.3 *POSIX Conformance*

320 This really means conformance to the base standard; however, since this revision includes the  
321 core material of the Single UNIX Specification, the standard developers decided that it was  
322 appropriate to segment the conformance requirements into two, the former for the base  
323 standard, and the latter for the Single UNIX Specification.

324 Within POSIX.1 there are some symbolic constants that, if defined, indicate that a certain option  
325 is enabled. Other symbolic constants exist in POSIX.1 for other reasons.

326 As part of the revision some alignment has occurred of the options with the FIPS 151-2 profile on  
327 the POSIX.1-1990 standard. The following options from the POSIX.1-1990 standard are now  
328 mandatory:

- 329 • `_POSIX_JOB_CONTROL`
- 330 • `_POSIX_SAVED_IDS`
- 331 • `_POSIX_VDISABLE`

332 A POSIX-conformant system may support the XSI extensions of the Single UNIX Specification.  
333 This was intentional since the standard developers intend them to be upwards-compatible, so  
334 that a system conforming to the Single UNIX Specification can also conform to the base standard  
335 at the same time.

336 A.2.1.4 *XSI Conformance*

337 This section is added since the revision merges in the base volumes of the Single UNIX  
338 Specification.

339 XSI conformance can be thought of as a profile, selecting certain options from  
340 IEEE Std 1003.1-200x.

341 A.2.1.5 *Option Groups*

342 The concept of *Option Groups* is introduced to IEEE Std 1003.1-200x to allow collections of  
343 related functions or options to be grouped together. This has been used as follows: the *XSI*  
344 *Option Groups* have been created to allow super-options, collections of underlying options and  
345 related functions, to be collectively supported by XSI-conforming systems. These reflect the  
346 *Feature Groups* from The Open Group System Interfaces and Headers, Issue 5 specification.

347 The standard developers considered the matter of subprofiling and decided it was better to  
348 include an enabling mechanism rather than detailed normative requirements. A set of  
349 subprofiling options was developed and included later in this volume of IEEE Std 1003.1-200x as  
350 an informative illustration.

351 A.2.1.6 *Options*

352 The final subsections within *Implementation Conformance* list the core options within  
353 IEEE Std 1003.1-200x. This includes both options for the System Interfaces volume of  
354 IEEE Std 1003.1-200x and the Shell and Utilities volume of IEEE Std 1003.1-200x.

355 **A.2.2 Application Conformance**

356 These definitions guide users or adaptors of applications in determining on which  
357 implementations an application will run and how much adaptation would be required to make  
358 it run on others. These definitions are modeled after related ones in the ISO C standard.

359 POSIX.1 occasionally uses the expressions *portable application* or *conforming application*. As they  
360 are used, these are synonyms for any of these terms. The differences between the classes of  
361 application conformance relate to the requirements for other standards, the options supported  
362 (such as the XSI extension) or, in the case of the Conforming POSIX.1 Application Using  
363 Extensions, to implementation extensions. When one of the less explicit expressions is used, it  
364 should be apparent from the context of the discussion which of the more explicit names is  
365 appropriate

366 A.2.2.1 *Strictly Conforming POSIX Application*

367 This definition is analogous to that of a ISO C standard *conforming program*.

368 The major difference between a *Strictly Conforming POSIX Application* and a ISO C standard  
369 *strictly conforming program* is that the latter is not allowed to use features of POSIX that are not in  
370 the ISO C standard.

371 A.2.2.2 *Conforming POSIX Application*

372 Examples of <National Bodies> include ANSI, BSI, and AFNOR.

373 A.2.2.3 *Conforming POSIX Application Using Extensions*

374 Due to possible requirements for configuration or implementation characteristics in excess of the  
 375 specifications in <limits.h> or related to the hardware (such as array size or file space), not every  
 376 Conforming POSIX Application Using Extensions will run on every conforming  
 377 implementation.

378 A.2.2.4 *Strictly Conforming XSI Application*

379 This is intended to be upwards-compatible with the definition of a Strictly Conforming POSIX  
 380 Application, with the addition of the facilities and functionality included in the XSI extension.

381 A.2.2.5 *Conforming XSI Application Using Extensions*

382 Such applications may use extensions beyond the facilities defined by IEEE Std 1003.1-200x  
 383 including the XSI extension, but need to document the additional requirements.

384 **A.2.3 Language-Dependent Services for the C Programming Language**

385 POSIX.1 is, for historical reasons, both a specification of an operating system interface, shell and  
 386 utilities, and a C binding for that specification. Efforts had been previously undertaken to  
 387 generate a language-independent specification; however, that had failed, and the fact that the  
 388 ISO C standard is the *de facto* primary language on POSIX and the UNIX system makes this a  
 389 necessary and workable situation.

390 **A.2.4 Other Language-Related Specifications**

391 There is no additional rationale provided for this section.

392 **A.3 Definitions**

393 The definitions in this section are stated so that they can be used as exact substitutes for the  
 394 terms in text. They should not contain requirements or cross-references to sections within  
 395 IEEE Std 1003.1-200x; that is accomplished by using an informative note. In addition, the term  
 396 should not be included in its own definition. Where requirements or descriptions need to be  
 397 addressed but cannot be included in the definitions, due to not meeting the above criteria, these  
 398 occur in the General Concepts chapter.

399 In this revision, the definitions have been reworked extensively to meet style requirements and |  
 400 to include terms from the base documents (see the Scope). |

401 Many of these definitions are necessarily circular, and some of the terms (such as *process*) are |  
 402 variants of basic computing science terms that are inherently hard to define. Where some |  
 403 definitions are more conceptual and contain requirements, these appear in the General Concepts |  
 404 chapter. Those listed in this section appear in an alphabetical glossary format of terms.

405 Some definitions must allow extension to cover terms or facilities that are not explicitly  
 406 mentioned in IEEE Std 1003.1-200x. For example, the definition of *Extended Security Controls*  
 407 permits implementations beyond those defined in IEEE Std 1003.1-200x.

408 Some terms in the following list of notes do not appear in IEEE Std 1003.1-200x; these are |  
 409 marked prefixed with a asterisk (\*). Many of them have been specifically excluded from |  
 410 IEEE Std 1003.1-200x because they concern system administration, implementation, or other |  
 411 issues that are not specific to the programming interface. Those are marked with a reason, such |  
 412 as “implementation-defined”.

**413 Appropriate Privileges**

414 One of the fundamental security problems with many historical UNIX systems has been that the  
415 privilege mechanism is monolithic—a user has either no privileges or *all* privileges. Thus, a  
416 successful “trojan horse” attack on a privileged process defeats all security provisions.  
417 Therefore, POSIX.1 allows more granular privilege mechanisms to be defined. For many  
418 historical implementations of the UNIX system, the presence of the term *appropriate privileges* in  
419 POSIX.1 may be understood as a synonym for *superuser* (UID 0). However, other systems have  
420 emerged where this is not the case and each discrete controllable action has *appropriate privileges*  
421 associated with it. Because this mechanism is implementation-defined, it must be described in  
422 the conformance document. Although that description affects several parts of POSIX.1 where  
423 the term *appropriate privilege* is used, because the term *implementation-defined* only appears here,  
424 the description of the entire mechanism and its effects on these other sections belongs in this  
425 equivalent section of the conformance document. This is especially convenient for  
426 implementations with a single mechanism that applies in all areas, since it only needs to be  
427 described once.

**428 Byte**

429 The restriction that a byte is now exactly eight bits was a conscious decision by the standard  
430 developers. It came about due to a combination of factors, primarily the use of the type **int8\_t**  
431 within the networking functions and the alignment with the ISO/IEC 9899:1999 standard, where  
432 the **intN\_t** types are now defined.

433 According to the ISO/IEC 9899:1999 standard:

- 434 • The **[u]intN\_t** types must be two’s complement with no padding bits and no illegal values.
- 435 • All types (apart from bit fields, which are not relevant here) must occupy an integral number  
436 of bytes.
- 437 • If a type with width  $W$  occupies  $B$  bytes with  $C$  bits per byte ( $C$  is the value of {CHAR\_BIT}),  
438 then it has  $P$  padding bits where  $P+W=B*C$ .
- 439 • For **int8\_t** we therefore have  $P=0$ ,  $W=8$ . Since  $B \geq 1$ ,  $C \geq 8$ , the only solution is  $B=1$ ,  $C=8$ .

440 The standard developers also felt that this was not an undue restriction for the current state of  
441 the art for this version of IEEE Std 1003.1-200x, but recognize that if industry trends continue, a  
442 wider character type may be required in the future.

**443 Character**

444 The term *character* is used to mean a sequence of one or more bytes representing a single graphic  
445 symbol. The deviation in the exact text of the ISO C standard definition for *byte* meets the intent  
446 of the rationale of the ISO C standard also clears up the ambiguity raised by the term *basic*  
447 *execution character set*. The octet-minimum requirement is a reflection of the {CHAR\_BIT} value.

**448 Clock Tick**

449 The ISO C standard defines a similar interval for use by the *clock()* function. There is no  
450 requirement that these intervals be the same. In historical implementations these intervals are  
451 different.

452 **Command**

453 The terms *command* and *utility* are related but have distinct meanings. Command is defined as “a  
454 directive to a shell to perform a specific task”. The directive can be in the form of a single utility  
455 name (for example, *ls*), or the directive can take the form of a compound command (for example,  
456 “*ls | grep name | pr*”). A utility is a program that can be called by name from a shell.  
457 Issuing only the name of the utility to a shell is the equivalent of a one-word command. A utility  
458 may be invoked as a separate program that executes in a different process than the command  
459 language interpreter, or it may be implemented as a part of the command language interpreter.  
460 For example, the *echo* command (the directive to perform a specific task) may be implemented  
461 such that the *echo* utility (the logic that performs the task of echoing) is in a separate program; |  
462 therefore, it is executed in a process that is different from the command language interpreter. |  
463 Conversely, the logic that performs the *echo* utility could be built into the command language  
464 interpreter; therefore, it could execute in the same process as the command language interpreter.

465 The terms *tool* and *application* can be thought of as being synonymous with *utility* from the  
466 perspective of the operating system kernel. Tools, applications, and utilities historically have  
467 run, typically, in processes above the kernel level. Tools and utilities historically have been a part  
468 of the operating system non-kernel code and have performed system-related functions, such as  
469 listing directory contents, checking file systems, repairing file systems, or extracting system  
470 status information. Applications have not generally been a part of the operating system, and  
471 they perform non-system-related functions, such as word processing, architectural design,  
472 mechanical design, workstation publishing, or financial analysis. Utilities have most frequently  
473 been provided by the operating system distributor, applications by third-party software  
474 distributors, or by the users themselves. Nevertheless, IEEE Std 1003.1-200x does not  
475 differentiate between tools, utilities, and applications when it comes to receiving services from  
476 the system, a shell, or the standard utilities. (For example, the *xargs* utility invokes another  
477 utility; it would be of fairly limited usefulness if the users could not run their own applications  
478 in place of the standard utilities.) Utilities are not applications in the sense that they are not  
479 themselves subject to the restrictions of IEEE Std 1003.1-200x or any other standard—there is no  
480 requirement for *grep*, *stty*, or any of the utilities defined here to be any of the classes of  
481 conforming applications.

482 **Column Positions**

483 In most 1-byte character sets, such as ASCII, the concept of column positions is identical to  
484 character positions and to bytes. Therefore, it has been historically acceptable for some  
485 implementations to describe line folding or tab stops or table column alignment in terms of bytes  
486 or character positions. Other character sets pose complications, as they can have internal  
487 representations longer than one octet and they can have display characters that have different  
488 widths on the terminal screen or printer.

489 In IEEE Std 1003.1-200x the term *column positions* has been defined to mean character—not  
490 byte—positions in input files (such as “column position 7 of the FORTRAN input”). Output files  
491 describe the column position in terms of the display width of the narrowest printable character  
492 in the character set, adjusted to fit the characteristics of the output device. It is very possible that  
493 *n* column positions will not be able to hold *n* characters in some character sets, unless all of those  
494 characters are of the narrowest width. It is assumed that the implementation is aware of the  
495 width of the various characters, deriving this information from the value of *LC\_CTYPE*, and thus  
496 can determine how many column positions to allot for each character in those utilities where it is  
497 important.

498 The term *column position* was used instead of the more natural *column* because the latter is  
499 frequently used in the different contexts of columns of figures, columns of table values, and so  
500 on. Wherever confusion might result, these latter types of columns are referred to as *text*

501 *columns.*

### 502 **Controlling Terminal**

503 The question of which of possibly several special files referring to the terminal is meant is not  
504 addressed in POSIX.1. The filename */dev/tty* is a synonym for the controlling terminal associated  
505 with a process.

### 506 **Device Number\***

507 The concept is handled in *stat()* as *ID of device*.

### 508 **Direct I/O**

509 Historically, direct I/O refers to the system bypassing intermediate buffering, but may be  
510 extended to cover implementation-defined optimizations.

### 511 **Directory**

512 The format of the directory file is implementation-defined and differs radically between  
513 System V and 4.3 BSD. However, routines (derived from 4.3 BSD) for accessing directories and  
514 certain constraints on the format of the information returned by those routines are described in  
515 the *<dirent.h>* header.

### 516 **Directory Entry**

517 Throughout IEEE Std 1003.1-200x, the term *link* is used (about the *link()* function, for example)  
518 in describing the objects that point to files from directories.

### 519 **Display**

520 The Shell and Utilities volume of IEEE Std 1003.1-200x assigns precise requirements for the  
521 terms *display* and *write*. Some historical systems have chosen to implement certain utilities  
522 without using the traditional file descriptor model. For example, the *vi* editor might employ  
523 direct screen memory updates on a personal computer, rather than a *write()* system call. An  
524 instance of user prompting might appear in a dialog box, rather than with standard error. When  
525 the Shell and Utilities volume of IEEE Std 1003.1-200x uses the term *display*, the method of  
526 outputting to the terminal is unspecified; many historical implementations use *termcap* or  
527 *terminfo*, but this is not a requirement. The term *write* is used when the Shell and Utilities volume  
528 of IEEE Std 1003.1-200x mandates that a file descriptor be used and that the output can be  
529 redirected. However, it is assumed that when the writing is directly to the terminal (it has not  
530 been redirected elsewhere), there is no practical way for a user or test suite to determine whether  
531 a file descriptor is being used. Therefore, the use of a file descriptor is mandated only for the  
532 redirection case and the implementation is free to use any method when the output is not  
533 redirected. The verb *write* is used almost exclusively, with the very few exceptions of those  
534 utilities where output redirection need not be supported: *tabs*, *talk*, *tput*, and *vi*.

**535 Dot**

536 The symbolic name *dot* is carefully used in POSIX.1 to distinguish the working directory  
537 filename from a period or a decimal point.

**538 Dot-Dot**

539 Historical implementations permit the use of these filenames without their special meanings.  
540 Such use precludes any meaningful use of these filenames by a Conforming POSIX.1  
541 Application. Therefore, such use is considered an extension, the use of which makes an  
542 implementation non-conforming; see also Section A.4.11 (on page 3327).

**543 Epoch**

544 Historically, the origin of UNIX system time was referred to as “00:00:00 GMT, January 1, 1970”.  
545 Greenwich Mean Time is actually not a term acknowledged by the international standards  
546 community; therefore, this term, *Epoch*, is used to abbreviate the reference to the actual standard,  
547 Coordinated Universal Time.

**548 FIFO Special File**

549 See *pipe* in **Pipe** (on page 3314).

**550 File**

551 It is permissible for an implementation-defined file type to be non-readable or non-writable.

**552 File Classes**

553 These classes correspond to the historical sets of permission bits. The classes are general to  
554 allow implementations flexibility in expanding the access mechanism for more stringent security  
555 environments. Note that a process is in one and only one class, so there is no ambiguity.

**556 Filename**

557 At the present time, the primary responsibility for truncating filenames containing multi-byte  
558 characters must reside with the application. Some industry groups involved in  
559 internationalization believe that in the future the responsibility must reside with the kernel. For  
560 the moment, a clearer understanding of the implications of making the kernel responsible for  
561 truncation of multi-byte filenames is needed.

562 Character-level truncation was not adopted because there is no support in POSIX.1 that advises  
563 how the kernel distinguishes between single and multi-byte characters. Until that time, it must  
564 be incumbent upon application writers to determine where multi-byte characters must be  
565 truncated.

**566 File System**

567 Historically, the meaning of this term has been overloaded with two meanings: that of the  
568 complete file hierarchy, and that of a mountable subset of that hierarchy; that is, a mounted file  
569 system. POSIX.1 uses the term *file system* in the second sense, except that it is limited to the scope  
570 of a process (and a process’ root directory). This usage also clarifies the domain in which a file  
571 serial number is unique.

**572 Graphic Character**

573 This definition is made available for those definitions (in particular, *TZ*) which must exclude  
574 control characters.

**575 Group Database**

576 See **User Database** (on page 3322).

**577 Group File\***

578 Implementation-defined; see **User Database** (on page 3322).

**579 Historical Implementations\***

580 This refers to previously existing implementations of programming interfaces and operating  
581 systems that are related to the interface specified by POSIX.1.

**582 Hosted Implementation\***

583 This refers to a POSIX.1 implementation that is accomplished through interfaces from the  
584 POSIX.1 services to some alternate form of operating system kernel services. Note that the line  
585 between a hosted implementation and a native implementation is blurred, since most  
586 implementations will provide some services directly from the kernel and others through some  
587 indirect path. (For example, *fopen()* might use *open()*; or *mkfifo()* might use *mknod()*.) There is  
588 no necessary relationship between the type of implementation and its correctness, performance,  
589 and/or reliability.

**590 Implementation\***

591 This term is generally used instead of its synonym, *system*, to emphasize the consequences of  
592 decisions to be made by system implementors. Perhaps if no options or extensions to POSIX.1  
593 were allowed, this usage would not have occurred.

594 The term *specific implementation* is sometimes used as a synonym for *implementation*. This should  
595 not be interpreted too narrowly; both terms can represent a relatively broad group of systems.  
596 For example, a hardware vendor could market a very wide selection of systems that all used the  
597 same instruction set, with some systems desktop models and others large multi-user  
598 minicomputers. This wide range would probably share a common POSIX.1 operating system,  
599 allowing an application compiled for one to be used on any of the others; this is a  
600 [*specific*] *implementation*.

601 However, that wide range of machines probably has some differences between the models.  
602 Some may have different clock rates, different file systems, different resource limits, different  
603 network connections, and so on, depending on their sizes or intended usages. Even on two  
604 identical machines, the system administrators may configure them differently. Each of these  
605 different systems is known by the term *a specific instance of a specific implementation*. This term is  
606 only used in the portions of POSIX.1 dealing with runtime queries: *sysconf()* and *pathconf()*.

**607 Incomplete Pathname\***

608 Absolute pathname has been adequately defined.

**609 Job Control**

610 In order to understand the job control facilities in POSIX.1 it is useful to understand how they  
611 are used by a job control-cognizant shell to create the user interface effect of job control.

612 While the job control facilities supplied by POSIX.1 can, in theory, support different types of  
613 interactive job control interfaces supplied by different types of shells, there was historically one |  
614 particular interface that was most common when the standard was originally developed |  
615 (provided by BSD C Shell). This discussion describes that interface as a means of illustrating  
616 how the POSIX.1 job control facilities can be used.

617 Job control allows users to selectively stop (suspend) the execution of processes and continue  
618 (resume) their execution at a later point. The user typically employs this facility via the  
619 interactive interface jointly supplied by the terminal I/O driver and a command interpreter  
620 (shell).

621 The user can launch jobs (command pipelines) in either the foreground or background. When  
622 launched in the foreground, the shell waits for the job to complete before prompting for  
623 additional commands. When launched in the background, the shell does not wait, but  
624 immediately prompts for new commands.

625 If the user launches a job in the foreground and subsequently regrets this, the user can type the  
626 suspend character (typically set to <control>-Z), which causes the foreground job to stop and the  
627 shell to begin prompting for new commands. The stopped job can be continued by the user (via  
628 special shell commands) either as a foreground job or as a background job. Background jobs can  
629 also be moved into the foreground via shell commands.

630 If a background job attempts to access the login terminal (controlling terminal), it is stopped by  
631 the terminal driver and the shell is notified, which, in turn, notifies the user. (Terminal access  
632 includes *read()* and certain terminal control functions, and conditionally includes *write()*.) The  
633 user can continue the stopped job in the foreground, thus allowing the terminal access to  
634 succeed in an orderly fashion. After the terminal access succeeds, the user can optionally move  
635 the job into the background via the suspend character and shell commands.

**636 Implementing Job Control Shells**

637 The interactive interface described previously can be accomplished using the POSIX.1 job  
638 control facilities in the following way.

639 The key feature necessary to provide job control is a way to group processes into jobs. This  
640 grouping is necessary in order to direct signals to a single job and also to identify which job is in  
641 the foreground. (There is at most one job that is in the foreground on any controlling terminal at  
642 a time.)

643 The concept of *process groups* is used to provide this grouping. The shell places each job in a  
644 separate process group via the *setpgid()* function. To do this, the *setpgid()* function is invoked by  
645 the shell for each process in the job. It is actually useful to invoke *setpgid()* twice for each  
646 process: once in the child process, after calling *fork()* to create the process, but before calling one  
647 of the *exec* family of functions to begin execution of the program, and once in the parent shell  
648 process, after calling *fork()* to create the child. The redundant invocation avoids a race condition  
649 by ensuring that the child process is placed into the new process group before either the parent  
650 or the child relies on this being the case. The *process group ID* for the job is selected by the shell to  
651 be equal to the *process ID* of one of the processes in the job. Some shells choose to make one  
652 process in the job be the parent of the other processes in the job (if any). Other shells (for

653 example, the C Shell) choose to make themselves the parent of all processes in the pipeline (job).  
654 In order to support this latter case, the *setpgid()* function accepts a process group ID parameter  
655 since the correct process group ID cannot be inherited from the shell. The shell itself is  
656 considered to be a job and is the sole process in its own process group.

657 The shell also controls which job is currently in the foreground. A foreground and background  
658 job differ in two ways: the shell waits for a foreground command to complete (or stop) before  
659 continuing to read new commands, and the terminal I/O driver inhibits terminal access by  
660 background jobs (causing the processes to stop). Thus, the shell must work cooperatively with  
661 the terminal I/O driver and have a common understanding of which job is currently in the  
662 foreground. It is the user who decides which command should be currently in the foreground,  
663 and the user informs the shell via shell commands. The shell, in turn, informs the terminal I/O  
664 driver via the *tcsetpgrp()* function. This indicates to the terminal I/O driver the process group ID  
665 of the foreground process group (job). When the current foreground job either stops or  
666 terminates, the shell places itself in the foreground via *tcsetpgrp()* before prompting for  
667 additional commands. Note that when a job is created the new process group begins as a  
668 background process group. It requires an explicit act of the shell via *tcsetpgrp()* to move a  
669 process group (job) into the foreground.

670 When a process in a job stops or terminates, its parent (for example, the shell) receives  
671 synchronous notification by calling the *waitpid()* function with the WUNTRACED flag set.  
672 Asynchronous notification is also provided when the parent establishes a signal handler for  
673 SIGCHLD and does not specify the SA\_NOCLDSTOP flag. Usually all processes in a job stop as  
674 a unit since the terminal I/O driver always sends job control stop signals to all processes in the  
675 process group.

676 To continue a stopped job, the shell sends the SIGCONT signal to the process group of the job. In  
677 addition, if the job is being continued in the foreground, the shell invokes *tcsetpgrp()* to place the  
678 job in the foreground before sending SIGCONT. Otherwise, the shell leaves itself in the  
679 foreground and reads additional commands.

680 There is additional flexibility in the POSIX.1 job control facilities that allows deviations from the  
681 typical interface. Clearing the TOSTOP terminal flag allows background jobs to perform *write()*  
682 functions without stopping. The same effect can be achieved on a per-process basis by having a  
683 process set the signal action for SIGTTOU to SIG\_IGN.

684 Note that the terms *job* and *process group* can be used interchangeably. A login session that is not  
685 using the job control facilities can be thought of as a large collection of processes that are all in  
686 the same job (process group). Such a login session may have a partial distinction between  
687 foreground and background processes; that is, the shell may choose to wait for some processes  
688 before continuing to read new commands and may not wait for other processes. However, the  
689 terminal I/O driver will consider all these processes to be in the foreground since they are all  
690 members of the same process group.

691 In addition to the basic job control operations already mentioned, a job control-cognizant shell  
692 needs to perform the following actions.

693 When a foreground (not background) job stops, the shell must sample and remember the current  
694 terminal settings so that it can restore them later when it continues the stopped job in the  
695 foreground (via the *tcgetattr()* and *tcsetattr()* functions).

696 Because a shell itself can be spawned from a shell, it must take special action to ensure that  
697 subshells interact well with their parent shells.

698 A subshell can be spawned to perform an interactive function (prompting the terminal for  
699 commands) or a non-interactive function (reading commands from a file). When operating non-  
700 interactively, the job control shell will refrain from performing the job control-specific actions

701 described above. It will behave as a shell that does not support job control. For example, all *jobs*  
702 will be left in the same process group as the shell, which itself remains in the process group  
703 established for it by its parent. This allows the shell and its children to be treated as a single job  
704 by a parent shell, and they can be affected as a unit by terminal keyboard signals.

705 An interactive subshell can be spawned from another job control-cognizant shell in either the  
706 foreground or background. (For example, from the C Shell, the user can execute the command,  
707 `csch &`.) Before the subshell activates job control by calling `setpgid()` to place itself in its own  
708 process group and `tcsetpgrp()` to place its new process group in the foreground, it needs to  
709 ensure that it has already been placed in the foreground by its parent. (Otherwise, there could  
710 be multiple job control shells that simultaneously attempt to control mediation of the terminal.)  
711 To determine this, the shell retrieves its own process group via `getpgrp()` and the process group  
712 of the current foreground job via `tcgetpgrp()`. If these are not equal, the shell sends SIGTTIN to  
713 its own process group, causing itself to stop. When continued later by its parent, the shell  
714 repeats the process group check. When the process groups finally match, the shell is in the  
715 foreground and it can proceed to take control. After this point, the shell ignores all the job  
716 control stop signals so that it does not inadvertently stop itself.

### 717 *Implementing Job Control Applications*

718 Most applications do not need to be aware of job control signals and operations; the intuitively  
719 correct behavior happens by default. However, sometimes an application can inadvertently  
720 interfere with normal job control processing, or an application may choose to overtly effect job  
721 control in cooperation with normal shell procedures.

722 An application can inadvertently subvert job control processing by “blindly” altering the  
723 handling of signals. A common application error is to learn how many signals the system  
724 supports and to ignore or catch them all. Such an application makes the assumption that it does  
725 not know what this signal is, but knows the right handling action for it. The system may  
726 initialize the handling of job control stop signals so that they are being ignored. This allows  
727 shells that do not support job control to inherit and propagate these settings and hence to be  
728 immune to stop signals. A job control shell will set the handling to the default action and  
729 propagate this, allowing processes to stop. In doing so, the job control shell is taking  
730 responsibility for restarting the stopped applications. If an application wishes to catch the stop  
731 signals itself, it should first determine their inherited handling states. If a stop signal is being  
732 ignored, the application should continue to ignore it. This is directly analogous to the  
733 recommended handling of SIGINT described in the referenced UNIX Programmer’s Manual.

734 If an application is reading the terminal and has disabled the interpretation of special characters  
735 (by clearing the ISIG flag), the terminal I/O driver will not send SIGTSTP when the suspend  
736 character is typed. Such an application can simulate the effect of the suspend character by  
737 recognizing it and sending SIGTSTP to its process group as the terminal driver would have  
738 done. Note that the signal is sent to the process group, not just to the application itself; this  
739 ensures that other processes in the job also stop. (Note also that other processes in the job could  
740 be children, siblings, or even ancestors.) Applications should not assume that the suspend  
741 character is `<control>-Z` (or any particular value); they should retrieve the current setting at  
742 startup.

743 *Implementing Job Control Systems*

744 The intent in adding 4.2 BSD-style job control functionality was to adopt the necessary 4.2 BSD  
745 programmatic interface with only minimal changes to resolve syntactic or semantic conflicts  
746 with System V or to close recognized security holes. The goal was to maximize the ease of  
747 providing both conforming implementations and Conforming POSIX.1 Applications.

748 It is only useful for a process to be affected by job control signals if it is the descendant of a job  
749 control shell. Otherwise, there will be nothing that continues the stopped process.

750 POSIX.1 does not specify how controlling terminal access is affected by a user logging out (that  
751 is, by a controlling process terminating). 4.2 BSD uses the *vhangup()* function to prevent any  
752 access to the controlling terminal through file descriptors opened prior to logout. System V does  
753 not prevent controlling terminal access through file descriptors opened prior to logout (except  
754 for the case of the special file, */dev/tty*). Some implementations choose to make processes  
755 immune from job control after logout (that is, such processes are always treated as if in the  
756 foreground); other implementations continue to enforce foreground/background checks after  
757 logout. Therefore, a Conforming POSIX.1 Application should not attempt to access the  
758 controlling terminal after logout since such access is unreliable. If an implementation chooses to  
759 deny access to a controlling terminal after its controlling process exits, POSIX.1 requires a certain  
760 type of behavior (see **Controlling Terminal** (on page 3305)).

761 **Kernel\***

762 See *system call*.

763 **Library Routine\***

764 See *system call*.

765 **Logical Device\***

766 Implementation-defined.

767 **Map**

768 The definition of map is included to clarify the usage of mapped pages in the description of the  
769 behavior of process memory locking.

770 **Memory-Resident**

771 The term *memory-resident* is historically understood to mean that the so-called resident pages are  
772 actually present in the physical memory of the computer system and are immune from  
773 swapping, paging, copy-on-write faults, and so on. This is the actual intent of  
774 IEEE Std 1003.1-200x in the process memory locking section for implementations where this is  
775 logical. But for some implementations—primarily mainframes—actually locking pages into  
776 primary storage is not advantageous to other system objectives, such as maximizing throughput.  
777 For such implementations, memory locking is a “hint” to the implementation that the  
778 application wishes to avoid situations that would cause long latencies in accessing memory.  
779 Furthermore, there are other implementation-defined issues with minimizing memory access  
780 latencies that “memory residency” does not address—such as MMU reload faults. The definition  
781 attempts to accommodate various implementations while allowing conforming applications to  
782 specify to the implementation that they want or need the best memory access times that the  
783 implementation can provide.

**784 Memory Object\***

785 The term *memory object* usually implies shared memory. If the object is the same as a filename in  
786 the file system name space of the implementation, it is expected that the data written into the  
787 memory object be preserved on disk. A memory object may also apply to a physical device on an  
788 implementation. In this case, writes to the memory object are sent to the controller for the device  
789 and reads result in control registers being returned.

**790 Mount Point\***

791 The directory on which a *mounted file system* is mounted. This term, like *mount()* and *umount()*,  
792 was not included because it was implementation-defined.

**793 Mounted File System\***

794 See *file system*.

**795 Name**

796 There are no explicit limits in IEEE Std 1003.1-200x on the sizes of names, words (see the  
797 definition of word in the Base Definitions volume of IEEE Std 1003.1-200x ), lines, or other  
798 objects. However, other implicit limits do apply: shell script lines produced by many of the  
799 standard utilities cannot exceed {LINE\_MAX} and the sum of exported variables comes under  
800 the {ARG\_MAX} limit. Historical shells dynamically allocate memory for names and words and  
801 parse incoming lines a character at a time. Lines cannot have an arbitrary {LINE\_MAX} limit  
802 because of historical practice, such as makefiles, where *make* removes the <newline>s associated  
803 with the commands for a target and presents the shell with one very long line. The text on  
804 INPUT FILES in the Shell and Utilities volume of IEEE Std 1003.1-200x, Section 1.11, Utility  
805 Description Defaults does allow a shell to run out of memory, but it cannot have arbitrary  
806 programming limits.

**807 Native Implementation\***

808 This refers to an implementation of POSIX.1 that interfaces directly to an operating system  
809 kernel; see also *hosted implementation* and *cooperating implementation*. A similar concept is a  
810 native UNIX system, which would be a kernel derived from one of the original UNIX system  
811 products.

**812 Nice Value**

813 This definition is not intended to suggest that all processes in a system have priorities that are  
814 comparable. Scheduling policy extensions, such as adding realtime priorities, make the notion of  
815 a single underlying priority for all scheduling policies problematic. Some implementations may |  
816 implement the features related to *nice* to affect all processes on the system, others to affect just |  
817 the general time-sharing activities implied by IEEE Std 1003.1-200x, and others may have no  
818 effect at all. Because of the use of “implementation-defined” in *nice* and *renice*, a wide range of  
819 implementation strategies is possible.

**820 Open File Description**

821 An *open file description*, as it is currently named, describes how a file is being accessed. What is  
822 currently called a *file descriptor* is actually just an identifier or “handle”; it does not actually  
823 describe anything.

824 The following alternate names were discussed:

- 825 • For *open file description*:  
826 *open instance, file access description, open file information, and file access information.*
- 827 • For *file descriptor*:  
828 *file handle, file number (c.f., `fileno()`). Some historical implementations use the term *file table*  
829 *entry.**

**830 Orphaned Process Group**

831 Historical implementations have a concept of an orphaned process, which is a process whose  
832 parent process has exited. When job control is in use, it is necessary to prevent processes from  
833 being stopped in response to interactions with the terminal after they no longer are controlled by  
834 a job control-cognizant program. Because signals generated by the terminal are sent to a process  
835 group and not to individual processes, and because a signal may be provoked by a process that  
836 is not orphaned, but sent to another process that is orphaned, it is necessary to define an  
837 orphaned process group. The definition assumes that a process group will be manipulated as a  
838 group and that the job control-cognizant process controlling the group is outside of the group  
839 and is the parent of at least one process in the group (so that state changes may be reported via  
840 *waitpid()*). Therefore, a group is considered to be controlled as long as at least one process in the  
841 group has a parent that is outside of the process group, but within the session.

842 This definition of orphaned process groups ensures that a session leader’s process group is  
843 always considered to be orphaned, and thus it is prevented from stopping in response to  
844 terminal signals.

**845 Page**

846 The term *page* is defined to support the description of the behavior of memory mapping for  
847 shared memory and memory mapped files, and the description of the behavior of process  
848 memory locking. It is not intended to imply that shared memory/file mapping and memory  
849 locking are applicable only to “paged” architectures. For the purposes of IEEE Std 1003.1-200x,  
850 whatever the granularity on which an architecture supports mapping or locking is considered to  
851 be a “page” . If an architecture cannot support the memory mapping or locking functions  
852 specified by IEEE Std 1003.1-200x on any granularity, then these options will not be  
853 implemented on the architecture.

**854 Passwd File\***

855 Implementation-defined; see **User Database** (on page 3322).

**856 Parent Directory**

857 There may be more than one directory entry pointing to a given directory in some  
858 implementations. The wording here identifies that exactly one of those is the parent directory. In  
859 *pathname resolution*, dot-dot is identified as the way that the unique directory is identified. (That  
860 is, the parent directory is the one to which dot-dot points.) In the case of a remote file system, if  
861 the same file system is mounted several times, it would appear as if they were distinct file  
862 systems (with interesting synchronization properties).

**863 Pipe**

864 It proved convenient to define a pipe as a special case of a FIFO, even though historically the  
865 latter was not introduced until System III and does not exist at all in 4.3 BSD.

**866 Portable Filename Character Set**

867 The encoding of this character set is not specified—specifically, ASCII is not required. But the  
868 implementation must provide a unique character code for each of the printable graphics  
869 specified by POSIX.1; see also Section A.4.6 (on page 3324).

870 Situations where characters beyond the portable filename character set (or historically ASCII or  
871 the ISO/IEC 646:1991 standard) would be used (in a context where the portable filename  
872 character set or the ISO/IEC 646:1991 standard is required by POSIX.1) are expected to be  
873 common. Although such a situation renders the use technically non-compliant, mutual  
874 agreement among the users of an extended character set will make such use portable between  
875 those users. Such a mutual agreement could be formalized as an optional extension to POSIX.1.  
876 (Making it required would eliminate too many possible systems, as even those systems using the  
877 ISO/IEC 646:1991 standard as a base character set extend their character sets for Western  
878 Europe and the rest of the world in different ways.)

879 Nothing in POSIX.1 is intended to preclude the use of extended characters where interchange is  
880 not required or where mutual agreement is obtained. It has been suggested that in several places  
881 “should” be used instead of “shall”. Because (in the worst case) use of any character beyond the  
882 portable filename character set would render the program or data not portable to all possible  
883 systems, no extensions are permitted in this context.

**884 Regular File**

885 POSIX.1 does not intend to preclude the addition of structuring data (for example, record  
886 lengths) in the file, as long as such data is not visible to an application that uses the features  
887 described in POSIX.1.

**888 Root Directory**

889 This definition permits the operation of *chroot()*, even though that function is not in POSIX.1; see  
890 also *file hierarchy*.

**891 Root File System\***

892 Implementation-defined.

893 **Root of a File System\***894 Implementation-defined; see *mount point*.895 **Signal**896 The definition implies a double meaning for the term. Although a signal is an event, common  
897 usage implies that a signal is an identifier of the class of event.898 **Superuser\***899 This concept, with great historical significance to UNIX system users, has been replaced with the  
900 notion of appropriate privileges.901 **Supplementary Group ID**902 The POSIX.1-1990 standard is inconsistent in its treatment of supplementary groups. The  
903 definition of supplementary group ID explicitly permits the effective group ID to be included in  
904 the set, but wording in the description of the *setuid()* and *setgid()* functions states: “Any  
905 supplementary group IDs of the calling process remain unchanged by these function calls”. In  
906 the case of *setgid()* this contradicts that definition. In addition, some felt that the unspecified  
907 behavior in the definition of supplementary group IDs adds unnecessary portability problems.  
908 The standard developers considered several solutions to this problem:

- 909 1. Reword the description of *setgid()* to permit it to change the supplementary group IDs to  
910 reflect the new effective group ID. A problem with this is that it adds more “may”s to the  
911 wording and does not address the portability problems of this optional behavior.
- 912 2. Mandate the inclusion of the effective group ID in the supplementary set (giving  
913 {NGROUPS\_MAX} a minimum value of 1). This is the behavior of 4.4 BSD. In that system,  
914 the effective group ID is the first element of the array of supplementary group IDs (there is  
915 no separate copy stored, and changes to the effective group ID are made only in the  
916 supplementary group set). By convention, the initial value of the effective group ID is  
917 duplicated elsewhere in the array so that the initial value is not lost when executing a set-  
918 group-ID program.
- 919 3. Change the definition of supplementary group ID to exclude the effective group ID and  
920 specify that the effective group ID does not change the set of supplementary group IDs.  
921 This is the behavior of 4.2 BSD, 4.3 BSD, and System V, Release 4.
- 922 4. Change the definition of supplementary group ID to exclude the effective group ID, and  
923 require that *getgroups()* return the union of the effective group ID and the supplementary  
924 group IDs.
- 925 5. Change the definition of {NGROUPS\_MAX} to be one more than the number of  
926 supplementary group IDs, so it continues to be the number of values returned by  
927 *getgroups()* and existing applications continue to work. This alternative is effectively the  
928 same as the second (and might actually have the same implementation).

929 The standard developers decided to permit either 2 or 3. The effective group ID is orthogonal to  
930 the set of supplementary group IDs, and it is implementation-defined whether *getgroups()*  
931 returns this. If the effective group ID is returned with the set of supplementary group IDs, then  
932 all changes to the effective group ID affect the supplementary group set returned by *getgroups()*.  
933 It is permissible to eliminate duplicates from the list returned by *getgroups()*. However, if a  
934 group ID is contained in the set of supplementary group IDs, setting the group ID to that value  
935 and then to a different value should not remove that value from the supplementary group IDs.

936 The definition of supplementary group IDs has been changed to not include the effective group  
937 ID. This simplifies permanent rationale and makes the relevant functions easier to understand.  
938 The *getgroups()* function has been modified so that it can, on an implementation-defined basis,  
939 return the effective group ID. By making this change, functions that modify the effective group  
940 ID do not need to discuss adding to the supplementary group list; the only view into the  
941 supplementary group list that the application writer has is through the *getgroups()* function.

## 942 **Symbolic Link**

943 Many implementations associate no attributes, including ownership with symbolic links.  
944 Security experts encouraged consideration for defining these attributes as optional.  
945 Consideration was given to changing *utime()* to allow modification of the times for a symbolic  
946 link, or as an alternative adding an *lutime()* interface. Modifications to *chown()* were also  
947 considered: allow changing symbolic link ownership or alternatively adding *lchown()*. As a  
948 result of the problems encountered in defining attributes for symbolic links (and interfaces to  
949 access/modify those attributes) and since implementations exist that do not associate these  
950 attributes with symbolic links, only the file type bits in the *st\_mode* member and the *st\_size*  
951 member of the **stat** structure are required to be applicable to symbolic links.

952 Historical implementations were followed when determining which interfaces should apply to  
953 symbolic links. Interfaces that historically followed symbolic links include *chmod()*, *link()*, and  
954 *utime()*. Interfaces that historically do not follow symbolic links include *chown()*, *lstat()*,  
955 *readlink()*, *rename()*, *remove()*, *rmdir()*, and *unlink()*. IEEE Std 1003.1-200x deviates from  
956 historical practice only in the case of *chown()*. Because there is no requirement that there be an  
957 association of ownership with symbolic links, there was no point in requiring an interface to  
958 change ownership. In addition, other implementations of symbolic links have modified *chown()*  
959 to follow symbolic links.

960 In the case of symbolic links, IEEE Std 1003.1-200x states that a trailing slash is considered to be  
961 the final component of a pathname rather than the pathname component that preceded it. This is  
962 the behavior of historical implementations. For example, for */a/b* and */a/b/*, if */a/b* is a symbolic  
963 link to a directory, then */a/b* refers to the symbolic link, and */a/b/* is the same as */a/b/.*, which is the  
964 directory to which the symbolic link points.

965 For multi-level security purposes, it is possible to have the link read mode govern permission for  
966 the *readlink()* function. It is also possible that the read permissions of the directory containing  
967 the link be used for this purpose. Implementations may choose to use either of these methods;  
968 however, this is not current practice and neither method is specified.

969 Several reasons were advanced for requiring that when a symbolic link is used as the source  
970 argument to the *link()* function, the resulting link will apply to the file named by the contents of  
971 the symbolic link rather than to the symbolic link itself. This is the case in historical  
972 implementations. This action was preferred, as it supported the traditional idea of persistence  
973 with respect to the target of a hard link. This decision is appropriate in light of a previous  
974 decision not to require association of attributes with symbolic links, thereby allowing  
975 implementations which do not use inodes. Opposition centered on the lack of symmetry on the  
976 part of the *link()* and *unlink()* function pair with respect to symbolic links.

977 Because a symbolic link and its referenced object coexist in the file system name space, confusion  
978 can arise in distinguishing between the link itself and the referenced object. Historically, utilities  
979 and system calls have adopted their own link following conventions in a somewhat *ad hoc*  
980 fashion. Rules for a uniform approach are outlined here, although historical practice has been  
981 adhered to as much as was possible. To promote consistent system use, user-written utilities are  
982 encouraged to follow these same rules.

983 Symbolic links are handled either by operating on the link itself, or by operating on the object  
 984 referenced by the link. In the latter case, an application or system call is said to follow the link.  
 985 Symbolic links may reference other symbolic links, in which case links are dereferenced until an  
 986 object that is not a symbolic link is found, a symbolic link that references a file that does not exist  
 987 is found, or a loop is detected. (Current implementations do not detect loops, but have a limit on  
 988 the number of symbolic links that they will dereference before declaring it an error.)

989 There are four domains for which default symbolic link policy is established in a system. In  
 990 almost all cases, there are utility options that override this default behavior. The four domains  
 991 are as follows:

- 992 1. Symbolic links specified to system calls that take filename arguments
- 993 2. Symbolic links specified as command line filename arguments to utilities that are not  
 994 performing a traversal of a file hierarchy
- 995 3. Symbolic links referencing files not of type directory, specified to utilities that are  
 996 performing a traversal of a file hierarchy
- 997 4. Symbolic links referencing files of type directory, specified to utilities that are performing a  
 998 traversal of a file hierarchy

#### 999 *First Domain*

1000 The first domain is considered in earlier rationale.

#### 1001 *Second Domain*

1002 The reason this category is restricted to utilities that are not traversing the file hierarchy is that  
 1003 some standard utilities take an option that specifies a hierarchical traversal, but by default  
 1004 operate on the arguments themselves. Generally, users specifying the option for a file hierarchy  
 1005 traversal wish to operate on a single, physical hierarchy, and therefore symbolic links, which  
 1006 may reference files outside of the hierarchy, are ignored. For example, *chown owner file* is a  
 1007 different operation from the same command with the **-R** option specified. In this example, the  
 1008 behavior of the command *chown owner file* is described here, while the behavior of the command  
 1009 *chown -R owner file* is described in the third and fourth domains.

1010 The general rule is that the utilities in this category follow symbolic links named as arguments.

1011 Exceptions in the second domain are:

- 1012 • The *mv* and *rm* utilities do not follow symbolic links named as arguments, but respectively  
 1013 attempt to rename or delete them.
- 1014 • The *ls* utility is also an exception to this rule. For compatibility with historical systems, when  
 1015 the **-R** option is not specified, the *ls* utility follows symbolic links named as arguments if the  
 1016 **-L** option is specified or if the **-F**, **-d**, or **-l** options are not specified. (If the **-L** option is  
 1017 specified, *ls* always follows symbolic links; it is the only utility where the **-L** option affects its  
 1018 behavior even though a tree walk is not being performed.)

1019 All other standard utilities, when not traversing a file hierarchy, always follow symbolic links  
 1020 named as arguments.

1021 Historical practice is that the **-h** option is specified if standard utilities are to act upon symbolic  
 1022 links instead of upon their targets. Examples of commands that have historically had a **-h** option  
 1023 for this purpose are the *chgrp*, *chown*, *file*, and *test* utilities.

1024 *Third Domain*

1025 The third domain is symbolic links, referencing files not of type directory, specified to utilities  
1026 that are performing a traversal of a file hierarchy. (This includes symbolic links specified as  
1027 command line filename arguments or encountered during the traversal.)

1028 The intention of the Shell and Utilities volume of IEEE Std 1003.1-200x is that the operation that  
1029 the utility is performing is applied to the symbolic link itself, if that operation is applicable to  
1030 symbolic links. The reason that the operation is not required is that symbolic links in some  
1031 implementations do not have such attributes as a file owner, and therefore the *chown* operation  
1032 would be meaningless. If symbolic links on the system have an owner, it is the intention that the  
1033 utility *chown* cause the owner of the symbolic link to change. If symbolic links do not have an  
1034 owner, the symbolic link should be ignored. Specifically, by default, no change should be made  
1035 to the file referenced by the symbolic link.

1036 *Fourth Domain*

1037 The fourth domain is symbolic links referencing files of type directory, specified to utilities that  
1038 are performing a traversal of a file hierarchy. (This includes symbolic links specified as  
1039 command line filename arguments or encountered during the traversal.)

1040 Most standard utilities do not, by default, indirect into the file hierarchy referenced by the  
1041 symbolic link. (The Shell and Utilities volume of IEEE Std 1003.1-200x uses the informal term  
1042 *physical walk* to describe this case. The case where the utility does indirect through the symbolic  
1043 link is termed a *logical walk*.)

1044 There are three reasons for the default to a physical walk:

- 1045 1. With very few exceptions, a physical walk has been the historical default on UNIX systems  
1046 supporting symbolic links. Because some utilities (that is, *rm*) must default to a physical  
1047 walk, regardless, changing historical practice in this regard would be confusing to users  
1048 and needlessly incompatible.
- 1049 2. For systems where symbolic links have the historical file attributes (that is, *owner*, *group*,  
1050 *mode*), defaulting to a logical traversal would require the addition of a new option to the  
1051 commands to modify the attributes of the link itself. This is painful and more complex  
1052 than the alternatives.
- 1053 3. There is a security issue with defaulting to a logical walk. Historically, the command  
1054 *chown -R user file* has been safe for the superuser because *setuid* and *setgid* bits were lost  
1055 when the ownership of the file was changed. If the walk were logical, changing ownership  
1056 would no longer be safe because a user might have inserted a symbolic link pointing to any  
1057 file in the tree. Again, this would necessitate the addition of an option to the commands  
1058 doing hierarchy traversal to not indirect through the symbolic links, and historical scripts  
1059 doing recursive walks would instantly become security problems. While this is mostly an  
1060 issue for system administrators, it is preferable to not have different defaults for different  
1061 classes of users.

1062 As consistently as possible, users may cause standard utilities performing a file hierarchy  
1063 traversal to follow any symbolic links named on the command line, regardless of the type of file  
1064 they reference, by specifying the *-H* (for half logical) option. This option is intended to make the  
1065 command line name space look like the logical name space.

1066 As consistently as possible, users may cause standard utilities performing a file hierarchy  
1067 traversal to follow any symbolic links named on the command line as well as any symbolic links  
1068 encountered during the traversal, regardless of the type of file they reference, by specifying the  
1069 *-L* (for logical) option. This option is intended to make the entire name space look like the  
1070 logical name space.

1071 For consistency, implementors are encouraged to use the **-P** (for physical) flag to specify the  
1072 physical walk in utilities that do logical walks by default for whatever reason. The only standard  
1073 utilities that require the **-P** option are *cd* and *pwd*; see the note below.

1074 When one or more of the **-H**, **-L**, and **-P** flags can be specified, the last one specified determines  
1075 the behavior of the utility. This permits users to alias commands so that the default behavior is a  
1076 logical walk and then override that behavior on the command line.

#### 1077 *Exceptions in the Third and Fourth Domains*

1078 The *ls* and *rm* utilities are exceptions to these rules. The *rm* utility never follows symbolic links  
1079 and does not support the **-H**, **-L**, or **-P** options. Some historical versions of *ls* always followed  
1080 symbolic links given on the command line whether the **-L** option was specified or not. Historical  
1081 versions of *ls* did not support the **-H** option. In IEEE Std 1003.1-200x, unless one of the **-H** or **-L**  
1082 options is specified, the *ls* utility only follows symbolic links to directories that are given as  
1083 operands. The *ls* utility does not support the **-P** option.

1084 The Shell and Utilities volume of IEEE Std 1003.1-200x requires that the standard utilities *ls*, *find*,  
1085 and *pax* detect infinite loops when doing logical walks; that is, a directory, or more commonly a  
1086 symbolic link, that refers to an ancestor in the current file hierarchy. If the file system itself is  
1087 corrupted, causing the infinite loop, it may be impossible to recover. Because *find* and *ls* are often  
1088 used in system administration and security applications, they should attempt to recover and  
1089 continue as best as they can. The *pax* utility should terminate because the archive it was creating  
1090 is by definition corrupted. Other, less vital, utilities should probably simply terminate as well.  
1091 Implementations are strongly encouraged to detect infinite loops in all utilities.

1092 Historical practice is shown in Table A-1 (on page 3320). The heading **SVID3** stands for the  
1093 Third Edition of the System V Interface Definition.

1094 Historically, several shells have had built-in versions of the *pwd* utility. In some of these shells,  
1095 *pwd* reported the physical path, and in others, the logical path. Implementations of the shell  
1096 corresponding to IEEE Std 1003.1-200x must report the logical path by default. Earlier versions  
1097 of IEEE Std 1003.1-200x did not require the *pwd* utility to be a built-in utility. Now that *pwd* is  
1098 required to set an environment variable in the current shell execution environment, it must be a  
1099 built-in utility.

1100 The *cd* command is required, by default, to treat the filename dot-dot logically. Implementors are  
1101 required to support the **-P** flag in *cd* so that users can have their current environment handled  
1102 physically. In 4.3 BSD, *chgrp* during tree traversal changed the group of the symbolic link, not  
1103 the target. Symbolic links in 4.4 BSD do not have *owner*, *group*, *mode*, or other standard UNIX  
1104 system file attributes.

1105

Table A-1 Historical Practice for Symbolic Links

Utility	SVID3	4.3 BSD	4.4 BSD	POSIX	Comments
1106				-L	Treat ". ." logically.
1107				-P	". ." physically.
1108					
1109			-H	-H	Follow command line symlinks.
1110			-h	-L	Follow symlinks.
1111	-h			-h	Affect the symlink.
1112					Affect the symlink.
1113			-H		Follow command line symlinks.
1114			-h		Follow symlinks.
1115			-H	-H	Follow command line symlinks.
1116			-h	-L	Follow symlinks.
1117	-h			-h	Affect the symlink.
1118			-H	-H	Follow command line symlinks.
1119			-h	-L	Follow symlinks.
1120	-L		-L		Follow symlinks.
1121			-H	-H	Follow command line symlinks.
1122			-h	-L	Follow symlinks.
1123	-h			-h	Affect the symlink.
1124			-H	-H	Follow command line symlinks.
1125			-h	-L	Follow symlinks.
1126	-follow		-follow		Follow symlinks.
1127	-s	-s	-s	-s	Create a symbolic link.
1128	-L	-L	-L	-L	Follow symlinks.
1129				-H	Follow command line symlinks.
1130					Operates on the symlink.
1131			-H	-H	Follow command line symlinks.
1132			-h	-L	Follow symlinks.
1133				-L	Printed path may contain symlinks.
1134				-P	Printed path will not contain symlinks.
1135					Operates on the symlink.
1136			-H		Follow command line symlinks.
1137		-h	-h		Follow symlinks.
1138	-h		-h	-h	Affect the symlink.

1139 **Synchronously-Generated Signal**

1140 Those signals that may be generated synchronously include SIGABRT, SIGBUS, SIGILL, SIGFPE,  
1141 SIGPIPE, and SIGSEGV.

1142 Any signal sent via the *raise()* function or a *kill()* function targeting the current process is also  
1143 considered synchronous.

1144 **System Call\***

1145 The distinction between a *system call* and a *library routine* is an implementation detail that may  
1146 differ between implementations and has thus been excluded from POSIX.1.

1147 See "Interface, Not Implementation" in the Preface.

**1148 System Reboot**

1149 A *system reboot* is an event initiated by an unspecified circumstance that causes all processes  
1150 (other than special system processes) to be terminated in an implementation-defined manner,  
1151 after which any changes to the state and contents of files created or written to by a Conforming  
1152 POSIX.1 Application prior to the event are implementation-defined.

**1153 Synchronized I/O Data (and File) Integrity Completion**

1154 These terms specify that for synchronized read operations, pending writes must be successfully  
1155 completed before the read operation can complete. This is motivated by two circumstances.  
1156 Firstly, when synchronizing processes can access the same file, but not share common buffers  
1157 (such as for a remote file system), this requirement permits the reading process to guarantee that  
1158 it can read data written remotely. Secondly, having data written synchronously is insufficient to  
1159 guarantee the order with respect to a subsequent write by a reading process, and thus this extra  
1160 read semantic is necessary.

**1161 Text File**

1162 The term *text file* does not prevent the inclusion of control or other non-printable characters  
1163 (other than NUL). Therefore, standard utilities that list text files as inputs or outputs are either  
1164 able to process the special characters or they explicitly describe their limitations within their  
1165 individual descriptions. The definition of *text file* has caused controversy. The only difference  
1166 between text and binary files is that text files have lines of less than {LINE\_MAX} bytes, with no  
1167 NUL characters, each terminated by a <newline>. The definition allows a file with a single  
1168 <newline>, but not a totally empty file, to be called a text file. If a file ends with an incomplete  
1169 line it is not strictly a text file by this definition. The <newline> referred to in  
1170 IEEE Std 1003.1-200x is not some generic line separator, but a single character; files created on  
1171 systems where they use multiple characters for ends of lines are not portable to all conforming  
1172 systems without some translation process unspecified by IEEE Std 1003.1-200x.

**1173 Thread**

1174 IEEE Std 1003.1-200x defines a thread to be a flow of control within a process. Each thread has a  
1175 minimal amount of private state; most of the state associated with a process is shared among all  
1176 of the threads in the process. While most multi-thread extensions to POSIX have taken this  
1177 approach, others have made different decisions.

1178 **Note:** The choice to put threads within a process does not constrain implementations to implement  
1179 threads in that manner. However, all functions have to behave as though threads share the  
1180 indicated state information with the process from which they were created.

1181 Threads need to share resources in order to cooperate. Memory has to be widely shared between  
1182 threads in order for the threads to cooperate at a fine level of granularity. Threads keep data  
1183 structures and the locks protecting those data structures in shared memory. For a data structure  
1184 to be usefully shared between threads, such structures should not refer to any data that can only  
1185 be interpreted meaningfully by a single thread. Thus, any system resources that might be  
1186 referred to in data structures need to be shared between all threads. File descriptors, pathnames,  
1187 and pointers to stack variables are all things that programmers want to share between their  
1188 threads. Thus, the file descriptor table, the root directory, the current working directory, and the  
1189 address space have to be shared.

1190 Library implementations are possible as long as the effective behavior is as if system services  
1191 invoked by one thread do not suspend other threads. This may be difficult for some library  
1192 implementations on systems that do not provide asynchronous facilities.

1193 See Section B.2.9 (on page 3439) for additional rationale.

#### 1194 **Thread ID**

1195 See Section B.2.9.2 (on page 3455) for additional rationale.

#### 1196 **Thread-Safe Function**

1197 All functions required by IEEE Std 1003.1-200x need to be thread-safe; see Section A.4.16 (on  
1198 page 3330) and Section B.2.9.1 (on page 3452) for additional rationale.

#### 1199 **User Database**

1200 There are no references in IEEE Std 1003.1-200x to a *passwd file* or a *group file*, and there is no  
1201 requirement that the *group* or *passwd* databases be kept in files containing editable text. Many  
1202 large timesharing systems use *passwd* databases that are hashed for speed. Certain security  
1203 classifications prohibit certain information in the *passwd* database from being publicly readable.

1204 The term *encoded* is used instead of *encrypted* in order to avoid the implementation connotations  
1205 (such as reversibility or use of a particular algorithm) of the latter term.

1206 The *getgrent()*, *setgrent()*, *endgrent()*, *getpwent()*, *setpwent()*, and *endpwent()* functions are not  
1207 included as part of the base standard because they provide a linear database search capability  
1208 that is not generally useful (the *getpwuid()*, *getpwnam()*, *getgrgid()*, and *getgrnam()* functions are  
1209 provided for keyed lookup) and because in certain distributed systems, especially those with  
1210 different authentication domains, it may not be possible or desirable to provide an application  
1211 with the ability to browse the system databases indiscriminately. They are provided on XSI-  
1212 conformant systems due to their historical usage by many existing applications.

1213 A change from historical implementations is that the structures used by these functions have  
1214 fields of the types **gid\_t** and **uid\_t**, which are required to be defined in the `<sys/types.h>` header.  
1215 IEEE Std 1003.1-200x requires implementations to ensure that these types are defined by  
1216 inclusion of `<grp.h>` and `<pwd.h>`, respectively, without imposing any name space pollution or  
1217 errors from redefinition of types.

1218 IEEE Std 1003.1-200x is silent about the content of the strings containing user or group names.  
1219 These could be digit strings. IEEE Std 1003.1-200x is also silent as to whether such digit strings  
1220 bear any relationship to the corresponding (numeric) user or group ID.

#### 1221 *Database Access*

1222 The thread-safe versions of the user and group database access functions return values in user-  
1223 supplied buffers instead of possibly using static data areas that may be overwritten by each call.

#### 1224 **Virtual Processor\***

1225 The term *virtual processor* was chosen as a neutral term describing all kernel-level schedulable  
1226 entities, such as processes, Mach tasks, or lightweight processes. Implementing threads using  
1227 multiple processes as virtual processors, or implementing multiplexed threads above a virtual  
1228 processor layer, should be possible, provided some mechanism has also been implemented for  
1229 sharing state between processes or virtual processors. Many systems may also wish to provide  
1230 implementations of threads on systems providing “shared processes” or “variable-weight  
1231 processes”. It was felt that exposing such implementation details would severely limit the type  
1232 of systems upon which the threads interface could be supported and prevent certain types of  
1233 valid implementations. It was also determined that a virtual processor interface was out of the  
1234 scope of the Rationale (Informative) volume of IEEE Std 1003.1-200x.

1235 **XSI**

1236 This is introduced to allow IEEE Std 1003.1-200x to be adopted as an IEEE standard and an Open  
1237 Group Technical Standard, serving both the POSIX and the Single UNIX Specification in a core  
1238 set of volumes.

1239 The term *XSI* has been used for 10 years in connection with the XPG series and the first and  
1240 second versions of the base volumes of the Single UNIX Specification. The XSI margin code was  
1241 introduced to denote the extended or more restrictive semantics beyond POSIX that are  
1242 applicable to UNIX systems.

1243 **A.4 General Concepts**1244 **A.4.1 Concurrent Execution**

1245 There is no additional rationale provided for this section.

1246 **A.4.2 Directory Protection**

1247 There is no additional rationale provided for this section.

1248 **A.4.3 Extended Security Controls**

1249 Allowing an implementation to define extended security controls enables the use of  
1250 IEEE Std 1003.1-200x in environments that require different or more rigorous security than that  
1251 provided in POSIX.1. Extensions are allowed in two areas: privilege and file access permissions.  
1252 The semantics of these areas have been defined to permit extensions with reasonable, but not  
1253 exact, compatibility with all existing practices. For example, the elimination of the superuser  
1254 definition precludes identifying a process as privileged or not by virtue of its effective user ID.

1255 **A.4.4 File Access Permissions**

1256 A process should not try to anticipate the result of an attempt to access data by *a priori* use of  
1257 these rules. Rather, it should make the attempt to access data and examine the return value (and  
1258 possibly *errno* as well), or use *access()*. An implementation may include other security  
1259 mechanisms in addition to those specified in POSIX.1, and an access attempt may fail because of  
1260 those additional mechanisms, even though it would succeed according to the rules given in this  
1261 section. (For example, the user's security level might be lower than that of the object of the access  
1262 attempt.) The supplementary group IDs provide another reason for a process to not attempt to  
1263 anticipate the result of an access attempt.

1264 **A.4.5 File Hierarchy**

1265 Though the file hierarchy is commonly regarded to be a tree, POSIX.1 does not define it as such  
1266 for three reasons:

- 1267 1. Links may join branches.
- 1268 2. In some network implementations, there may be no single absolute root directory; see  
1269 *pathname resolution*.
- 1270 3. With symbolic links, the file system need not be a tree or even a directed acyclic graph.

1271 **A.4.6 Filenames**

1272 Historically, certain filenames have been reserved. This list includes **core**, **/etc/passwd**, and so on. Conforming applications should avoid these. |

1274 Most historical implementations prohibit case folding in filenames; that is, treating uppercase and lowercase alphabetic characters as identical. However, some consider case folding desirable:

- 1276 • For user convenience
- 1277 • For ease-of-implementation of the POSIX.1 interface as a hosted system on some popular  
1278 operating systems

1279 Variants, such as maintaining case distinctions in filenames, but ignoring them in comparisons, have been suggested. Methods of allowing escaped characters of the case opposite the default have been proposed.

1282 Many reasons have been expressed for not allowing case folding, including:

- 1283 • No solid evidence has been produced as to whether case-sensitivity or case-insensitivity is  
1284 more convenient for users.
- 1285 • Making case-insensitivity a POSIX.1 implementation option would be worse than either  
1286 having it or not having it, because:
  - 1287 — More confusion would be caused among users.
  - 1288 — Application developers would have to account for both cases in their code.
  - 1289 — POSIX.1 implementors would still have other problems with native file systems, such as  
1290 short or otherwise constrained filenames or pathnames, and the lack of hierarchical  
1291 directory structure.
- 1292 • Case folding is not easily defined in many European languages, both because many of them  
1293 use characters outside the US ASCII alphabetic set, and because:
  - 1294 — In Spanish, the digraph "ll" is considered to be a single letter, the capitalized form of  
1295 which may be either "Ll" or "LL", depending on context.
  - 1296 — In French, the capitalized form of a letter with an accent may or may not retain the accent,  
1297 depending on the country in which it is written.
  - 1298 — In German, the sharp ess may be represented as a single character resembling a Greek  
1299 beta (β) in lowercase, but as the digraph "SS" in uppercase.
  - 1300 — In Greek, there are several lowercase forms of some letters; the one to use depends on its  
1301 position in the word. Arabic has similar rules.
- 1302 • Many East Asian languages, including Japanese, Chinese, and Korean, do not distinguish  
1303 case and are sometimes encoded in character sets that use more than one byte per character.
- 1304 • Multiple character codes may be used on the same machine simultaneously. There are  
1305 several ISO character sets for European alphabets. In Japan, several Japanese character codes  
1306 are commonly used together, sometimes even in filenames; this is evidently also the case in  
1307 China. To handle case insensitivity, the kernel would have to at least be able to distinguish  
1308 for which character sets the concept made sense.
- 1309 • The file system implementation historically deals only with bytes, not with characters, except  
1310 for slash and the null byte.
- 1311 • The purpose of POSIX.1 is to standardize the common, existing definition, not to change it.  
1312 Mandating case-insensitivity would make all historical implementations non-standard.

1313 • Not only the interface, but also application programs would need to change, counter to the  
1314 purpose of having minimal changes to existing application code.

1315 • At least one of the original developers of the UNIX system has expressed objection in the  
1316 strongest terms to either requiring case-insensitivity or making it an option, mostly on the  
1317 basis that POSIX.1 should not hinder portability of application programs across related  
1318 implementations in order to allow compatibility with unrelated operating systems.

1319 Two proposals were entertained regarding case folding in filenames:

1320 1. Remove all wording that previously permitted case folding.

1321 Rationale Case folding is inconsistent with portable filename character set definition  
1322 and filename definition (all characters except slash and null). No known  
1323 implementations allowing all characters except slash and null also do case  
1324 folding.

1325 2. Change “though this practice is not recommended:” to “although this practice is strongly  
1326 discouraged.”

1327 Rationale If case folding must be included in POSIX.1, the wording should be stronger  
1328 to discourage the practice.

1329 The consensus selected the first proposal. Otherwise, a conforming application would have to  
1330 assume that case folding would occur when it was not wanted, but that it would not occur when  
1331 it was wanted.

#### 1332 **A.4.7 File Times Update**

1333 This section reflects the actions of historical implementations. The times are not updated  
1334 immediately, but are only marked for update by the functions. An implementation may update  
1335 these times immediately.

1336 The accuracy of the time update values is intentionally left unspecified so that systems can  
1337 control the bandwidth of a possible covert channel.

1338 The wording was carefully chosen to make it clear that there is no requirement that the  
1339 conformance document contain information that might incidentally affect file update times. Any  
1340 function that performs pathname resolution might update several *st\_atime* fields. Functions such  
1341 as *getpwnam()* and *getgrnam()* might update the *st\_atime* field of some specific file or files. It  
1342 is intended that these are not required to be documented in the conformance document, but they  
1343 should appear in the system documentation.

#### 1344 **A.4.8 Host and Network Byte Order**

1345 There is no additional rationale provided for this section.

#### 1346 **A.4.9 Measurement of Execution Time**

1347 The methods used to measure the execution time of processes and threads, and the precision of  
1348 these measurements, may vary considerably depending on the software architecture of the  
1349 implementation, and on the underlying hardware. Implementations can also make tradeoffs  
1350 between the scheduling overhead and the precision of the execution time measurements.  
1351 IEEE Std 1003.1-200x does not impose any requirement on the accuracy of the execution time; it  
1352 instead specifies that the measurement mechanism and its precision are implementation-  
1353 defined.

1354 **A.4.10 Memory Synchronization**

1355 In older multi-processors, access to memory by the processors was strictly multiplexed. This  
 1356 meant that a processor executing program code interrogates or modifies memory in the order  
 1357 specified by the code and that all the memory operation of all the processors in the system  
 1358 appear to happen in some global order, though the operation histories of different processors are  
 1359 interleaved arbitrarily. The memory operations of such machines are said to be sequentially  
 1360 consistent. In this environment, threads can synchronize using ordinary memory operations. For  
 1361 example, a producer thread and a consumer thread can synchronize access to a circular data  
 1362 buffer as follows:

```

1363     int rdptr = 0;
1364     int wrptr = 0;
1365     data_t buf[BUFSIZE];

1366     Thread 1:
1367         while (work_to_do) {
1368             int next;

1369             buf[wrptr] = produce();
1370             next = (wrptr + 1) % BUFSIZE;
1371             while (rdptr == next)
1372                 ;
1373             wrptr = next;
1374         }

1375     Thread 2:
1376         while (work_to_do) {
1377             while (rdptr == wrptr)
1378                 ;
1379             consume(buf[rdptr]);
1380             rdptr = (rdptr + 1) % BUFSIZE;
1381         }

```

1382 In modern multi-processors, these conditions are relaxed to achieve greater performance. If one  
 1383 processor stores values in location A and then location B, then other processors loading data  
 1384 from location B and then location A may see the new value of B but the old value of A. The  
 1385 memory operations of such machines are said to be weakly ordered. On these machines, the  
 1386 circular buffer technique shown in the example will fail because the consumer may see the new  
 1387 value of *wrptr* but the old value of the data in the buffer. In such machines, synchronization can  
 1388 only be achieved through the use of special instructions that enforce an order on memory  
 1389 operations. Most high-level language compilers only generate ordinary memory operations to  
 1390 take advantage of the increased performance. They usually cannot determine when memory  
 1391 operation order is important and generate the special ordering instructions. Instead, they rely on  
 1392 the programmer to use synchronization primitives correctly to ensure that modifications to a  
 1393 location in memory are ordered with respect to modifications and/or access to the same location  
 1394 in other threads. Access to read-only data need not be synchronized. The resulting program is  
 1395 said to be data race-free.

1396 Synchronization is still important even when accessing a single primitive variable (for example,  
 1397 an integer). On machines where the integer may not be aligned to the bus data width or be larger  
 1398 than the data width, a single memory load may require multiple memory cycles. This means  
 1399 that it may be possible for some parts of the integer to have an old value while other parts have a  
 1400 newer value. On some processor architectures this cannot happen, but portable programs cannot  
 1401 rely on this.

1402 In summary, a portable multi-threaded program, or a multi-process program that shares  
1403 writable memory between processes, has to use the synchronization primitives to synchronize  
1404 data access. It cannot rely on modifications to memory being observed by other threads in the  
1405 order written in the program or even on modification of a single variable being seen atomically.

1406 Conforming applications may only use the functions listed to synchronize threads of control  
1407 with respect to memory access. There are many other candidates for functions that might also be  
1408 used. Examples are: signal sending and reception, or pipe writing and reading. In general, any  
1409 function that allows one thread of control to wait for an action caused by another thread of  
1410 control is a candidate. IEEE Std 1003.1-200x does not require these additional functions to  
1411 synchronize memory access since this would imply the following:

- 1412 • All these functions would have to be recognized by advanced compilation systems so that  
1413 memory operations and calls to these functions are not reordered by optimization.
- 1414 • All these functions would potentially have to have memory synchronization instructions  
1415 added, depending on the particular machine.
- 1416 • The additional functions complicate the model of how memory is synchronized and make  
1417 automatic data race detection techniques impractical.

1418 Formal definitions of the memory model were rejected as unreadable by the vast majority of  
1419 programmers. In addition, most of the formal work in the literature has concentrated on the  
1420 memory as provided by the hardware as opposed to the application programmer through the  
1421 compiler and runtime system. It was believed that a simple statement intuitive to most  
1422 programmers would be most effective. IEEE Std 1003.1-200x defines functions that can be used  
1423 to synchronize access to memory, but it leaves open exactly how one relates those functions to  
1424 the semantics of each function as specified elsewhere in IEEE Std 1003.1-200x.  
1425 IEEE Std 1003.1-200x also does not make a formal specification of the partial ordering in time  
1426 that the functions can impose, as that is implied in the description of the semantics of each  
1427 function. It simply states that the programmer has to ensure that modifications do not occur  
1428 “simultaneously” with other access to a memory location.

#### 1429 **A.4.11 Pathname Resolution**

1430 It is necessary to differentiate between the definition of pathname and the concept of pathname  
1431 resolution with respect to the handling of trailing slashes. By specifying the behavior here, it is  
1432 not possible to provide an implementation that is conforming but extends all interfaces that  
1433 handle pathnames to also handle strings that are not legal pathnames (because they have trailing  
1434 slashes).

1435 Pathnames that end with one or more trailing slash characters must refer to directory paths.  
1436 Previous versions of IEEE Std 1003.1-200x were not specific about the distinction between  
1437 trailing slashes on files and directories, and both were permitted.

1438 Two types of implementation have been prevalent; those that ignored trailing slash characters  
1439 on all pathnames regardless, and those that only permitted them only on existing directories.

1440 IEEE Std 1003.1-200x requires that a pathname with a trailing slash character be treated as if it  
1441 had a trailing " / . " everywhere.

1442 Note that this change does not break any conforming applications; since there were two  
1443 different types of implementation, no application could have portably depended on either  
1444 behavior. This change does however require some implementations to be altered to remain  
1445 compliant. Substantial discussion over a three-year period has shown that the benefits to  
1446 application developers outweighs the disadvantages for some vendors.

1447 On a historical note, some early applications automatically appended a '/' to every path.  
1448 Rather than fix the applications, the system implementation was modified to accept this  
1449 behavior by ignoring any trailing slash.

1450 Each directory has exactly one parent directory which is represented by the name **dot-dot** in the  
1451 first directory. No other directory, regardless of linkages established by symbolic links, is  
1452 considered the parent directory by IEEE Std 1003.1-200x.

1453 There are two general categories of interfaces involving pathname resolution: those that follow  
1454 the symbolic link, and those that do not. There are several exceptions to this rule; for example,  
1455 *open(path,O\_CREAT|O\_EXCL)* will fail when *path* names a symbolic link. However, in all other  
1456 situations, the *open()* function will follow the link.

1457 What the filename **dot-dot** refers to relative to the root directory is implementation-defined. In  
1458 Version 7 it refers to the root directory itself; this is the behavior mentioned in  
1459 IEEE Std 1003.1-200x. In some networked systems the construction *././hostname/* is used to refer  
1460 to the root directory of another host, and POSIX.1 permits this behavior.

1461 Other networked systems use the construct *//hostname* for the same purpose; that is, a double  
1462 initial slash is used. There is a potential problem with existing applications that create full  
1463 pathnames by taking a trunk and a relative pathname and making them into a single string  
1464 separated by '/', because they can accidentally create networked pathnames when the trunk is  
1465 '/'. This practice is not prohibited because such applications can be made to conform by  
1466 simply changing to use "//" as a separator instead of '/':

- 1467 • If the trunk is '/', the full pathname will begin with "///" (the initial '/' and the  
1468 separator '//'). This is the same as '/', which is what is desired. (This is the general case  
1469 of making a relative pathname into an absolute one by prefixing with "///" instead of '/'.)
- 1470 • If the trunk is "/A", the result is "/A//..."; since non-leading sequences of two or more  
1471 slashes are treated as a single slash, this is equivalent to the desired "/A/...".
- 1472 • If the trunk is "//A", the implementation-defined semantics will apply. (The multiple slash  
1473 rule would apply.)

1474 Application developers should avoid generating pathnames that start with "///".  
1475 Implementations are strongly encouraged to avoid using this special interpretation since a  
1476 number of applications currently do not follow this practice and may inadvertently generate  
1477 "///...".

1478 The term *root directory* is only defined in POSIX.1 relative to the process. In some  
1479 implementations, there may be no absolute root directory. The initialization of the root directory  
1480 of a process is implementation-defined.

#### 1481 **A.4.12 Process ID Reuse**

1482 There is no additional rationale provided for this section.

1483 **A.4.13 Scheduling Policy**

1484 There is no additional rationale provided for this section.

1485 **A.4.14 Seconds Since the Epoch**

1486 Coordinated Universal Time (UTC) includes leap seconds. However, in POSIX time (seconds  
1487 since the Epoch), leap seconds are ignored (not applied) to provide an easy and compatible  
1488 method of computing time differences. Broken-down POSIX time is therefore not necessarily  
1489 UTC, despite its appearance.

1490 As of September 2000, 24 leap seconds had been added to UTC since the Epoch, 1 January, 1970.  
1491 Historically, one leap second is added every 15 months on average, so this offset can be expected  
1492 to grow steadily with time.

1493 Most systems' notion of "time" is that of a continuously increasing value, so this value should  
1494 increase even during leap seconds. However, not only do most systems not keep track of leap  
1495 seconds, but most systems are probably not synchronized to any standard time reference.  
1496 Therefore, it is inappropriate to require that a time represented as seconds since the Epoch  
1497 precisely represent the number of seconds between the referenced time and the Epoch.

1498 It is sufficient to require that applications be allowed to treat this time as if it represented the  
1499 number of seconds between the referenced time and the Epoch. It is the responsibility of the  
1500 vendor of the system, and the administrator of the system, to ensure that this value represents  
1501 the number of seconds between the referenced time and the Epoch as closely as necessary for the  
1502 application being run on that system.

1503 It is important that the interpretation of time names and *seconds since the Epoch* values be  
1504 consistent across conforming systems; that is, it is important that all conforming systems  
1505 interpret "536 457 599 seconds since the Epoch" as 59 seconds, 59 minutes, 23 hours 31 December  
1506 1986, regardless of the accuracy of the system's idea of the current time. The expression is given  
1507 to assure a consistent interpretation, not to attempt to specify the calendar. The relationship  
1508 between *tm\_yday* and the day of week, day of month, and month is presumed to be specified  
1509 elsewhere and is not given in POSIX.1.

1510 Consistent interpretation of *seconds since the Epoch* can be critical to certain types of distributed  
1511 applications that rely on such timestamps to synchronize events. The accrual of leap seconds in  
1512 a time standard is not predictable. The number of leap seconds since the Epoch will likely  
1513 increase. POSIX.1 is more concerned about the synchronization of time between applications of  
1514 astronomically short duration.

1515 Note that *tm\_yday* is zero-based, not one-based, so the day number in the example above is 364.  
1516 Note also that the division is an integer division (discarding remainder) as in the C language.

1517 Note also that the meaning of *gmtime()*, *localtime()*, and *mktime()* is specified in terms of this  
1518 expression. However, the ISO C standard computes *tm\_yday* from *tm\_mday*, *tm\_mon*, and  
1519 *tm\_year* in *mktime()*. Because it is stated as a (bidirectional) relationship, not a function, and  
1520 because the conversion between month-day-year and day-of-year dates is presumed well known  
1521 and is also a relationship, this is not a problem.

1522 Implementations that implement **time\_t** as a signed 32-bit integer will overflow in 2 038. The  
1523 data size for **time\_t** is as per the ISO C standard definition, which is implementation-defined.

1524 See also **Epoch** (on page 3306).

1525 The topic of whether seconds since the Epoch should account for leap seconds has been debated  
1526 on a number of occasions, and each time consensus was reached (with acknowledged dissent  
1527 each time) that the majority of users are best served by treating all days identically. (That is, the

1528 majority of applications were judged to assume a single length—as measured in seconds since  
 1529 the Epoch—for all days. Thus, leap seconds are not applied to seconds since the Epoch.) Those  
 1530 applications which do care about leap seconds can determine how to handle them in whatever  
 1531 way those applications feel is best. This was particularly emphasized because there was  
 1532 disagreement about what the best way of handling leap seconds might be. It is a practical  
 1533 impossibility to mandate that a conforming implementation must have a fixed relationship to  
 1534 any particular official clock (consider isolated systems, or systems performing “reruns” by  
 1535 setting the clock to some arbitrary time).

1536 Note that as a practical consequence of this, the length of a second as measured by some external  
 1537 standard is not specified. This unspecified second is nominally equal to an International System  
 1538 (SI) second in duration. Applications must be matched to a system that provides the particular  
 1539 handling of external time in the way required by the application.

#### 1540 **A.4.15 Semaphore**

1541 There is no additional rationale provided for this section.

#### 1542 **A.4.16 Thread-Safety**

1543 Where the interface of a function required by IEEE Std 1003.1-200x precludes thread-safety, an  
 1544 alternate form that shall be thread-safe is provided. The names of these thread-safe forms are the  
 1545 same as the non-thread-safe forms with the addition of the suffix “\_r”. The suffix “\_r” is  
 1546 historical, where the ‘r’ stood for “reentrant”.

1547 In some cases, thread-safety is provided by restricting the arguments to an existing function.

1548 See also Section B.2.9.1 (on page 3452).

#### 1549 **A.4.17 Tracing**

1550 Refer to Section B.2.11 (on page 3468).

#### 1551 **A.4.18 Treatment of Error Conditions for Mathematical Functions**

1552 There is no additional rationale provided for this section. |

#### 1553 **A.4.19 Treatment of NaN Arguments for Mathematical Functions**

1554 There is no additional rationale provided for this section. |

#### 1555 **A.4.20 Utility**

1556 There is no additional rationale provided for this section.

#### 1557 **A.4.21 Variable Assignment**

1558 There is no additional rationale provided for this section.

## 1559 A.5 File Format Notation

1560 The notation for spaces allows some flexibility for application output. Note that an empty  
 1561 character position in *format* represents one or more <blank>s on the output (not *white space*,  
 1562 which can include <newline>s). Therefore, another utility that reads that output as its input  
 1563 must be prepared to parse the data using *scanf()*, *awk*, and so on. The 'Δ' character is used when  
 1564 exactly one <space> is output.

1565 The treatment of integers and spaces is different from the *printf()* function in that they can be  
 1566 surrounded with <blank>s. This was done so that, given a format such as:

```
1567     "%d\n", <foo>
```

1568 the implementation could use a *printf()* call such as:

```
1569     printf("%6d\n", foo);
```

1570 and still conform. This notation is thus somewhat like *scanf()* in addition to *printf()*.

1571 The *printf()* function was chosen as a model because most of the standard developers were  
 1572 familiar with it. One difference from the C function *printf()* is that the *l* and *h* conversion  
 1573 specifier characters are not used. As expressed by the Shell and Utilities volume of  
 1574 IEEE Std 1003.1-200x, there is no differentiation between decimal values for type **int**, type **long**,  
 1575 or type **short**. The conversion specifications *%d* or *%i* should be interpreted as an arbitrary  
 1576 length sequence of digits. Also, no distinction is made between single precision and double  
 1577 precision numbers (**float** or **double** in C). These are simply referred to as floating-point numbers.

1578 Many of the output descriptions in the Shell and Utilities volume of IEEE Std 1003.1-200x use the  
 1579 term *line*, such as:

```
1580     "%s", <input line>
```

1581 Since the definition of *line* includes the trailing <newline> already, there is no need to include a  
 1582 '\n' in the format; a double <newline> would otherwise result.

## 1583 A.6 Character Set

### 1584 A.6.1 Portable Character Set

1585 The portable character set is listed in full so there is no dependency on the ISO/IEC 646:1991  
 1586 standard (or historically ASCII) encoded character set, although the set is identical to the  
 1587 characters defined in the International Reference version of the ISO/IEC 646:1991 standard.

1588 IEEE Std 1003.1-200x poses no requirement that multiple character sets or codesets be  
 1589 supported, leaving this as a marketing differentiation for implementors. Although multiple  
 1590 charmap files are supported, it is the responsibility of the implementation to provide the file(s);  
 1591 if only one is provided, only that one will be accessible using the *localedef -f* option.

1592 The statement about invariance in codesets for the portable character set is worded to avoid  
 1593 precluding implementations where multiple incompatible codesets are available (for instance,  
 1594 ASCII and EBCDIC). The standard utilities cannot be expected to produce predictable results if  
 1595 they access portable characters that vary on the same implementation.

1596 Not all character sets need include the portable character set, but each locale must include it. For  
 1597 example, a Japanese-based locale might be supported by a mixture of character sets: JIS X 0201  
 1598 Roman (a Japanese version of the ISO/IEC 646:1991 standard), JIS X 0208, and JIS X 0201  
 1599 Katakana. Not all of these character sets include the portable characters, but at least one does  
 1600 (JIS X 0201 Roman).

1601 **A.6.2 Character Encoding**

1602 Encoding mechanisms based on single shifts, such as the EUC encoding used in some Asian and  
 1603 other countries, can be supported via the current charmap mechanism. With single-shift  
 1604 encoding, each character is preceded by a shift code (SS2 or SS3). A complete EUC code,  
 1605 consisting of the portable character set (G0) and up to three additional character sets (G1, G2,  
 1606 G3), can be described using the current charmap mechanism; the encoding for each character in  
 1607 additional character sets G2 and G3 must then include their single-shift code. Other mechanisms  
 1608 to support locales based on encoding mechanisms such as locking shift are not addressed by this  
 1609 volume of IEEE Std 1003.1-200x.

1610 **A.6.3 C Language Wide-Character Codes**

1611 There is no additional rationale provided for this section.

1612 **A.6.4 Character Set Description File**

1613 IEEE PASC Interpretation 1003.2 #196 is applied, removing three lines of text dealing with |  
 1614 ranges of symbolic names using position constant values which had been erroneously included |  
 1615 in the final 1003.2b draft. |

1616 **A.6.4.1 State-Dependent Character Encodings**

1617 A requirement was considered that would force utilities to eliminate any redundant locking  
 1618 shifts, but this was left as a quality of implementation issue.

1619 This change satisfies the following requirement from the ISO POSIX-2:1993 standard, Annex  
 1620 H.1:

1621 *The support of state-dependent (shift encoding) character sets should be addressed fully. See*  
 1622 *descriptions of these in the Base Definitions volume of IEEE Std 1003.1-200x, Section 6.2, Character*  
 1623 *Encoding. If such character encodings are supported, it is expected that this will impact the Base*  
 1624 *Definitions volume of IEEE Std 1003.1-200x, Section 6.2, Character Encoding, the Base Definitions*  
 1625 *volume of IEEE Std 1003.1-200x, Chapter 7, Locale, the Base Definitions volume of*  
 1626 *IEEE Std 1003.1-200x, Chapter 9, Regular Expressions , and the comm, cut, diff, grep, head, join,*  
 1627 *paste, and tail utilities.*

1628 The character set description file provides:

- 1629 • The capability to describe character set attributes (such as collation order or character  
 1630 classes) independent of character set encoding, and using only the characters in the portable  
 1631 character set. This makes it possible to create generic *localedef* source files for all codesets that  
 1632 share the portable character set (such as the ISO 8859 family or IBM Extended ASCII).
- 1633 • Standardized symbolic names for all characters in the portable character set, making it  
 1634 possible to refer to any such character regardless of encoding.

1635 Implementations are free to choose their own symbolic names, as long as the names identified  
 1636 by this volume of IEEE Std 1003.1-200x are also defined; this provides support for already  
 1637 existing “character names”.

1638 The names selected for the members of the portable character set follow the  
 1639 ISO/IEC 8859-1:1998 standard and the ISO/IEC 10646-1:2000 standard. However, several  
 1640 commonly used UNIX system names occur as synonyms in the list:

- 1641 • The historical UNIX system names are used for control characters.

- 1642 • The word “slash” is given in addition to “solidus”.
  - 1643 • The word “backslash” is given in addition to “reverse-solidus”.
  - 1644 • The word “hyphen” is given in addition to “hyphen-minus”.
  - 1645 • The word “period” is given in addition to “full-stop”.
  - 1646 • For digits, the word “digit” is eliminated.
  - 1647 • For letters, the words “Latin Capital Letter” and “Latin Small Letter” are eliminated.
  - 1648 • The words “left brace” and “right brace” are given in addition to “left-curly-bracket” and
  - 1649 “right-curly-bracket”.
  - 1650 • The names of the digits are preferred over the numbers to avoid possible confusion between
  - 1651 ‘0’ and ‘o’, and between ‘1’ and ‘l’ (one and the letter ell).
- 1652 The names for the control characters in the Base Definitions volume of IEEE Std 1003.1-200x,
- 1653 Chapter 6, Character Set were taken from the ISO/IEC 4873: 1991 standard.
- 1654 The charmap file was introduced to resolve problems with the portability of, especially, *localedef*
- 1655 sources. IEEE Std 1003.1-200x assumes that the portable character set is constant across all
- 1656 locales, but does not prohibit implementations from supporting two incompatible codings, such
- 1657 as both ASCII and EBCDIC. Such dual-support implementations should have all charmaps and
- 1658 *localedef* sources encoded using one portable character set, in effect cross-compiling for the other
- 1659 environment. Naturally, charmaps (and *localedef* sources) are only portable without
- 1660 transformation between systems using the same encodings for the portable character set. They
- 1661 can, however, be transformed between two sets using only a subset of the actual characters (the
- 1662 portable character set). However, the particular coded character set used for an application or an
- 1663 implementation does not necessarily imply different characteristics or collation; on the contrary,
- 1664 these attributes should in many cases be identical, regardless of codeset. The charmap provides
- 1665 the capability to define a common locale definition for multiple codesets (the same *localedef*
- 1666 source can be used for codesets with different extended characters; the ability in the charmap to
- 1667 define empty names allows for characters missing in certain codesets).
- 1668 The `<escape_char>` declaration was added at the request of the international community to ease
- 1669 the creation of portable charmap files on terminals not implementing the default backslash
- 1670 escape. The `<comment_char>` declaration was added at the request of the international
- 1671 community to eliminate the potential confusion between the number sign and the pound sign.
- 1672 The octal number notation with no leading zero required was selected to match those of *awk* and
- 1673 *tr* and is consistent with that used by *localedef*. To avoid confusion between an octal constant
- 1674 and the back-references used in *localedef* source, the octal, hexadecimal, and decimal constants
- 1675 shall contain at least two digits. As single-digit constants are relatively rare, this should not
- 1676 impose any significant hardship. Provision is made for more digits to account for systems in
- 1677 which the byte size is larger than 8 bits. For example, a Unicode (ISO/IEC 10646-1:2000
- 1678 standard) system that has defined 16-bit bytes may require six octal, four hexadecimal, and five
- 1679 decimal digits.
- 1680 The decimal notation is supported because some newer international standards define character
- 1681 values in decimal, rather than in the old column/row notation.
- 1682 The charmap identifies the coded character sets supported by an implementation. At least one
- 1683 charmap shall be provided, but no implementation is required to provide more than one.
- 1684 Likewise, implementations can allow users to generate new charmaps (for instance, for a new
- 1685 version of the ISO 8859 family of coded character sets), but does not have to do so. If users are
- 1686 allowed to create new charmaps, the system documentation describes the rules that apply (for
- 1687 instance, “only coded character sets that are supersets of the ISO/IEC 646: 1991 standard IRV, no

1688 multi-byte characters’).

1689 This addition of the **WIDTH** specification satisfies the following requirement from the  
1690 ISO POSIX-2: 1993 standard, Annex H.1:

1691 (9) *The definition of column position relies on the implementation’s knowledge of the integral width*  
1692 *of the characters. The charmap or LC\_CTYPE locale definitions should be enhanced to allow*  
1693 *application specification of these widths.*

1694 The character “width” information was first considered for inclusion under *LC\_CTYPE* but was  
1695 moved because it is more closely associated with the information in the *charmap* than  
1696 information in the locale source (cultural conventions information). Concerns were raised that  
1697 formalizing this type of information is moving the locale source definition from the codeset-  
1698 independent entity that it was designed to be to a repository of codeset-specific information. A  
1699 similar issue occurred with the `<code_set_name>`, `<mb_cur_max>`, and `<mb_cur_min>`  
1700 information, which was resolved to reside in the *charmap* definition.

1701 The width definition was added to the IEEE P1003.2b draft standard with the intent that the  
1702 *wcswidth()* and/or *wcwidth()* functions (currently specified in the System Interfaces volume of  
1703 IEEE Std 1003.1-200x) be the mechanism to retrieve the character width information.

## 1704 **A.7 Locale**

### 1705 **A.7.1 General**

1706 The description of locales is based on work performed in the UniForum Technical Committee  
1707 Subcommittee on Internationalization. Wherever appropriate, keywords are taken from the  
1708 ISO C standard or the X/Open Portability Guide.

1709 The value used to specify a locale with environment variables is the name specified as the *name*  
1710 operand to the *localedef* utility when the locale was created. This provides a verifiable method to  
1711 create and invoke a locale.

1712 The “object” definitions need not be portable, as long as “source” definitions are. Strictly  
1713 speaking, source definitions are portable only between implementations using the same  
1714 character set(s). Such source definitions, if they use symbolic names only, easily can be ported  
1715 between systems using different codesets, as long as the characters in the portable character set  
1716 (see the Base Definitions volume of IEEE Std 1003.1-200x, Section 6.1, Portable Character Set )  
1717 have common values between the codesets; this is frequently the case in historical  
1718 implementations. Of source, this requires that the symbolic names used for characters outside  
1719 the portable character set be identical between character sets. The definition of symbolic names  
1720 for characters is outside the scope of IEEE Std 1003.1-200x, but is certainly within the scope of  
1721 other standards organizations.

1722 Applications can select the desired locale by invoking the *setlocale()* function (or equivalent)  
1723 with the appropriate value. If the function is invoked with an empty string, the value of the  
1724 corresponding environment variable is used. If the environment variable is not set or is set to the  
1725 empty string, the implementation sets the appropriate environment as defined in the Base  
1726 Definitions volume of IEEE Std 1003.1-200x, Chapter 8, Environment Variables.

**1727 A.7.2 POSIX Locale**

1728 The POSIX locale is equal to the C locale. To avoid being classified as a C-language function, the  
1729 name has been changed to the POSIX locale; the environment variable value can be either  
1730 "POSIX" or, for historical reasons, "C".

1731 The POSIX definitions mirror the historical UNIX system behavior.

1732 The use of symbolic names for characters in the tables does not imply that the POSIX locale must  
1733 be described using symbolic character names, but merely that it may be advantageous to do so.

**1734 A.7.3 Locale Definition**

1735 The decision to separate the file format from the *localedef* utility description was only partially  
1736 editorial. Implementations may provide other interfaces than *localedef*. Requirements on “the  
1737 utility”, mostly concerning error messages, are described in this way because they are meant to  
1738 affect the other interfaces implementations may provide as well as *localedef*.

1739 The text about POSIX2\_LOCALEDEF does not mean that internationalization is optional; only  
1740 that the functionality of the *localedef* utility is. REs, for instance, must still be able to recognize,  
1741 for example, character class expressions such as "[[:alpha:]]". A possible analogy is with  
1742 an applications development environment; while all conforming implementations must be  
1743 capable of executing applications, not all need to have the development environment installed.  
1744 The assumption is that the capability to modify the behavior of utilities (and applications) via  
1745 locale settings must be supported. If the *localedef* utility is not present, then the only choice is to  
1746 select an existing (presumably implementation-documented) locale. An implementation could,  
1747 for example, choose to support only the POSIX locale, which would in effect limit the amount of  
1748 changes from historical implementations quite drastically. The *localedef* utility is still required,  
1749 but would always terminate with an exit code indicating that no locale could be created.  
1750 Supported locales must be documented using the syntax defined in this chapter. (This ensures  
1751 that users can accurately determine what capabilities are provided. If the implementation  
1752 decides to provide additional capabilities to the ones in this chapter, that is already provided  
1753 for.)

1754 If the option is present (that is, locales can be created), then the *localedef* utility must be capable  
1755 of creating locales based on the syntax and rules defined in this chapter. This does not mean that  
1756 the implementation cannot also provide alternate means for creating locales.

1757 The octal, decimal, and hexadecimal notations are the same employed by the charmap facility  
1758 (see the Base Definitions volume of IEEE Std 1003.1-200x, Section 6.4, Character Set Description  
1759 File). To avoid confusion between an octal constant and a back-reference, the octal, hexadecimal,  
1760 and decimal constants must contain at least two digits. As single-digit constants are relatively  
1761 rare, this should not impose any significant hardship. Provision is made for more digits to  
1762 account for systems in which the byte size is larger than 8 bits. For example, a Unicode (see the  
1763 ISO/IEC 10646-1:2000 standard) system that has defined 16-bit bytes may require six octal, four  
1764 hexadecimal, and five decimal digits. As with the charmap file, multi-byte characters are  
1765 described in the locale definition file using “big-endian” notation for reasons of portability.  
1766 There is no requirement that the internal representation in the computer memory be in this same  
1767 order.

1768 One of the guidelines used for the development of this volume of IEEE Std 1003.1-200x is that  
1769 characters outside the invariant part of the ISO/IEC 646:1991 standard should not be used in  
1770 portable specifications. The backslash character is not in the invariant part; the number sign is,  
1771 but with multiple representations: as a number sign, and as a pound sign. As far as general  
1772 usage of these symbols, they are covered by the “grandfather clause”, but for newly defined  
1773 interfaces, the WG15 POSIX working group has requested that POSIX provide alternate

1774 representations. Consequently, while the default escape character remains the backslash and the  
1775 default comment character is the number sign, implementations are required to recognize  
1776 alternative representations, identified in the applicable source file via the `<escape_char>` and  
1777 `<comment_char>` keywords.

#### 1778 A.7.3.1 *LC\_CTYPE*

1779 The *LC\_CTYPE* category is primarily used to define the encoding-independent aspects of a  
1780 character set, such as character classification. In addition, certain encoding-dependent  
1781 characteristics are also defined for an application via the *LC\_CTYPE* category.  
1782 IEEE Std 1003.1-200x does not mandate that the encoding used in the locale is the same as the  
1783 one used by the application because an implementation may decide that it is advantageous to  
1784 define locales in a system-wide encoding rather than having multiple, logically identical locales  
1785 in different encodings, and to convert from the application encoding to the system-wide  
1786 encoding on usage. Other implementations could require encoding-dependent locales.

1787 In either case, the *LC\_CTYPE* attributes that are directly dependent on the encoding, such as  
1788 `<mb_cur_max>` and the display width of characters, are not user-specifiable in a locale source  
1789 and are consequently not defined as keywords.

1790 Implementations may define additional keywords or extend the *LC\_CTYPE* mechanism to allow  
1791 application-defined keywords.

1792 The text “The ellipsis specification shall only be valid within a single encoded character set” is  
1793 present because it is possible to have a locale supported by multiple character encodings, as  
1794 explained in the rationale for the Base Definitions volume of IEEE Std 1003.1-200x, Section 6.1,  
1795 Portable Character Set. An example given there is of a possible Japanese-based locale supported  
1796 by a mixture of the character sets JIS X 0201 Roman, JIS X 0208, and JIS X 0201 Katakana.  
1797 Attempting to express a range of characters across these sets is not logical and the  
1798 implementation is free to reject such attempts.

1799 As the *LC\_CTYPE* character classes are based on the ISO C standard character class definition,  
1800 the category does not support multi-character elements. For instance, the German character  
1801 `<sharp-s>` is traditionally classified as a lowercase letter. There is no corresponding uppercase  
1802 letter; in proper capitalization of German text, the `<sharp-s>` will be replaced by "SS"; that is, by  
1803 two characters. This kind of conversion is outside the scope of the **toupper** and **tolower**  
1804 keywords.

1805 Where IEEE Std 1003.1-200x specifies that only certain characters can be specified, as for the  
1806 keywords **digit** and **xdigit**, the specified characters shall be from the portable character set, as  
1807 shown. As an example, only the Arabic digits 0 through 9 are acceptable as digits.

1808 The character classes **digit**, **xdigit**, **lower**, **upper**, and **space** have a set of automatically included  
1809 characters. These only need to be specified if the character values (that is, encoding) differs from  
1810 the implementation default values. It is not possible to define a locale without these  
1811 automatically included characters unless some implementation extension is used to prevent  
1812 their inclusion. Such a definition would not be a proper superset of the C locale, and thus, it  
1813 might not be possible for the standard utilities to be implemented as programs conforming to  
1814 the ISO C standard.

1815 The definition of character class **digit** requires that only ten characters—the ones defining  
1816 digits—can be specified; alternate digits (for example, Hindi or Kanji) cannot be specified here.  
1817 However, the encoding may vary if an implementation supports more than one encoding.

1818 The definition of character class **xdigit** requires that the characters included in character class  
1819 **digit** are included here also and allows for different symbols for the hexadecimal digits 10  
1820 through 15.

1821 The inclusion of the **charclass** keyword satisfies the following requirement from the  
1822 ISO POSIX-2: 1993 standard, Annex H.1:

1823 (3) *The LC\_CTYPE (2.5.2.1) locale definition should be enhanced to allow user-specified additional*  
1824 *character classes, similar in concept to the ISO C standard Multibyte Support Extension (MSE)*  
1825 *is\_wctype() function.*

1826 This keyword was previously included in The Open Group specifications and is now mandated  
1827 in the Shell and Utilities volume of IEEE Std 1003.1-200x.

1828 The symbolic constant {CHARCLASS\_NAME\_MAX} was also adopted from The Open Group  
1829 specifications. Application portability is enhanced by the use of symbolic constants.

### 1830 A.7.3.2 LC\_COLLATE

1831 The rules governing collation depend to some extent on the use. At least five different levels of  
1832 increasingly complex collation rules can be distinguished:

- 1833 1. *Byte/machine code order*: This is the historical collation order in the UNIX system and many  
1834 proprietary operating systems. Collation is here performed character by character, without  
1835 any regard to context. The primary virtue is that it usually is quite fast and also  
1836 completely deterministic; it works well when the native machine collation sequence  
1837 matches the user expectations.
- 1838 2. *Character order*: On this level, collation is also performed character by character, without  
1839 regard to context. The order between characters is, however, not determined by the code  
1840 values, but on the expectations by the user of the “correct” order between characters. In  
1841 addition, such a (simple) collation order can specify that certain characters collate equally  
1842 (for example, uppercase and lowercase letters).
- 1843 3. *String ordering*: On this level, entire strings are compared based on relatively  
1844 straightforward rules. Several “passes” may be required to determine the order between  
1845 two strings. Characters may be ignored in some passes, but not in others; the strings may  
1846 be compared in different directions; and simple string substitutions may be performed  
1847 before strings are compared. This level is best described as “dictionary” ordering; it is  
1848 based on the spelling, not the pronunciation, or meaning, of the words.
- 1849 4. *Text search ordering*: This is a further refinement of the previous level, best described as  
1850 “telephone book ordering”; some common homonyms (words spelled differently but with  
1851 the same pronunciation) are collated together; numbers are collated as if they were spelled  
1852 out, and so on.
- 1853 5. *Semantic-level ordering*: Words and strings are collated based on their meaning; entire words  
1854 (such as “the”) are eliminated; the ordering is not deterministic. This usually requires  
1855 special software and is highly dependent on the intended use.

1856 While the historical collation order formally is at level 1, for the English language it corresponds  
1857 roughly to elements at level 2. The user expects to see the output from the *ls* utility sorted very  
1858 much as it would be in a dictionary. While telephone book ordering would be an optimal goal  
1859 for standard collation, this was ruled out as the order would be language-dependent.  
1860 Furthermore, a requirement was that the order must be determined solely from the text string  
1861 and the collation rules; no external information (for example, “pronunciation dictionaries”)   
1862 could be required.

1863 As a result, the goal for the collation support is at level 3. This also matches the requirements for  
1864 the Canadian collation order, as well as other, known collation requirements for alphabetic  
1865 scripts. It specifically rules out collation based on pronunciation rules or based on semantic

1866 analysis of the text.

1867 The syntax for the *LC\_COLLATE* category source meets the requirements for level 3 and has  
1868 been verified to produce the correct result with examples based on French, Canadian, and  
1869 Danish collation order. Because it supports multi-character collating elements, it is also capable  
1870 of supporting collation in codesets where a character is expressed using non-spacing characters  
1871 followed by the base character (such as the ISO/IEC 6937: 1994 standard).

1872 The directives that can be specified in an operand to the **order\_start** keyword are based on the  
1873 requirements specified in several proposed standards and in customary use. The following is a  
1874 rephrasing of rules defined for “lexical ordering in English and French” by the Canadian  
1875 Standards Association (the text in square brackets is rephrased):

- 1876 • Once special characters [punctuation] have been removed from original strings, the ordering  
1877 is determined by scanning forwards (left to right) [disregarding case and diacriticals].
- 1878 • In case of equivalence, special characters are once again removed from original strings and  
1879 the ordering is determined by scanning backwards (starting from the rightmost character of  
1880 the string and back), character by character [disregarding case but considering diacriticals].
- 1881 • In case of repeated equivalence, special characters are removed again from original strings  
1882 and the ordering is determined by scanning forwards, character by character [considering  
1883 both case and diacriticals].
- 1884 • If there is still an ordering equivalence after the first three rules have been applied, then only  
1885 special characters and the position they occupy in the string are considered to determine  
1886 ordering. The string that has a special character in the lowest position comes first. If two  
1887 strings have a special character in the same position, the character [with the lowest collation  
1888 value] comes first. In case of equality, the other special characters are considered until there  
1889 is a difference or until all special characters have been exhausted.

1890 It is estimated that this part of IEEE Std 1003.1-200x covers the requirements for all European  
1891 languages, and no particular problems are anticipated with Slavic or Middle East character sets.

1892 The Far East (particularly Japanese/Chinese) collations are often based on contextual  
1893 information and pronunciation rules (the same ideogram can have different meanings and  
1894 different pronunciations). Such collation, in general, falls outside the desired goal of  
1895 IEEE Std 1003.1-200x. There are, however, several other collation rules (stroke/radical or “most  
1896 common pronunciation”) that can be supported with the mechanism described here.

1897 The character order is defined by the order in which characters and elements are specified  
1898 between the **order\_start** and **order\_end** keywords. Weights assigned to the characters and  
1899 elements define the collation sequence; in the absence of weights, the character order is also the  
1900 collation sequence.

1901 The **position** keyword provides the capability to consider, in a compare, the relative position of  
1902 characters not subject to **IGNORE**. As an example, consider the two strings "o-ring" and  
1903 "or-ing". Assuming the hyphen is subject to **IGNORE** on the first pass, the two strings  
1904 compare equal, and the position of the hyphen is immaterial. On second pass, all characters  
1905 except the hyphen are subject to **IGNORE**, and in the normal case the two strings would again  
1906 compare equal. By taking position into account, the first collates before the second.

1907 A.7.3.3 LC\_MONETARY

1908 The currency symbol does not appear in LC\_MONETARY because it is not defined in the C locale  
 1909 of the ISO C standard.

1910 The ISO C standard limits the size of decimal points and thousands delimiters to single-byte  
 1911 values. In locales based on multi-byte coded character sets, this cannot be enforced;  
 1912 IEEE Std 1003.1-200x does not prohibit such characters, but makes the behavior unspecified (in  
 1913 the text “In contexts where other standards ...”).

1914 The grouping specification is based on, but not identical to, the ISO C standard. The -1 signals  
 1915 that no further grouping shall be performed; the equivalent of {CHAR\_MAX} in the ISO C  
 1916 standard.

1917 The text “the value is not available in the locale” is taken from the ISO C standard and is used  
 1918 instead of the “unspecified” text in early proposals. There is no implication that omitting these  
 1919 keywords or assigning them values of " " or -1 produces unspecified results; such omissions or  
 1920 assignments eliminate the effects described for the keyword or produce zero-length strings, as  
 1921 appropriate.

1922 The locale definition is an extension of the ISO C standard localeconv() specification. In  
 1923 particular, rules on how currency\_symbol is treated are extended to also cover int\_curr\_symbol,  
 1924 and p\_sep\_by\_space and n\_sep\_by\_space have been augmented with the value 2, which places  
 1925 a <space> between the sign and the symbol (if they are adjacent; otherwise, it should be treated  
 1926 as a 0). The following table shows the result of various combinations:

		p_sep_by_space		
		2	1	0
p_cs_precedes = 1	p_sign_posn = 0	(\$1.25)	(\$ 1.25)	(\$1.25)
	p_sign_posn = 1	+ \$1.25	+\$ 1.25	+\$1.25
	p_sign_posn = 2	\$1.25 +	\$ 1.25+	\$1.25+
	p_sign_posn = 3	+ \$1.25	+\$ 1.25	+\$1.25
	p_sign_posn = 4	\$ +1.25	\$+ 1.25	+\$1.25
p_cs_precedes = 0	p_sign_posn = 0	(1.25 \$)	(1.25 \$)	(1.25\$)
	p_sign_posn = 1	+1.25 \$	+1.25 \$	+1.25\$
	p_sign_posn = 2	1.25\$ +	1.25 \$+	1.25\$+
	p_sign_posn = 3	1.25+ \$	1.25 +\$	1.25+\$
	p_sign_posn = 4	1.25\$ +	1.25 \$+	1.25\$+

1939 The following is an example of the interpretation of the mon\_grouping keyword. Assuming that  
 1940 the value to be formatted is 123456789 and the mon\_thousands\_sep is ' ', then the following  
 1941 table shows the result. The third column shows the equivalent string in the ISO C standard that  
 1942 would be used by the localeconv() function to accommodate this grouping.

mon_grouping	Formatted Value	ISO C String
3;-1	123456'789	"\3\177"
3	123'456'789	"\3"
3;2;-1	1234'56'789	"\3\2\177"
3;2	12'34'56'789	"\3\2"
-1	123456789	"\177"

1949 In these examples, the octal value of {CHAR\_MAX} is 177.

1950 A.7.3.4 *LC\_NUMERIC*

1951 See the rationale for *LC\_MONETARY* for a description of the behavior of grouping.

1952 A.7.3.5 *LC\_TIME*

1953 Although certain of the conversion specifications in the POSIX locale (such as the name of the  
1954 month) are shown with initial capital letters, this need not be the case in other locales. Programs  
1955 using these conversion specifications may need to adjust the capitalization if the output is going  
1956 to be used at the beginning of a sentence.

1957 The *LC\_TIME* descriptions of **abday**, **day**, **mon**, and **abmon** imply a Gregorian style calendar (7-  
1958 day weeks, 12-month years, leap years, and so on). Formatting time strings for other types of  
1959 calendars is outside the scope of IEEE Std 1003.1-200x.

1960 While the ISO 8601:2000 standard numbers the weekdays starting with Monday, historical  
1961 practice is to use the Sunday as the first day. Rather than change the order and introduce  
1962 potential confusion, the days must be specified beginning with Sunday; previous references to  
1963 “first day” have been removed. Note also that the Shell and Utilities volume of  
1964 IEEE Std 1003.1-200x *date* utility supports numbering compliant with the ISO 8601:2000  
1965 standard.

1966 As specified under *date* in the Shell and Utilities volume of IEEE Std 1003.1-200x and *strftime()* in  
1967 the System Interfaces volume of IEEE Std 1003.1-200x, the conversion specifications  
1968 corresponding to the optional keywords consist of a modifier followed by a traditional  
1969 conversion specification (for instance, %Ex). If the optional keywords are not supported by the  
1970 implementation or are unspecified for the current locale, these modified conversion  
1971 specifications are treated as the traditional conversion specifications. For example, assume the  
1972 following keywords:

```
1973 alt_digits "0th";"1st";"2nd";"3rd";"4th";"5th";\  
1974 "6th";"7th";"8th";"9th";"10th"  
1975 d_fmt "The %Od day of %B in %Y"
```

1976 On July 4th 1776, the %x conversion specifications would result in "The 4th day of July  
1977 in 1776", while on July 14th 1789 it would result in "The 14 day of July in 1789". It  
1978 can be noted that the above example is for illustrative purposes only; the %O modifier is  
1979 primarily intended to provide for Kanji or Hindi digits in *date* formats.

1980 The following is an example for Japan that supports the current plus last three Emperors and  
1981 reverts to Western style numbering for years prior to the Meiji era. The example also allows for  
1982 the custom of using a special name for the first year of an era instead of using 1. (The examples  
1983 substitute romaji where kanji should be used.)

```
1984 era_d_fmt "%EY%mgatsu%dnichi (%a)"  
1985 era "+:2:1990/01/01:+*:Heisei:%EC%Eynen";\  
1986 "+:1:1989/01/08:1989/12/31:Heisei:%ECgannen";\  
1987 "+:2:1927/01/01:1989/01/07:Shouwa:%EC%Eynen";\  
1988 "+:1:1926/12/25:1926/12/31:Shouwa:%ECgannen";\  
1989 "+:2:1913/01/01:1926/12/24:Taishou:%EC%Eynen";\  
1990 "+:1:1912/07/30:1912/12/31:Taishou:%ECgannen";\  
1991 "+:2:1869/01/01:1912/07/29:Meiji:%EC%Eynen";\  
1992 "+:1:1868/09/08:1868/12/31:Meiji:%ECgannen";\  
1993 "-:1868:1868/09/07:-*:%Ey"
```

1994 Assuming that the current date is September 21, 1991, a request to *date* or *strftime()* would yield  
 1995 the following results:

```
1996      %Ec - Heisei3nen9gatsu21nichi (Sat) 14:39:26
1997      %EC - Heisei
1998      %Ex - Heisei3nen9gatsu21nichi (Sat)
1999      %Ey - 3
2000      %EY - Heisei3nen
```

2001 Example era definitions for the Republic of China:

```
2002      era      "+:2:1913/01/01:+*:ChungHwaMingGuo:%EC%EyNen";\
2003      "+:1:1912/1/1:1912/12/31:ChungHwaMingGuo:%ECYuenNen";\
2004      "+:1:1911/12/31:-*:MingChien:%EC%EyNen"
```

2005 Example definitions for the Christian Era:

```
2006      era      "+:0:0001/01/01:+*:AD:%EC %Ey";\
2007      "+:1:-0001/12/31:-*:BC:%Ey %EC"
```

#### 2008 A.7.3.6 LC\_MESSAGES

2009 The **yesstr** and **nostr** locale keywords and the YESSTR and NOSTR *langinfo* items were formerly  
 2010 used to match user affirmative and negative responses. In IEEE Std 1003.1-200x, the **yesexpr**,  
 2011 **noexpr**, YESEXPR, and NOEXPR extended regular expressions have replaced them. |  
 2012 Applications should use the general locale-based messaging facilities to issue prompting |  
 2013 messages which include sample desired responses. |

### 2014 A.7.4 Locale Definition Grammar

2015 There is no additional rationale provided for this section.

#### 2016 A.7.4.1 Locale Lexical Conventions

2017 There is no additional rationale provided for this section.

#### 2018 A.7.4.2 Locale Grammar

2019 There is no additional rationale provided for this section.

### 2020 A.7.5 Locale Definition Example

2021 The following is an example of a locale definition file that could be used as input to the *localedef*  
 2022 utility. It assumes that the utility is executed with the *-f* option, naming a *charmap* file with (at  
 2023 least) the following content:

```

2024     CHARMAP
2025     <space>          \x20
2026     <dollar>         \x24
2027     <A>             \101
2028     <a>             \141
2029     <A-acute>       \346
2030     <a-acute>       \365
2031     <A-grave>       \300
2032     <a-grave>       \366
2033     <b>             \142
2034     <C>             \103
2035     <c>             \143
2036     <c-cedilla>     \347
2037     <d>             \x64
2038     <H>             \110
2039     <h>             \150
2040     <eszet>         \xb7
2041     <s>             \x73
2042     <z>             \x7a
2043     END CHARMAP

```

2044 It should not be taken as complete or to represent any actual locale, but only to illustrate the  
 2045 syntax.

```

2046     #
2047     LC_CTYPE
2048     lower <a>;<b>;<c>;<c-cedilla>;<d>;...;<z>
2049     upper A;B;C;Ç;...;Z
2050     space \x20;\x09;\x0a;\x0b;\x0c;\x0d
2051     blank \040;\011
2052     toupper (<a>,<A>);(b,B);(c,C);(ç,Ç);(d,D);(z,Z)
2053     END LC_CTYPE
2054     #
2055     LC_COLLATE
2056     #
2057     # The following example of collation is based on
2058     # Canadian standard Z243.4.1-1998, "Canadian Alphanumeric
2059     # Ordering Standard For Character sets of CSA Z234.4 Standard".
2060     # (Other parts of this example locale definition file do not
2061     # purport to relate to Canada, or to any other real culture.)
2062     # The proposed standard defines a 4-weight collation, such that
2063     # in the first pass, characters are compared without regard to
2064     # case or accents; in second pass, backwards compare without
2065     # regard to case; in the third pass, forward compare without
2066     # regard to diacriticals. In the 3 first passes, non-alphabetic
2067     # characters are ignored; in the fourth pass, only special
2068     # characters are considered, such that "The string that has a
2069     # special character in the lowest position comes first. If two
2070     # strings have a special character in the same position, the
2071     # collation value of the special character determines ordering.
2072     #
2073     # Only a subset of the character set is used here; mostly to
2074     # illustrate the set-up.

```

```

2075      #
2076      collating-symbol <NULL>
2077      collating-symbol <LOW_VALUE>
2078      collating-symbol <LOWER-CASE>
2079      collating-symbol <SUBSCRIPT-LOWER>
2080      collating-symbol <SUPERSCRIPT-LOWER>
2081      collating-symbol <UPPER-CASE>
2082      collating-symbol <NO-ACCENT>
2083      collating-symbol <PECULIAR>
2084      collating-symbol <LIGATURE>
2085      collating-symbol <ACUTE>
2086      collating-symbol <GRAVE>
2087      # Further collating-symbols follow.
2088      #
2089      # Properly, the standard does not include any multi-character
2090      # collating elements; the one below is added for completeness.
2091      #
2092      collating_element <ch> from "<c><h>"
2093      collating_element <CH> from "<C><H>"
2094      collating_element <Ch> from "<C><h>"
2095      #
2096      order_start forward;backward;forward;forward,position
2097      #
2098      # Collating symbols are specified first in the sequence to allocate
2099      # basic collation values to them, lower than that of any character.
2100      <NULL>
2101      <LOW_VALUE>
2102      <LOWER-CASE>
2103      <SUBSCRIPT-LOWER>
2104      <SUPERSCRIPT-LOWER>
2105      <UPPER-CASE>
2106      <NO-ACCENT>
2107      <PECULIAR>
2108      <LIGATURE>
2109      <ACUTE>
2110      <GRAVE>
2111      <RING-ABOVE>
2112      <DIAERESIS>
2113      <TILDE>
2114      # Further collating symbols are given a basic collating value here.
2115      #
2116      # Here follow special characters.
2117      <space>          IGNORE;IGNORE;IGNORE;<space>
2118      # Other special characters follow here.
2119      #
2120      # Here follow the regular characters.
2121      <a>              <a>;<NO-ACCENT>;<LOWER-CASE>;IGNORE
2122      <A>              <a>;<NO-ACCENT>;<UPPER-CASE>;IGNORE
2123      <a-acute>        <a>;<ACUTE>;<LOWER-CASE>;IGNORE
2124      <A-acute>        <a>;<ACUTE>;<UPPER-CASE>;IGNORE
2125      <a-grave>        <a>;<GRAVE>;<LOWER-CASE>;IGNORE
2126      <A-grave>        <a>;<GRAVE>;<UPPER-CASE>;IGNORE

```

```

2127 <ae>      "<a><e>" ; "<LIGATURE><LIGATURE>" ; \
2128          "<LOWER-CASE><LOWER-CASE>" ; IGNORE
2129 <AE>      "<a><e>" ; "<LIGATURE><LIGATURE>" ; \
2130          "<UPPER-CASE><UPPER-CASE>" ; IGNORE
2131 <b>        <b> ; <NO-ACCENT> ; <LOWER-CASE> ; IGNORE
2132 <B>        <b> ; <NO-ACCENT> ; <UPPER-CASE> ; IGNORE
2133 <c>        <c> ; <NO-ACCENT> ; <LOWER-CASE> ; IGNORE
2134 <C>        <c> ; <NO-ACCENT> ; <UPPER-CASE> ; IGNORE
2135 <ch>       <ch> ; <NO-ACCENT> ; <LOWER-CASE> ; IGNORE
2136 <Ch>       <ch> ; <NO-ACCENT> ; <PECULIAR> ; IGNORE
2137 <CH>       <ch> ; <NO-ACCENT> ; <UPPER-CASE> ; IGNORE
2138 #
2139 # As an example, the strings "Bach" and "bach" could be encoded (for
2140 # compare purposes) as:
2141 # "Bach"   <b> ; <a> ; <ch> ; <LOW_VALUE> ; <NO-ACCENT> ; <NO-ACCENT> ; \
2142 #          <NO-ACCENT> ; <LOW_VALUE> ; <UPPER-CASE> ; <LOWER-CASE> ; \
2143 #          <LOWER-CASE> ; <NULL>
2144 # "bach"   <b> ; <a> ; <ch> ; <LOW_VALUE> ; <NO-ACCENT> ; <NO-ACCENT> ; \
2145 #          <NO-ACCENT> ; <LOW_VALUE> ; <LOWER-CASE> ; <LOWER-CASE> ; \
2146 #          <LOWER-CASE> ; <NULL>
2147 #
2148 # The two strings are equal in pass 1 and 2, but differ in pass 3.
2149 #
2150 # Further characters follow.
2151 #
2152 UNDEFINED  IGNORE ; IGNORE ; IGNORE ; IGNORE
2153 #
2154 order_end
2155 #
2156 END LC_COLLATE
2157 #
2158 LC_MONETARY
2159 int_curr_symbol      "USD "
2160 currency_symbol     "$"
2161 mon_decimal_point   "."
2162 mon_grouping        3;0
2163 positive_sign       ""
2164 negative_sign       "- "
2165 p_cs_precedes       1
2166 n_sign_posn         0
2167 END LC_MONETARY
2168 #
2169 LC_NUMERIC
2170 copy "US_en.ASCII"
2171 END LC_NUMERIC
2172 #
2173 LC_TIME
2174 abday "Sun" ; "Mon" ; "Tue" ; "Wed" ; "Thu" ; "Fri" ; "Sat"
2175 #
2176 day "Sunday" ; "Monday" ; "Tuesday" ; "Wednesday" ; \
2177    "Thursday" ; "Friday" ; "Saturday"
2178 #

```

```

2179     abmon    "Jan"; "Feb"; "Mar"; "Apr"; "May"; "Jun"; \
2180           "Jul"; "Aug"; "Sep"; "Oct"; "Nov"; "Dec"
2181     #
2182     mon      "January"; "February"; "March"; "April"; \
2183           "May"; "June"; "July"; "August"; "September"; \
2184           "October"; "November"; "December"
2185     #
2186     d_t_fmt  "%a %b %d %T %Z %Y\n"
2187     END LC_TIME
2188     #
2189     LC_MESSAGES
2190     yesexpr  "^[yY][[:alpha:]]*" | (OK) "
2191     #
2192     noexpr   "^[nN][[:alpha:]]*"
2193     END LC_MESSAGES

```

## 2194 **A.8 Environment Variables**

### 2195 **A.8.1 Environment Variable Definition**

2196 The variable *environ* is not intended to be declared in any header, but rather to be declared by the  
 2197 user for accessing the array of strings that is the environment. This is the traditional usage of the  
 2198 symbol. Putting it into a header could break some programs that use the symbol for their own  
 2199 purposes.

2200 The decision to restrict conforming systems to the use of digits, uppercase letters, and  
 2201 underscores for environment variable names allows applications to use lowercase letters in their  
 2202 environment variable names without conflicting with any conforming system.

2203 In addition to the obvious conflict with the shell syntax for positional parameter substitution,  
 2204 some historical applications (including some shells) exclude names with leading digits from the  
 2205 environment.

### 2206 **A.8.2 Internationalization Variables**

2207 The text about locale implies that any utilities written in standard C and conforming to  
 2208 IEEE Std 1003.1-200x must issue the following call:

```
2209     setlocale(LC_ALL, "")
```

2210 If this were omitted, the ISO C standard specifies that the C locale would be used.

2211 If any of the environment variables are invalid, it makes sense to default to an implementation-  
 2212 defined, consistent locale environment. It is more confusing for a user to have partial settings  
 2213 occur in case of a mistake. All utilities would then behave in one language/cultural  
 2214 environment. Furthermore, it provides a way of forcing the whole environment to be the  
 2215 implementation-defined default. Disastrous results could occur if a pipeline of utilities partially  
 2216 uses the environment variables in different ways. In this case, it would be appropriate for  
 2217 utilities that use *LANG* and related variables to exit with an error if any of the variables are  
 2218 invalid. For example, users typing individual commands at a terminal might want *date* to work if  
 2219 *LC\_MONETARY* is invalid as long as *LC\_TIME* is valid. Since these are conflicting reasonable  
 2220 alternatives, IEEE Std 1003.1-200x leaves the results unspecified if the locale environment  
 2221 variables would not produce a complete locale matching the specification of the user.

2222 The locale settings of individual categories cannot be truly independent and still guarantee  
 2223 correct results. For example, when collating two strings, characters must first be extracted from  
 2224 each string (governed by *LC\_CTYPE*) before being mapped to collating elements (governed by  
 2225 *LC\_COLLATE*) for comparison. That is, if *LC\_CTYPE* is causing parsing according to the rules of  
 2226 a large, multi-byte code set (potentially returning 20 000 or more distinct character codeset  
 2227 values), but *LC\_COLLATE* is set to handle only an 8-bit codeset with 256 distinct characters,  
 2228 meaningful results are obviously impossible.

2229 The *LC\_MESSAGES* variable affects the language of messages generated by the standard  
 2230 utilities.

2231 The description of the environment variable names starting with the characters “LC\_”  
 2232 acknowledges the fact that the interfaces presented may be extended as new international  
 2233 functionality is required. In the ISO C standard, names preceded by “LC\_” are reserved in the  
 2234 name space for future categories.

2235 To avoid name clashes, new categories and environment variables are divided into two  
 2236 classifications: *implementation-independent* and *implementation-defined*.

2237 Implementation-independent names will have the following format:

2238 `LC_NAME`

2239 where *NAME* is the name of the new category and environment variable. Capital letters must be  
 2240 used for implementation-independent names.

2241 Implementation-defined names must be in lowercase letters, as below:

2242 `LC_name`

### 2243 **A.8.3 Other Environment Variables**

2244 The quoted form of the timezone variable allows timezone names of the form UTC+1 (or any  
 2245 name that contains the character plus ('+'), the character minus ('-'), or digits), which may be  
 2246 appropriate for countries that do not have an official timezone name. It would be coded as  
 2247 <UTC+1>+1<UTC+2>, which would cause *std* to have a value of UTC+1 and *dst* a value of  
 2248 UTC+2, each with a length of 5 characters. This does not appear to conflict with any existing  
 2249 usage. The characters '<' and '>' were chosen for quoting because they are easier to parse  
 2250 visually than a quoting character that does not provide some sense of bracketing (and in a string  
 2251 like this, such bracketing is helpful). They were also chosen because they do not need special  
 2252 treatment when assigning to the *TZ* variable. Users are often confused by embedding quotes in a  
 2253 string. Because '<' and '>' are meaningful to the shell, the whole string would have to be  
 2254 quoted, but that is easily explained. (Parentheses would have presented the same problems.)  
 2255 Although the '>' symbol could have been permitted in the string by either escaping it or  
 2256 doubling it, it seemed of little value to require that. This could be provided as an extension if  
 2257 there was a need. Timezone names of this new form lead to a requirement that the value of  
 2258 `{_POSIX_TZNAME_MAX}` change from 3 to 6.

### 2259 **COLUMNS, LINES**

2260 The default value for the number of column positions, *COLUMNS*, and screen height, *LINES*, are  
 2261 unspecified because historical implementations use different methods to determine values  
 2262 corresponding to the size of the screen in which the utility is run. This size is typically known to  
 2263 the implementation through the value of *TERM*, or by more elaborate methods such as  
 2264 extensions to the *stty* utility or knowledge of how the user is dynamically resizing windows on a  
 2265 bit-mapped display terminal. Users should not need to set these variables in the environment  
 2266 unless there is a specific reason to override the default behavior of the implementation, such as

2267 to display data in an area arbitrarily smaller than the terminal or window. Values for these  
 2268 variables that are not decimal integers greater than zero are implicitly undefined values; it is  
 2269 unnecessary to enumerate all of the possible values outside of the acceptable set.

## 2270 **PATH**

2271 Many historical implementations of the Bourne shell do not interpret a trailing colon to represent  
 2272 the current working directory and are thus non-conforming. The C Shell and the KornShell  
 2273 conform to IEEE Std 1003.1-200x on this point. The usual name of dot may also be used to refer  
 2274 to the current working directory.

2275 Many implementations historically have used a default value of **/bin** and **/usr/bin** for the *PATH*  
 2276 variable. IEEE Std 1003.1-200x does not mandate this default path be identical to that retrieved  
 2277 from *getconf \_CS\_PATH* because it is likely that the standardized utilities may be provided in  
 2278 another directory separate from the directories used by some historical applications.

## 2279 **LOGNAME**

2280 In most implementations, the value of such a variable is easily forged, so security-critical  
 2281 applications should rely on other means of determining user identity. *LOGNAME* is required to  
 2282 be constructed from the portable filename character set for reasons of interchange. No diagnostic  
 2283 condition is specified for violating this rule, and no requirement for enforcement exists. The  
 2284 intent of the requirement is that if extended characters are used, the “guarantee” of portability  
 2285 implied by a standard is void.

## 2286 **SHELL**

2287 The *SHELL* variable names the preferred shell of the user; it is a guide to applications. There is  
 2288 no direct requirement that that shell conform to IEEE Std 1003.1-200x; that decision should rest  
 2289 with the user. It is the intention of the standard developers that alternative shells be permitted, if  
 2290 the user chooses to develop or acquire one. An operating system that builds its shell into the  
 2291 “kernel” in such a manner that alternative shells would be impossible does not conform to the  
 2292 spirit of IEEE Std 1003.1-200x.

## 2293 **CHANGE HISTORY**

### 2294 **Issue 6**

2295 Changed format of *TZ* field to allow for the quoted form as defined in previous  
 2296 versions of the ISO POSIX-1 standard.

## 2297 **A.9 Regular Expressions**

2298 Rather than repeating the description of REs for each utility supporting REs, the standard  
 2299 developers preferred a common, comprehensive description of regular expressions in one place.  
 2300 The most common behavior is described here, and exceptions or extensions to this are  
 2301 documented for the respective utilities, as appropriate.

2302 The BRE corresponds to the *ed* or historical *grep* type, and the ERE corresponds to the historical  
 2303 *egrep* type (now *grep -E*).

2304 The text is based on the *ed* description and substantially modified, primarily to aid developers  
 2305 and others in the understanding of the capabilities and limitations of REs. Much of this was  
 2306 influenced by internationalization requirements.

2307 It should be noted that the definitions in this section do not cover the *tr* utility; the *tr* syntax does  
2308 not employ REs.

2309 The specification of REs is particularly important to internationalization because pattern  
2310 matching operations are very basic operations in business and other operations. The syntax and  
2311 rules of REs are intended to be as intuitive as possible to make them easy to understand and use.  
2312 The historical rules and behavior do not provide that capability to non-English language users,  
2313 and do not provide the necessary support for commonly used characters and language  
2314 constructs. It was necessary to provide extensions to the historical RE syntax and rules to  
2315 accommodate other languages.

2316 As they are limited to bracket expressions, the rationale for these modifications is in the Base  
2317 Definitions volume of IEEE Std 1003.1-200x, Section 9.3.5, RE Bracket Expression.

### 2318 A.9.1 Regular Expression Definitions

2319 It is possible to determine what strings correspond to subexpressions by recursively applying  
2320 the leftmost longest rule to each subexpression, but only with the proviso that the overall match  
2321 is leftmost longest. For example, matching "`\(ac*\\)c*d[ac]*\1`" against *acdacaaa* matches  
2322 *acdacaaa* (with `\1=a`); simply matching the longest match for "`\(ac*\\)`" would yield `\1=ac`, but  
2323 the overall match would be smaller (*acdac*). Conceptually, the implementation must examine  
2324 every possible match and among those that yield the leftmost longest total matches, pick the one  
2325 that does the longest match for the leftmost subexpression, and so on. Note that this means that  
2326 matching by subexpressions is context-dependent: a subexpression within a larger RE may  
2327 match a different string from the one it would match as an independent RE, and two instances of  
2328 the same subexpression within the same larger RE may match different lengths even in similar  
2329 sequences of characters. For example, in the ERE "`(a.*b)(a.*b)`", the two identical  
2330 subexpressions would match four and six characters, respectively, of *accbacccb*.

2331 The definition of *single character* has been expanded to include also collating elements consisting  
2332 of two or more characters; this expansion is applicable only when a bracket expression is  
2333 included in the BRE or ERE. An example of such a collating element may be the Dutch *ij*, which  
2334 collates as a 'y'. In some encodings, a ligature "i with j" exists as a character and would  
2335 represent a single-character collating element. In another encoding, no such ligature exists, and  
2336 the two-character sequence *ij* is defined as a multi-character collating element. Outside brackets,  
2337 the *ij* is treated as a two-character RE and matches the same characters in a string. Historically, a  
2338 bracket expression only matched a single character. The ISO POSIX-2:1993 standard required  
2339 bracket expressions like "`^[[:lower:]]`" to match multi-character collating elements such as  
2340 "*ij*". However, this requirement led to behavior that many users did not expect and that could  
2341 not feasibly be mimicked in user code, and it was rarely if ever implemented correctly. The  
2342 current standard leaves it unspecified whether a bracket expression matches a multi-character  
2343 collating element, allowing both historical and ISO POSIX-2:1993 standard implementations to  
2344 conform.

2345 Also, in the current standard, it is unspecified whether character class expressions like  
2346 "`[:lower:]`" can include multi-character collating elements like "*ij*"; hence  
2347 "`[[:lower:]]`" can match "*ij*", and "`^[[:lower:]]`" can fail to match "*ij*". Common  
2348 practice is for a character class expression to match a collating element if it matches the collating  
2349 element's first character.

2350 **A.9.2 Regular Expression General Requirements**

2351 The definition of which sequence is matched when several are possible is based on the leftmost-  
2352 longest rule historically used by deterministic recognizers. This rule is easier to define and  
2353 describe, and arguably more useful, than the first-match rule historically used by non-  
2354 deterministic recognizers. It is thought that dependencies on the choice of rule are rare; carefully  
2355 contrived examples are needed to demonstrate the difference.

2356 A formal expression of the leftmost-longest rule is:

2357       The search is performed as if all possible suffixes of the string were tested for a prefix  
2358       matching the pattern; the longest suffix containing a matching prefix is chosen, and the  
2359       longest possible matching prefix of the chosen suffix is identified as the matching sequence.

2360 Historically, most RE implementations only match lines, not strings. However, that is more an  
2361 effect of the usage than of an inherent feature of REs themselves. Consequently,  
2362 IEEE Std 1003.1-200x does not regard <newline>s as special; they are ordinary characters, and  
2363 both a period and a non-matching list can match them. Those utilities (like *grep*) that do not  
2364 allow <newline>s to match are responsible for eliminating any <newline> from strings before  
2365 matching against the RE. The *regcomp()* function, however, can provide support for such  
2366 processing without violating the rules of this section.

2367 Some implementations of *egrep* have had very limited flexibility in handling complex EREs.  
2368 IEEE Std 1003.1-200x does not attempt to define the complexity of a BRE or ERE, but does place  
2369 a lower limit on it—any RE must be handled, as long as it can be expressed in 256 bytes or less.  
2370 (Of course, this does not place an upper limit on the implementation.) There are historical  
2371 programs using a non-deterministic-recognizer implementation that should have no difficulty  
2372 with this limit. It is possible that a good approach would be to attempt to use the faster, but  
2373 more limited, deterministic recognizer for simple expressions and to fall back on the non-  
2374 deterministic recognizer for those expressions requiring it. Non-deterministic implementations  
2375 must be careful to observe the rules on which match is chosen; the longest match, not the first  
2376 match, starting at a given character is used.

2377 The term *invalid* highlights a difference between this section and some others:  
2378 IEEE Std 1003.1-200x frequently avoids mandating of errors for syntax violations because they  
2379 can be used by implementors to trigger extensions. However, the authors of the  
2380 internationalization features of REs wanted to mandate errors for certain conditions to identify  
2381 usage problems or non-portable constructs. These are identified within this rationale as  
2382 appropriate. The remaining syntax violations have been left implicitly or explicitly undefined.  
2383 For example, the BRE construct " $\{1,2,3\}$ " does not comply with the grammar. A  
2384 conforming application cannot rely on it producing an error nor matching the literal characters  
2385 " $\{1,2,3\}$ ". The term "undefined" was used in favor of "unspecified" because many of the  
2386 situations are considered errors on some implementations, and the standard developers  
2387 considered that consistency throughout the section was preferable to mixing undefined and  
2388 unspecified.

2389 **A.9.3 Basic Regular Expressions**

2390 There is no additional rationale provided for this section.

2391 *A.9.3.1 BREs Matching a Single Character or Collating Element*

2392 There is no additional rationale provided for this section.

2393 *A.9.3.2 BRE Ordinary Characters*

2394 There is no additional rationale provided for this section.

2395 *A.9.3.3 BRE Special Characters*

2396 There is no additional rationale provided for this section.

2397 *A.9.3.4 Periods in BREs*

2398 There is no additional rationale provided for this section.

2399 *A.9.3.5 RE Bracket Expression*

2400 Range expressions are, historically, an integral part of REs. However, the requirements of  
 2401 “natural language behavior” and portability do conflict. In the POSIX locale, ranges must be  
 2402 treated according to the collating sequence and include such characters that fall within the range  
 2403 based on that collating sequence, regardless of character values. In other locales, ranges have  
 2404 unspecified behavior.

2405 Some historical implementations allow range expressions where the ending range point of one  
 2406 range is also the starting point of the next (for instance, "[a-m-o]"). This behavior should not  
 2407 be permitted, but to avoid breaking historical implementations, it is now *undefined* whether it is a  
 2408 valid expression and how it should be interpreted.

2409 Current practice in *awk* and *lex* is to accept escape sequences in bracket expressions as per the  
 2410 Base Definitions volume of IEEE Std 1003.1-200x, Table 5-1, Escape Sequences and Associated  
 2411 Actions, while the normal ERE behavior is to regard such a sequence as consisting of two  
 2412 characters. Allowing the *awk/lex* behavior in EREs would change the normal behavior in an  
 2413 unacceptable way; it is expected that *awk* and *lex* will decode escape sequences in EREs before  
 2414 passing them to *regcomp()* or comparable routines. Each utility describes the escape sequences it  
 2415 accepts as an exception to the rules in this section; the list is not the same, for historical reasons.

2416 As noted previously, the new syntax and rules have been added to accommodate other  
 2417 languages than English. The remainder of this section describes the rationale for these  
 2418 modifications.

2419 In the POSIX locale, a regular expression that starts with a range expression matches a set of  
 2420 strings that are contiguously sorted, but this is not necessarily true in other locales. For example,  
 2421 a French locale might have the following behavior:

```
2422 $ ls
2423 alpha Alpha estimé ESTIMÉ été eurêka
2424 $ ls [a-e]*
2425 alpha Alpha estimé eurêka
```

2426 Such disagreements between matching and contiguous sorting are unavoidable because POSIX  
 2427 sorting cannot be implemented in terms of a deterministic finite-state automaton (DFA), but  
 2428 range expressions by design are implementable in terms of DFAs.

2429 Historical implementations used native character order to interpret range expressions. The  
 2430 ISO POSIX-2:1993 standard instead required collating element order (CEO): the order that  
 2431 collating elements were specified between the **order\_start** and **order\_end** keywords in the  
 2432 *LC\_COLLATE* category of the current locale. CEO had some advantages in portability over the  
 2433 native character order, but it also had some disadvantages:

- 2434 • CEO could not feasibly be mimicked in user code, leading to inconsistencies between POSIX  
 2435 matchers and matchers in popular user programs like Emacs, *ksh*, and Perl.
- 2436 • CEO caused range expressions to match accented and capitalized letters contrary to many  
 2437 users' expectations. For example, "[a-e]" typically matched both 'E' and 'á' but neither  
 2438 'A' nor 'é'.
- 2439 • CEO was not consistent across implementations. In practice, CEO was often less portable  
 2440 than native character order. For example, it was common for the CEOs of two  
 2441 implementation-supplied locales to disagree, even if both locales were named "da\_DK".

2442 Because of these problems, some implementations of regular expressions continued to use  
 2443 native character order. Others used the collation sequence, which is more consistent with sorting  
 2444 than either CEO or native order, but which departs further from the traditional POSIX semantics  
 2445 because it generally requires "[a-e]" to match either 'A' or 'E' but not both. As a result of  
 2446 this kind of implementation variation, programmers who wanted to write portable regular  
 2447 expressions could not rely on the ISO POSIX-2:1993 standard guarantees in practice.

2448 While revising the standard, lengthy consideration was given to proposals to attack this problem  
 2449 by adding an API for querying the CEO to allow user-mode matchers, but none of these  
 2450 proposals had implementation experience and none achieved consensus. Leaving the standard  
 2451 alone was also considered, but rejected due to the problems described above.

2452 The current standard leaves unspecified the behavior of a range expression outside the POSIX  
 2453 locale. This makes it clearer that conforming applications should avoid range expressions  
 2454 outside the POSIX locale, and it allows implementations and compatible user-mode matchers to  
 2455 interpret range expressions using native order, CEO, collation sequence, or other, more  
 2456 advanced techniques.

2457 The ISO POSIX-2:1993 standard required "[b-a]" to be an invalid expression in the POSIX  
 2458 locale, but this requirement has been relaxed in this version of the standard so that "[b-a]" can  
 2459 instead be treated as a valid expression that does not match any string.

#### 2460 A.9.3.6 BREs Matching Multiple Characters

2461 The limit of nine back-references to subexpressions in the RE is based on the use of a single-digit  
 2462 identifier; increasing this to multiple digits would break historical applications. This does not  
 2463 imply that only nine subexpressions are allowed in REs. The following is a valid BRE with ten  
 2464 subexpressions:

```
2465 \(\(\(ab\) *c\) *d\)\(ef\) *\ (gh)\{2\}\(ij\) *\ (kl)\ *(mn)\ *(op)\ *(qr)\ *
```

2466 The standard developers regarded the common historical behavior, which supported "\n\*", but  
 2467 not "\n\{min,max\}", "\(\.\.\)\\*", or "\(\.\.\)\{min,max\}", as a non-intentional  
 2468 result of a specific implementation, and they supported both duplication and interval  
 2469 expressions following subexpressions and back-references.

2470 The changes to the processing of the back-reference expression remove an unspecified or  
 2471 ambiguous behavior in the Shell and Utilities volume of IEEE Std 1003.1-200x, aligning it with  
 2472 the requirements specified for the *regcomp()* expression, and is the result of PASC Interpretation  
 2473 1003.2-92 #43 submitted for the ISO POSIX-2:1993 standard.

## 2474 A.9.3.7 BRE Precedence

2475 There is no additional rationale provided for this section.

## 2476 A.9.3.8 BRE Expression Anchoring

2477 Often, the dollar sign is viewed as matching the ending <newline> in text files. This is not  
2478 strictly true; the <newline> is typically eliminated from the strings to be matched, and the dollar  
2479 sign matches the terminating null character.

2480 The ability of '^', '\$', and '\*' to be non-special in certain circumstances may be confusing to  
2481 some programmers, but this situation was changed only in a minor way from historical practice  
2482 to avoid breaking many historical scripts. Some consideration was given to making the use of  
2483 the anchoring characters undefined if not escaped and not at the beginning or end of strings.  
2484 This would cause a number of historical BREs, such as "2^10", "\$HOME", and "\$1.35", that  
2485 relied on the characters being treated literally, to become invalid.

2486 However, one relatively uncommon case was changed to allow an extension used on some  
2487 implementations. Historically, the BREs "^foo" and "\(^foo\)" did not match the same  
2488 string, despite the general rule that subexpressions and entire BREs match the same strings. To  
2489 increase consensus, IEEE Std 1003.1-200x has allowed an extension on some implementations to  
2490 treat these two cases in the same way by declaring that anchoring *may* occur at the beginning or  
2491 end of a subexpression. Therefore, portable BREs that require a literal circumflex at the  
2492 beginning or a dollar sign at the end of a subexpression must escape them. Note that a BRE such  
2493 as "a\(^bc\)" will either match "a^bc" or nothing on different systems under the rules.

2494 ERE anchoring has been different from BRE anchoring in all historical systems. An unescaped  
2495 anchor character has never matched its literal counterpart outside a bracket expression. Some  
2496 implementations treated "foo\$bar" as a valid expression that never matched anything; others  
2497 treated it as invalid. IEEE Std 1003.1-200x mandates the former, valid unmatched behavior.

2498 Some implementations have extended the BRE syntax to add alternation. For example, the  
2499 subexpression "\ (foo\$|bar\)" would match either "foo" at the end of the string or "bar"  
2500 anywhere. The extension is triggered by the use of the undefined "\|" sequence. Because the  
2501 BRE is undefined for portable scripts, the extending system is free to make other assumptions,  
2502 such that the '\$' represents the end-of-line anchor in the middle of a subexpression. If it were  
2503 not for the extension, the '\$' would match a literal dollar sign under the rules.

## 2504 A.9.4 Extended Regular Expressions

2505 As with BREs, the standard developers decided to make the interpretation of escaped ordinary  
2506 characters undefined.

2507 The right parenthesis is not listed as an ERE special character because it is only special in the  
2508 context of a preceding left parenthesis. If found without a preceding left parenthesis, the right  
2509 parenthesis has no special meaning.

2510 The *interval expression*, "{m,n}", has been added to EREs. Historically, the interval expression  
2511 has only been supported in some ERE implementations. The standard developers estimated that  
2512 the addition of interval expressions to EREs would not decrease consensus and would also make  
2513 BREs more of a subset of EREs than in many historical implementations.

2514 It was suggested that, in addition to interval expressions, back-references ('\n') should also be  
2515 added to EREs. This was rejected by the standard developers as likely to decrease consensus.

2516 In historical implementations, multiple duplication symbols are usually interpreted from left to  
2517 right and treated as additive. As an example, "a+b" matches zero or more instances of 'a'  
2518 followed by a 'b'. In IEEE Std 1003.1-200x, multiple duplication symbols are undefined; that is,

- 2519 they cannot be relied upon for conforming applications. One reason for this is to provide some |  
 2520 scope for future enhancements.
- 2521 The precedence of operations differs between EREs and those in *lex*; in *lex*, for historical reasons,  
 2522 interval expressions have a lower precedence than concatenation.
- 2523 **A.9.4.1 EREs Matching a Single Character or Collating Element**
- 2524 There is no additional rationale provided for this section.
- 2525 **A.9.4.2 ERE Ordinary Characters**
- 2526 There is no additional rationale provided for this section.
- 2527 **A.9.4.3 ERE Special Characters**
- 2528 There is no additional rationale provided for this section.
- 2529 **A.9.4.4 Periods in EREs**
- 2530 There is no additional rationale provided for this section.
- 2531 **A.9.4.5 ERE Bracket Expression**
- 2532 There is no additional rationale provided for this section.
- 2533 **A.9.4.6 EREs Matching Multiple Characters**
- 2534 There is no additional rationale provided for this section.
- 2535 **A.9.4.7 ERE Alternation**
- 2536 There is no additional rationale provided for this section.
- 2537 **A.9.4.8 ERE Precedence**
- 2538 There is no additional rationale provided for this section.
- 2539 **A.9.4.9 ERE Expression Anchoring**
- 2540 There is no additional rationale provided for this section.
- 2541 **A.9.5 Regular Expression Grammar**
- 2542 The grammars are intended to represent the range of acceptable syntaxes available to |  
 2543 conforming applications. There are instances in the text where undefined constructs are |  
 2544 described; as explained previously, these allow implementation extensions. There is no intended  
 2545 requirement that an implementation extension must somehow fit into the grammars shown  
 2546 here.
- 2547 The BRE grammar does not permit L\_ANCHOR or R\_ANCHOR inside "\(" and "\)" (which  
 2548 implies that '^' and '\$' are ordinary characters). This reflects the semantic limits on the  
 2549 application, as noted in the Base Definitions volume of IEEE Std 1003.1-200x, Section 9.3.8, BRE  
 2550 Expression Anchoring. Implementations are permitted to extend the language to interpret '^'  
 2551 and '\$' as anchors in these locations, and as such, conforming applications cannot use |  
 2552 unescaped '^' and '\$' in positions inside "\(" and "\)" that might be interpreted as anchors.
- 2553 The ERE grammar does not permit several constructs that the Base Definitions volume of  
 2554 IEEE Std 1003.1-200x, Section 9.4.2, ERE Ordinary Characters and the Base Definitions volume of

2555 IEEE Std 1003.1-200x, Section 9.4.3, ERE Special Characters specify as having undefined results:

- 2556 • ORD\_CHAR preceded by '\'
- 2557 • *ERE\_dupl\_symbol*(s) appearing first in an ERE, or immediately following '|', '^', or '('
- 2558 • '{' not part of a valid *ERE\_dupl\_symbol*
- 2559 • '|' appearing first or last in an ERE, or immediately following '|' or '(', or immediately
- 2560 preceding ')'

2561 Implementations are permitted to extend the language to allow these. Conforming applications |  
2562 cannot use such constructs. |

#### 2563 A.9.5.1 BRE/ERE Grammar Lexical Conventions

2564 There is no additional rationale provided for this section.

#### 2565 A.9.5.2 RE and Bracket Expression Grammar

2566 The removal of the *Back\_open\_paren Back\_close\_paren* option from the *nondupl\_RE* specification is  
2567 the result of PASC Interpretation 1003.2-92 #43 submitted for the ISO POSIX-2: 1993 standard.  
2568 Although the grammar required support for null subexpressions, this section does not describe  
2569 the meaning of, and historical practice did not support, this construct.

#### 2570 A.9.5.3 ERE Grammar

2571 There is no additional rationale provided for this section.

## 2572 A.10 Directory Structure and Devices

### 2573 A.10.1 Directory Structure and Files

2574 A description of the historical **/usr/tmp** was omitted, removing any concept of differences in  
2575 emphasis between the / and **/usr** directories. The descriptions of **/bin**, **/usr/bin**, **/lib**, and **/usr/lib**  
2576 were omitted because they are not useful for applications. In an early draft, a distinction was  
2577 made between system and application directory usage, but this was not found to be useful.

2578 The directories / and **/dev** are included because the notion of a hierarchical directory structure is  
2579 key to other information presented elsewhere in IEEE Std 1003.1-200x. In early drafts, it was  
2580 argued that special devices and temporary files could conceivably be handled without a  
2581 directory structure on some implementations. For example, the system could treat the characters  
2582 `"/tmp"` as a special token that would store files using some non-POSIX file system structure.  
2583 This notion was rejected by the standard developers, who required that all the files in this  
2584 section be implemented via POSIX file systems.

2585 The **/tmp** directory is retained in IEEE Std 1003.1-200x to accommodate historical applications  
2586 that assume its availability. Implementations are encouraged to provide suitable directory  
2587 names in the environment variable *TMPDIR* and applications are encouraged to use the contents  
2588 of *TMPDIR* for creating temporary files.

2589 The standard files **/dev/null** and **/dev/tty** are required to be both readable and writable to allow  
2590 applications to have the intended historical access to these files.

2591 The standard file **/dev/console** has been added for alignment with the Single UNIX Specification.

**2592 A.10.2 Output Devices and Terminal Types**

2593 There is no additional rationale provided for this section.

**2594 A.11 General Terminal Interface**

2595 If the implementation does not support this interface on any device types, it should behave as if  
2596 it were being used on a device that is not a terminal device (in most cases *errno* will be set to  
2597 [ENOTTY] on return from functions defined by this interface). This is based on the fact that  
2598 many applications are written to run both interactively and in some non-interactive mode, and  
2599 they adapt themselves at runtime. Requiring that they all be modified to test an environment  
2600 variable to determine whether they should try to adapt is unnecessary. On a system that  
2601 provides no general terminal interface, providing all the entry points as stubs that return  
2602 [ENOTTY] (or an equivalent, as appropriate) has the same effect and requires no changes to the  
2603 application.

2604 Although the needs of both interface implementors and application developers were addressed  
2605 throughout IEEE Std 1003.1-200x, this section pays more attention to the needs of the latter. This  
2606 is because, while many aspects of the programming interface can be hidden from the user by the  
2607 application developer, the terminal interface is usually a large part of the user interface.  
2608 Although to some extent the application developer can build missing features or work around  
2609 inappropriate ones, the difficulties of doing that are greater in the terminal interface than  
2610 elsewhere. For example, efficiency prohibits the average program from interpreting every  
2611 character passing through it in order to simulate character erase, line kill, and so on. These  
2612 functions should usually be done by the operating system, possibly at the interrupt level.

2613 The *tc\**() functions were introduced as a way of avoiding the problems inherent in the  
2614 traditional *ioctl*() function and in variants of it that were proposed. For example, *tcsetattr*()  
2615 is specified in place of the use of the TCSETA *ioctl*() command function. This allows specification  
2616 of all the arguments in a manner consistent with the ISO C standard unlike the varying third  
2617 argument of *ioctl*(), which is sometimes a pointer (to any of many different types) and  
2618 sometimes an **int**.

2619 The advantages of this new method include:

- 2620 • It allows strict type checking.
- 2621 • The direction of transfer of control data is explicit.
- 2622 • Portable capabilities are clearly identified.
- 2623 • The need for a general interface routine is avoided.
- 2624 • Size of the argument is well-defined (there is only one type).

2625 The disadvantages include:

- 2626 • No historical implementation used the new method.
- 2627 • There are many small routines instead of one general-purpose one.
- 2628 • The historical parallel with *fcntl*() is broken.

2629 The issue of modem control was excluded from IEEE Std 1003.1-200x on the grounds that:

- 2630 • It was concerned with setting and control of hardware timers.
- 2631 • The appropriate timers and settings vary widely internationally.

2632           • Feedback from European computer manufacturers indicated that this facility was not  
2633           consistent with European needs and that specification of such a facility was not a  
2634           requirement for portability.

### 2635 **A.11.1 Interface Characteristics**

#### 2636 *A.11.1.1 Opening a Terminal Device File*

2637           There is no additional rationale provided for this section.

#### 2638 *A.11.1.2 Process Groups*

2639           There is a potential race when the members of the foreground process group on a terminal leave  
2640           that process group, either by exit or by changing process groups. After the last process exits the  
2641           process group, but before the foreground process group ID of the terminal is changed (usually  
2642           by a job-control shell), it would be possible for a new process to be created with its process ID  
2643           equal to the terminal's foreground process group ID. That process might then become the  
2644           process group leader and accidentally be placed into the foreground on a terminal that was not  
2645           necessarily its controlling terminal. As a result of this problem, the controlling terminal is  
2646           defined to not have a foreground process group during this time.

2647           The cases where a controlling terminal has no foreground process group occur when all  
2648           processes in the foreground process group either terminate and are waited for or join other  
2649           process groups via *setpgid()* or *setsid()*. If the process group leader terminates, this is the first  
2650           case described; if it leaves the process group via *setpgid()*, this is the second case described (a  
2651           process group leader cannot successfully call *setsid()*). When one of those cases causes a  
2652           controlling terminal to have no foreground process group, it has two visible effects on  
2653           applications. The first is the value returned by *tcgetpgrp()*. The second (which occurs only in the  
2654           case where the process group leader terminates) is the sending of signals in response to special  
2655           input characters. The intent of IEEE Std 1003.1-200x is that no process group be wrongly  
2656           identified as the foreground process group by *tcgetpgrp()* or unintentionally receive signals  
2657           because of placement into the foreground.

2658           In 4.3 BSD, the old process group ID continues to be used to identify the foreground process  
2659           group and is returned by the function equivalent to *tcgetpgrp()*. In that implementation it is  
2660           possible for a newly created process to be assigned the same value as a process ID and then form  
2661           a new process group with the same value as a process group ID. The result is that the new  
2662           process group would receive signals from this terminal for no apparent reason, and  
2663           IEEE Std 1003.1-200x precludes this by forbidding a process group from entering the foreground  
2664           in this way. It would be more direct to place part of the requirement made by the last sentence  
2665           under *fork()*, but there is no convenient way for that section to refer to the value that *tcgetpgrp()*  
2666           returns, since in this case there is no process group and thus no process group ID.

2667           One possibility for a conforming implementation is to behave similarly to 4.3 BSD, but to  
2668           prevent this reuse of the ID, probably in the implementation of *fork()*, as long as it is in use by  
2669           the terminal.

2670           Another possibility is to recognize when the last process stops using the terminal's foreground  
2671           process group ID, which is when the process group lifetime ends, and to change the terminal's  
2672           foreground process group ID to a reserved value that is never used as a process ID or process  
2673           group ID. (See the definition of *process group lifetime* in the definitions section.) The process ID  
2674           can then be reserved until the terminal has another foreground process group.

2675           The 4.3 BSD implementation permits the leader (and only member) of the foreground process  
2676           group to leave the process group by calling the equivalent of *setpgid()* and to later return,  
2677           expecting to return to the foreground. There are no known application needs for this behavior,

2678 and IEEE Std 1003.1-200x neither requires nor forbids it (except that it is forbidden for session  
2679 leaders) by leaving it unspecified.

#### 2680 A.11.1.3 *The Controlling Terminal*

2681 IEEE Std 1003.1-200x does not specify a mechanism by which to allocate a controlling terminal.  
2682 This is normally done by a system utility (such as *getty*) and is considered an administrative  
2683 feature outside the scope of IEEE Std 1003.1-200x.

2684 Historical implementations allocate controlling terminals on certain *open()* calls. Since *open()* is  
2685 part of POSIX.1, its behavior had to be dealt with. The traditional behavior is not required  
2686 because it is not very straightforward or flexible for either implementations or applications.  
2687 However, because of its prevalence, it was not practical to disallow this behavior either. Thus, a  
2688 mechanism was standardized to ensure portable, predictable behavior in *open()*.

2689 Some historical implementations deallocate a controlling terminal on the last system-wide close.  
2690 This behavior is neither required nor prohibited. Even on implementations that do provide this  
2691 behavior, applications generally cannot depend on it due to its system-wide nature.

#### 2692 A.11.1.4 *Terminal Access Control*

2693 The access controls described in this section apply only to a process that is accessing its  
2694 controlling terminal. A process accessing a terminal that is not its controlling terminal is  
2695 effectively treated the same as a member of the foreground process group. While this may seem  
2696 unintuitive, note that these controls are for the purpose of job control, not security, and job  
2697 control relates only to a process' controlling terminal. Normal file access permissions handle  
2698 security.

2699 If the process calling *read()* or *write()* is in a background process group that is orphaned, it is not  
2700 desirable to stop the process group, as it is no longer under the control of a job control shell that  
2701 could put it into foreground again. Accordingly, calls to *read()* or *write()* functions by such  
2702 processes receive an immediate error return. This is different from 4.2 BSD, which kills orphaned  
2703 processes that receive terminal stop signals.

2704 The foreground/background/orphaned process group check performed by the terminal driver  
2705 must be repeatedly performed until the calling process moves into the foreground or until the  
2706 process group of the calling process becomes orphaned. That is, when the terminal driver  
2707 determines that the calling process is in the background and should receive a job control signal,  
2708 it sends the appropriate signal (SIGTTIN or SIGTTOU) to every process in the process group of  
2709 the calling process and then it allows the calling process to immediately receive the signal. The  
2710 latter is typically performed by blocking the process so that the signal is immediately noticed.  
2711 Note, however, that after the process finishes receiving the signal and control is returned to the  
2712 driver, the terminal driver must reexecute the foreground/background/orphaned process group  
2713 check. The process may still be in the background, either because it was continued in the  
2714 background by a job-control shell, or because it caught the signal and did nothing.

2715 The terminal driver repeatedly performs the foreground/background/orphaned process group  
2716 checks whenever a process is about to access the terminal. In the case of *write()* or the control  
2717 *tc\*()* functions, the check is performed at the entry of the function. In the case of *read()*, the check  
2718 is performed not only at the entry of the function, but also after blocking the process to wait for  
2719 input characters (if necessary). That is, once the driver has determined that the process calling  
2720 the *read()* function is in the foreground, it attempts to retrieve characters from the input queue.  
2721 If the queue is empty, it blocks the process waiting for characters. When characters are available  
2722 and control is returned to the driver, the terminal driver must return to the repeated  
2723 foreground/background/orphaned process group check again. The process may have moved  
2724 from the foreground to the background while it was blocked waiting for input characters.

2725 A.11.1.5 *Input Processing and Reading Data*

2726 There is no additional rationale provided for this section.

2727 A.11.1.6 *Canonical Mode Input Processing*

2728 The term *character* is intended here. ERASE should erase the last character, not the last byte. In  
2729 the case of multi-byte characters, these two may be different.

2730 4.3 BSD has a WERASE character that erases the last “word” typed (but not any preceding  
2731 <blank>s or <tab>s). A word is defined as a sequence of non-<blank>s, with <tab>s counted as  
2732 <blank>s. Like ERASE, WERASE does not erase beyond the beginning of the line. This  
2733 WERASE feature has not been specified in POSIX.1 because it is difficult to define in the  
2734 international environment. It is only useful for languages where words are delimited by  
2735 <blank>s. In some ideographic languages, such as Japanese and Chinese, words are not  
2736 delimited at all. The WERASE character should presumably take one back to the beginning of a  
2737 sentence in those cases; practically, this means it would not get much use for those languages.

2738 It should be noted that there is a possible inherent deadlock if the application and  
2739 implementation conflict on the value of MAX\_CANON. With ICANON set (if IXOFF is  
2740 enabled) and more than MAX\_CANON characters transmitted without a <linefeed>,  
2741 transmission will be stopped, the <linefeed> (or <carriage-return> when ICRLF is set) will never  
2742 arrive, and the *read()* will never be satisfied.

2743 An application should not set IXOFF if it is using canonical mode unless it knows that (even in  
2744 the face of a transmission error) the conditions described previously cannot be met or unless it is  
2745 prepared to deal with the possible deadlock in some other way, such as timeouts.

2746 It should also be noted that this can be made to happen in non-canonical mode if the trigger  
2747 value for sending IXOFF is less than VMIN and VTIME is zero.

2748 A.11.1.7 *Non-Canonical Mode Input Processing*

2749 Some points to note about MIN and TIME:

- 2750 1. The interactions of MIN and TIME are not symmetric. For example, when MIN>0 and  
2751 TIME=0, TIME has no effect. However, in the opposite case where MIN=0 and TIME>0,  
2752 both MIN and TIME play a role in that MIN is satisfied with the receipt of a single  
2753 character.
- 2754 2. Also note that in case A (MIN>0, TIME>0), TIME represents an inter-character timer, while  
2755 in case C (MIN=0, TIME>0), TIME represents a read timer.

2756 These two points highlight the dual purpose of the MIN/TIME feature. Cases A and B, where  
2757 MIN>0, exist to handle burst-mode activity (for example, file transfer programs) where a  
2758 program would like to process at least MIN characters at a time. In case A, the inter-character  
2759 timer is activated by a user as a safety measure; in case B, it is turned off.

2760 Cases C and D exist to handle single-character timed transfers. These cases are readily adaptable  
2761 to screen-based applications that need to know if a character is present in the input queue before  
2762 refreshing the screen. In case C, the read is timed; in case D, it is not.

2763 Another important note is that MIN is always just a minimum. It does not denote a record  
2764 length. That is, if a program does a read of 20 bytes, MIN is 10, and 25 characters are present, 20  
2765 characters shall be returned to the user. In the special case of MIN=0, this still applies: if more  
2766 than one character is available, they all will be returned immediately.

2767 **A.11.1.8 Writing Data and Output Processing**

2768 There is no additional rationale provided for this section.

2769 **A.11.1.9 Special Characters**

2770 There is no additional rationale provided for this section.

2771 **A.11.1.10 Modem Disconnect**

2772 There is no additional rationale provided for this section.

2773 **A.11.1.11 Closing a Terminal Device File**

2774 IEEE Std 1003.1-200x does not specify that a *close()* on a terminal device file include the  
2775 equivalent of a call to *tcfow(fd,TCOON)*.

2776 An implementation that discards output at the time *close()* is called after reporting the return  
2777 value to the *write()* call that data was written does not conform with IEEE Std 1003.1-200x. An  
2778 application has functions such as *tcdrain()*, *tcfush()*, and *tcfow()* available to obtain the detailed  
2779 behavior it requires with respect to flushing of output.

2780 At the time of the last close on a terminal device, an application relinquishes any ability to exert  
2781 flow control via *tcfow()*.

2782 **A.11.2 Parameters that Can be Set**2783 **A.11.2.1 The termios Structure**

2784 This structure is part of an interface that, in general, retains the historic grouping of flags.  
2785 Although a more optimal structure for implementations may be possible, the degree of change  
2786 to applications would be significantly larger.

2787 **A.11.2.2 Input Modes**

2788 Some historical implementations treated a long break as multiple events, as many as one per  
2789 character time. The wording in POSIX.1 explicitly prohibits this.

2790 Although the ISTRIP flag is normally superfluous with today's terminal hardware and software,  
2791 it is historically supported. Therefore, applications may be using ISTRIP, and there is no  
2792 technical problem with supporting this flag. Also, applications may wish to receive only 7-bit  
2793 input bytes and may not be connected directly to the hardware terminal device (for example,  
2794 when a connection traverses a network).

2795 Also, there is no requirement in general that the terminal device ensures that high-order bits  
2796 beyond the specified character size are cleared. ISTRIP provides this function for 7-bit  
2797 characters, which are common.

2798 In dealing with multi-byte characters, the consequences of a parity error in such a character, or in  
2799 an escape sequence affecting the current character set, are beyond the scope of POSIX.1 and are  
2800 best dealt with by the application processing the multi-byte characters.

2801 *A.11.2.3 Output Modes*

2802 POSIX.1 does not describe postprocessing of output to a terminal or detailed control of that from |  
2803 a conforming application. (That is, translation of <newline> to <carriage-return> followed by |  
2804 <linefeed> or <tab> processing.) There is nothing that a conforming application should do to its |  
2805 output for a terminal because that would require knowledge of the operation of the terminal. It  
2806 is the responsibility of the operating system to provide postprocessing appropriate to the output  
2807 device, whether it is a terminal or some other type of device.

2808 Extensions to POSIX.1 to control the type of postprocessing already exist and are expected to  
2809 continue into the future. The control of these features is primarily to adjust the interface between  
2810 the system and the terminal device so the output appears on the display correctly. This should  
2811 be set up before use by any application.

2812 In general, both the input and output modes should not be set absolutely, but rather modified  
2813 from the inherited state.

2814 *A.11.2.4 Control Modes*

2815 This section could be misread that the symbol “CSIZE” is a title in the **termios** *c\_cflag* field .  
2816 Although it does serve that function, it is also a required symbol, as a literal reading of POSIX.1  
2817 (and the caveats about typography) would indicate.

2818 *A.11.2.5 Local Modes*

2819 Non-canonical mode is provided to allow fast bursts of input to be read efficiently while still  
2820 allowing single-character input.

2821 The ECHONL function historically has been in many implementations. Since there seems to be  
2822 no technical problem with supporting ECHONL, it is included in POSIX.1 to increase consensus.

2823 The alternate behavior possible when ECHOK or ECHOE are specified with ICANON is  
2824 permitted as a compromise depending on what the actual terminal hardware can do. Erasing  
2825 characters and lines is preferred, but is not always possible.

2826 *A.11.2.6 Special Control Characters*

2827 Permitting VMIN and VTIME to overlap with VEOF and VEOL was a compromise for historical  
2828 implementations. Only when backwards-compatibility of object code is a serious concern to an  
2829 implementor should an implementation continue this practice. Correct applications that work  
2830 with the overlap (at the source level) should also work if it is not present, but not the reverse.

2831 **A.12 Utility Conventions**2832 **A.12.1 Utility Argument Syntax**

2833 The standard developers considered that recent trends toward diluting the SYNOPSIS sections  
2834 of historical reference pages to the equivalent of:

2835 `command [options][operands]`

2836 were a disservice to the reader. Therefore, considerable effort was placed into rigorous  
2837 definitions of all the command line arguments and their interrelationships. The relationships  
2838 depicted in the synopses are normative parts of IEEE Std 1003.1-200x; this information is  
2839 sometimes repeated in textual form, but that is only for clarity within context.

2840 The use of “undefined” for conflicting argument usage and for repeated usage of the same |  
2841 option is meant to prevent conforming applications from using conflicting arguments or |  
2842 repeated options unless specifically allowed (as is the case with *ls*, which allows simultaneous,  
2843 repeated use of the *-C*, *-l*, and *-1* options). Many historical implementations will tolerate this  
2844 usage, choosing either the first or the last applicable argument. This tolerance can continue, but |  
2845 conforming applications cannot rely upon it. (Other implementations may choose to print usage |  
2846 messages instead.)

2847 The use of “undefined” for conflicting argument usage also allows an implementation to make  
2848 reasonable extensions to utilities where the implementor considers mutually-exclusive options  
2849 according to IEEE Std 1003.1-200x to have a sensible meaning and result.

2850 IEEE Std 1003.1-200x does not define the result of a command when an option-argument or  
2851 operand is not followed by ellipses and the application specifies more than one of that option-  
2852 argument or operand. This allows an implementation to define valid (although non-standard)  
2853 behavior for the utility when more than one such option or operand is specified.

2854 Allowing <blank>s after an option (that is, placing an option and its option-argument into  
2855 separate argument strings) when IEEE Std 1003.1-200x does not require it encourages portability  
2856 of users, while still preserving backwards-compatibility of scripts. Inserting <blank>s between  
2857 the option and the option-argument is preferred; however, historical usage has not been  
2858 consistent in this area; therefore, <blank>s are required to be handled by all implementations,  
2859 but implementations are also allowed to handle the historical syntax. Another justification for  
2860 selecting the multiple-argument method was that the single-argument case is inherently  
2861 ambiguous when the option-argument can legitimately be a null string.

2862 IEEE Std 1003.1-200x explicitly states that digits are permitted as operands and option-  
2863 arguments. The lower and upper bounds for the values of the numbers used for operands and  
2864 option-arguments were derived from the ISO C standard values for {LONG\_MIN} and  
2865 {LONG\_MAX}. The requirement on the standard utilities is that numbers in the specified range  
2866 do not cause a syntax error, although the specification of a number need not be semantically  
2867 correct for a particular operand or option-argument of a utility. For example, the specification of:

2868 `dd obs=3000000000`

2869 would yield undefined behavior for the application and could be a syntax error because the  
2870 number 3 000 000 000 is outside of the range *-2 147 483 647* to *+2 147 483 647*. On the other hand:

2871 `dd obs=2000000000`

2872 may cause some error, such as “blocksize too large”, rather than a syntax error.

2873 **A.12.2 Utility Syntax Guidelines**

2874 This section is based on the rules listed in the SVID. It was included for two reasons:

- 2875 1. The individual utility descriptions in the Shell and Utilities volume of  
2876 IEEE Std 1003.1-200x, Chapter 4, Utilities needed a set of common (although not universal)  
2877 actions on which they could anchor their descriptions of option and operand syntax. Most  
2878 of the standard utilities actually do use these guidelines, and many of their historical  
2879 implementations use the *getopt()* function for their parsing. Therefore, it was simpler to  
2880 cite the rules and merely identify exceptions.
- 2881 2. Writers of conforming applications need suggested guidelines if the POSIX community is  
2882 to avoid the chaos of historical UNIX system command syntax.

2883 It is recommended that all *future* utilities and applications use these guidelines to enhance “user  
2884 portability”. The fact that some historical utilities could not be changed (to avoid breaking  
2885 historical applications) should not deter this future goal.

2886 The voluntary nature of the guidelines is highlighted by repeated uses of the word *should*  
2887 throughout. This usage should not be misinterpreted to imply that utilities that claim  
2888 conformance in their OPTIONS sections do not always conform.

2889 Guidelines 1 and 2 are offered as guidance for locales using Latin alphabets. No  
2890 recommendations are made by IEEE Std 1003.1-200x concerning utility naming in other locales.

2891 In the Shell and Utilities volume of IEEE Std 1003.1-200x, Section 2.9.1, Simple Commands, it is  
2892 further stated that a command used in the Shell Command Language cannot be named with a  
2893 trailing colon.

2894 Guideline 3 was changed to allow alphanumeric characters (letters and digits) from the character  
2895 set to allow compatibility with historical usage. Historical practice allows the use of digits  
2896 wherever practical, and there are no portability issues that would prohibit the use of digits. In  
2897 fact, from an internationalization viewpoint, digits (being non-language-dependent) are  
2898 preferable over letters (a *-2* is intuitively self-explanatory to any user, while in the *-f filename* the  
2899 letter ‘*f*’ is a mnemonic aid only to speakers of Latin-based languages where “filename”  
2900 happens to translate to a word that begins with ‘*f*’). Since guideline 3 still retains the word  
2901 “single”, multi-digit options are not allowed. Instances of historical utilities that used them have  
2902 been marked obsolescent, with the numbers being changed from option names to option-  
2903 arguments.

2904 It was difficult to achieve a satisfactory solution to the problem of name space in option  
2905 characters. When the standard developers desired to extend the historical *cc* utility to accept  
2906 ISO C standard programs, they found that all of the portable alphabet was already in use by  
2907 various vendors. Thus, they had to devise a new name, *c89*, rather than something like *cc -X*.  
2908 There were suggestions that implementors be restricted to providing extensions through various  
2909 means (such as using a plus sign as the option delimiter or using option characters outside the  
2910 alphanumeric set) that would reserve all of the remaining alphanumeric characters for future  
2911 POSIX standards. These approaches were resisted because they lacked the historical style of  
2912 UNIX systems. Furthermore, if a vendor-provided option should become commonly used in the  
2913 industry, it would be a candidate for standardization. It would be desirable to standardize such a  
2914 feature using historical practice for the syntax (the semantics can be standardized with any  
2915 syntax). This would not be possible if the syntax was one reserved for the vendor. However,  
2916 since the standardization process may lead to minor changes in the semantics, it may prove to be  
2917 better for a vendor to use a syntax that will not be affected by standardization.

2918 Guideline 8 includes the concept of comma-separated lists in a single argument. It is up to the  
2919 utility to parse such a list itself because *getopt()* just returns the single string. This situation was

2920 retained so that certain historical utilities would not violate the guidelines. Applications  
2921 preparing for international use should be aware of an occasional problem with comma-  
2922 separated lists: in some locales, the comma is used as the radix character. Thus, if an application  
2923 is preparing operands for a utility that expects a comma-separated lists, it should avoid  
2924 generating non-integer values through one of the means that is influenced by setting the  
2925 *LC\_NUMERIC* variable (such as *awk*, *bc*, *printf*, or *printf()*).

2926 Applications calling any utility with a first operand starting with '-' should usually specify --,  
2927 as indicated by Guideline 10, to mark the end of the options. This is true even if the SYNOPSIS in  
2928 the Shell and Utilities volume of IEEE Std 1003.1-200x does not specify any options;  
2929 implementations may provide options as extensions to the Shell and Utilities volume of  
2930 IEEE Std 1003.1-200x. The standard utilities that do not support Guideline 10 indicate that fact in  
2931 the OPTIONS section of the utility description.

2932 Guideline 11 was modified to clarify that the order of different options should not matter  
2933 relative to one another. However, the order of repeated options that also have option-arguments  
2934 may be significant; therefore, such options are required to be interpreted in the order that they  
2935 are specified. The *make* utility is an instance of a historical utility that uses repeated options  
2936 in which the order is significant. Multiple files are specified by giving multiple instances of the -f  
2937 option; for example:

```
2938     make -f common_header -f specific_rules target
```

2939 Guideline 13 does not imply that all of the standard utilities automatically accept the operand  
2940 '-' to mean standard input or output, nor does it specify the actions of the utility upon  
2941 encountering multiple '-' operands. It simply says that, by default, '-' operands are not used  
2942 for other purposes in the file reading or writing (but not when using *stat*, *unlink*, *touch*, and so on)  
2943 utilities. All information concerning actual treatment of the '-' operand is found in the  
2944 individual utility sections.

2945 An area of concern was that as implementations mature, implementation-defined utilities and  
2946 implementation-defined utility options will result. The idea was expressed that there needed to  
2947 be a standard way, say an environment variable or some such mechanism, to identify  
2948 implementation-defined utilities separately from standard utilities that may have the same  
2949 name. It was decided that there already exist several ways of dealing with this situation and that  
2950 it is outside of the POSIX.2 scope to attempt to standardize in the area of non-standard items. A  
2951 method that exists on some historical implementations is the use of the so-called */local/bin* or  
2952 */usr/local/bin* directory to separate local or additional copies or versions of utilities. Another  
2953 method that is also used is to isolate utilities into completely separate domains. Still another  
2954 method to ensure that the desired utility is being used is to request the utility by its full  
2955 pathname. There are many approaches to this situation; the examples given above serve to  
2956 illustrate that there is more than one.

2957 **A.13 Headers**2958 **A.13.1 Format of Entries**

2959 Each header reference page has a common layout of sections describing the interface. This layout  
2960 is similar to the manual page or “man” page format shipped with most UNIX systems, and each  
2961 header has sections describing the SYNOPSIS and DESCRIPTION. These are the two sections  
2962 that relate to conformance.

2963 Additional sections are informative, and add considerable information for the application  
2964 developer. APPLICATION USAGE sections provide additional caveats, issues, and  
2965 recommendations to the developer. RATIONALE sections give additional information on the  
2966 decisions made in defining the interface.

2967 FUTURE DIRECTIONS sections act as pointers to related work that may impact the interface in  
2968 the future, and often cautions the developer to architect the code to account for a change in this  
2969 area. Note that a future directions statement should not be taken as a commitment to adopt a  
2970 feature or interface in the future.

2971 The CHANGE HISTORY section describes when the interface was introduced, and how it has  
2972 changed.

2973 Option labels and margin markings in the page can be useful in guiding the application |  
2974 developer.

2975 / *Rationale (Informative)*

2976 **Part B:**

2977 **System Interfaces**

2978 *The Open Group*  
2979 *The Institute of Electrical and Electronics Engineers, Inc.*



# Rationale for System Interfaces

2980

## 2981 **B.1 Introduction**

### 2982 **B.1.1 Scope**

2983 Refer to Section A.1.1 (on page 3293).

### 2984 **B.1.2 Conformance**

2985 Refer to Section A.2 (on page 3299).

### 2986 **B.1.3 Normative References**

2987 There is no additional rationale provided for this section. |

### 2988 **B.1.4 Change History**

2989 The change history is provided as an informative section, to track changes from previous issues |  
 2990 of IEEE Std 1003.1-200x. |

2991 The following sections describe changes made to the System Interfaces volume of |  
 2992 IEEE Std 1003.1-200x since Issue 5 of the base document. The CHANGE HISTORY section for |  
 2993 each entry details the technical changes that have been made to that entry since Issue 5. |  
 2994 Changes between earlier issues of the base document and Issue 5 are not included. |

2995 The change history between Issue 5 and Issue 6 also lists the changes since the |  
 2996 ISO POSIX-1: 1996 standard. |

#### 2997 **Changes from Issue 5 to Issue 6 (IEEE Std 1003.1-200x)**

2998 The following list summarizes the major changes that were made in the System Interfaces |  
 2999 volume of IEEE Std 1003.1-200x from Issue 5 to Issue 6: |

- 3000 • This volume of IEEE Std 1003.1-200x is extensively revised so it can be both an IEEE POSIX |  
 3001 Standard and an Open Group Technical Standard. |
- 3002 • The POSIX System Interfaces requirements incorporate support of FIPS 151-2. |
- 3003 • The POSIX System Interfaces requirements are updated to align with some features of the |  
 3004 Single UNIX Specification. |
- 3005 • A RATIONALE section is added to each reference page. |
- 3006 • Networking interfaces from the XNS, Issue 5.2 specification are incorporated. |
- 3007 • IEEE Std 1003.1d-1999 is incorporated. |
- 3008 • IEEE Std 1003.1j-2000 is incorporated. |
- 3009 • IEEE Std 1003.1q-2000 is incorporated. |
- 3010 • IEEE P1003.1a draft standard is incorporated. |

- 3011 • Existing functionality is aligned with the ISO/IEC 9899: 1999 standard.
- 3012 • New functionality from the ISO/IEC 9899: 1999 standard is incorporated.
- 3013 • IEEE PASC Interpretations are applied.
- 3014 • The Open Group corrigenda and resolutions are applied.

### 3015 **New Features in Issue 6**

3016 The functions first introduced in Issue 6 (over the Issue 5 Base document) are listed in the table  
 3017 below:

3018  
 3019

3020  
 3021  
 3022  
 3023  
 3024  
 3025  
 3026  
 3027  
 3028  
 3029  
 3030  
 3031  
 3032  
 3033  
 3034  
 3035  
 3036  
 3037  
 3038  
 3039  
 3040  
 3041  
 3042  
 3043  
 3044  
 3045  
 3046  
 3047  
 3048  
 3049  
 3050  
 3051  
 3052  
 3053  
 3054  
 3055  
 3056

<b>New Functions in Issue 6</b>		
<i>acosf()</i>	<i>catanh()</i>	<i>cprojf()</i>
<i>acoshf()</i>	<i>catanl()</i>	<i>cprojl()</i>
<i>acoshl()</i>	<i>cbtrf()</i>	<i>creal()</i>
<i>acosl()</i>	<i>cbtrl()</i>	<i>crealf()</i>
<i>asinf()</i>	<i>ccos()</i>	<i>creall()</i>
<i>asinhf()</i>	<i>ccosf()</i>	<i>csin()</i>
<i>asinhl()</i>	<i>ccosh()</i>	<i>csinf()</i>
<i>asinl()</i>	<i>ccoshf()</i>	<i>csinh()</i>
<i>atan2f()</i>	<i>ccoshl()</i>	<i>csinhf()</i>
<i>atan2l()</i>	<i>ccosl()</i>	<i>csinhl()</i>
<i>atanf()</i>	<i>ceilf()</i>	<i>csinl()</i>
<i>atanhf()</i>	<i>ceil()</i>	<i>csqrt()</i>
<i>atanhl()</i>	<i>cexp()</i>	<i>csqrtf()</i>
<i>atanl()</i>	<i>cexpf()</i>	<i>csqrtl()</i>
<i>atoll()</i>	<i>cexpl()</i>	<i>ctan()</i>
<i>cabs()</i>	<i>cimag()</i>	<i>ctanf()</i>
<i>cabsf()</i>	<i>cimagf()</i>	<i>ctanh()</i>
<i>cabsl()</i>	<i>cimagl()</i>	<i>ctanhf()</i>
<i>cacos()</i>	<i>clock_getcpuclockid()</i>	<i>ctanhl()</i>
<i>cacosf()</i>	<i>clock_nanosleep()</i>	<i>ctanl()</i>
<i>cacosh()</i>	<i>clog()</i>	<i>erfcf()</i>
<i>cacoshf()</i>	<i>clogf()</i>	<i>erfcl()</i>
<i>cacoshl()</i>	<i>clogl()</i>	<i>erff()</i>
<i>cacosl()</i>	<i>conj()</i>	<i>erfl()</i>
<i>carg()</i>	<i>conjf()</i>	<i>exp2()</i>
<i>cargf()</i>	<i>conjl()</i>	<i>exp2f()</i>
<i>cargl()</i>	<i>copysign()</i>	<i>exp2l()</i>
<i>casin()</i>	<i>copysignf()</i>	<i>expf()</i>
<i>casinf()</i>	<i>copysignl()</i>	<i>expl()</i>
<i>casinh()</i>	<i>cosf()</i>	<i>expm1f()</i>
<i>casinhf()</i>	<i>coshf()</i>	<i>expm1l()</i>
<i>casinhl()</i>	<i>coshl()</i>	<i>fabsf()</i>
<i>casinl()</i>	<i>cosl()</i>	<i>fabsl()</i>
<i>catan()</i>	<i>cpow()</i>	<i>fdim()</i>
<i>catanf()</i>	<i>cpowf()</i>	<i>fdimf()</i>
<i>catanh()</i>	<i>cpowl()</i>	<i>fdiml()</i>
<i>catanhf()</i>	<i>cproj()</i>	<i>feclearexcept()</i>

3057  
3058  
3059  
3060  
3061  
3062  
3063  
3064  
3065  
3066  
3067  
3068  
3069  
3070  
3071  
3072  
3073  
3074  
3075  
3076  
3077  
3078  
3079  
3080  
3081  
3082  
3083  
3084  
3085  
3086  
3087  
3088  
3089  
3090  
3091  
3092  
3093  
3094  
3095  
3096  
3097  
3098  
3099  
3100  
3101  
3102

**New Functions in Issue 6**

<i>fegetenv()</i>	<i>ldexpl()</i>	<i>posix_fallocate()</i>
<i>fegetexceptflag()</i>	<i>lgammaf()</i>	<i>posix_madvise()</i>
<i>fegetround()</i>	<i>lgammal()</i>	<i>posix_mem_offset()</i>
<i>fehldexcept()</i>	<i>llabs()</i>	<i>posix_memalign()</i>
<i>feraiseexcept()</i>	<i>lldiv()</i>	<i>posix_openpt()</i>
<i>fesetenv()</i>	<i>llrint()</i>	<i>posix_spawn()</i>
<i>fesetexceptflag()</i>	<i>llrintf()</i>	<i>posix_spawn_file_actions_addclose()</i>
<i>fesetround()</i>	<i>llrintl()</i>	<i>posix_spawn_file_actions_adddup2()</i>
<i>fetestexcept()</i>	<i>llround()</i>	<i>posix_spawn_file_actions_addopen()</i>
<i>feupdateenv()</i>	<i>llroundf()</i>	<i>posix_spawn_file_actions_destroy()</i>
<i>floorf()</i>	<i>llroundl()</i>	<i>posix_spawn_file_actions_init()</i>
<i>floorl()</i>	<i>log10f()</i>	<i>posix_spawnattr_destroy()</i>
<i>fna()</i>	<i>log10l()</i>	<i>posix_spawnattr_getflags()</i>
<i>fnaf()</i>	<i>log1pf()</i>	<i>posix_spawnattr_getpgroup()</i>
<i>fnal()</i>	<i>log1pl()</i>	<i>posix_spawnattr_getschedparam()</i>
<i>fnax()</i>	<i>log2()</i>	<i>posix_spawnattr_getschedpolicy()</i>
<i>fnaxf()</i>	<i>log2f()</i>	<i>posix_spawnattr_getsigdefault()</i>
<i>fnaxl()</i>	<i>log2l()</i>	<i>posix_spawnattr_getsigmask()</i>
<i>fnin()</i>	<i>logbf()</i>	<i>posix_spawnattr_init()</i>
<i>fninf()</i>	<i>logbl()</i>	<i>posix_spawnattr_setflags()</i>
<i>fninl()</i>	<i>logf()</i>	<i>posix_spawnattr_setpgroup()</i>
<i>fnodf()</i>	<i>logl()</i>	<i>posix_spawnattr_setschedparam()</i>
<i>fnodl()</i>	<i>lrint()</i>	<i>posix_spawnattr_setschedpolicy()</i>
<i>fpclassify()</i>	<i>lrintf()</i>	<i>posix_spawnattr_setsigdefault()</i>
<i>frexpf()</i>	<i>lrintl()</i>	<i>posix_spawnattr_setsigmask()</i>
<i>frexpl()</i>	<i>lround()</i>	<i>posix_spawnnp()</i>
<i>hypotf()</i>	<i>lroundf()</i>	<i>posix_trace_attr_destroy()</i>
<i>hypotl()</i>	<i>lroundl()</i>	<i>posix_trace_attr_getclockres()</i>
<i>ilogbf()</i>	<i>modff()</i>	<i>posix_trace_attr_getcreatetime()</i>
<i>ilogbl()</i>	<i>modfl()</i>	<i>posix_trace_attr_getgenversion()</i>
<i>imaxabs()</i>	<i>mq_timedreceive()</i>	<i>posix_trace_attr_getinherited()</i>
<i>imaxdiv()</i>	<i>mq_timedsend()</i>	<i>posix_trace_attr_getlogfullpolicy()</i>
<i>isblank()</i>	<i>nan()</i>	<i>posix_trace_attr_getlogsize()</i>
<i>isfinite()</i>	<i>nanf()</i>	<i>posix_trace_attr_getmaxdatasize()</i>
<i>isgreater()</i>	<i>nanl()</i>	<i>posix_trace_attr_getmaxsystemeventsz()</i>
<i>isgreaterequal()</i>	<i>nearbyint()</i>	<i>posix_trace_attr_getmaxuserereventsiz()</i>
<i>isinf()</i>	<i>nearbyintf()</i>	<i>posix_trace_attr_getname()</i>
<i>isless()</i>	<i>nearbyintl()</i>	<i>posix_trace_attr_getstreamfullpolicy()</i>
<i>islessequal()</i>	<i>nextafterf()</i>	<i>posix_trace_attr_getstreamsize()</i>
<i>islessgreater()</i>	<i>nextafterl()</i>	<i>posix_trace_attr_init()</i>
<i>isnormal()</i>	<i>nexttoward()</i>	<i>posix_trace_attr_setinherited()</i>
<i>isunordered()</i>	<i>nexttowardf()</i>	<i>posix_trace_attr_setlogfullpolicy()</i>
<i>iswblank()</i>	<i>nexttowardl()</i>	<i>posix_trace_attr_setlogsize()</i>
<i>ldexpf()</i>	<i>posix_fadvise()</i>	<i>posix_trace_create()</i>

3103  
3104  
3105  
3106  
3107  
3108  
3109  
3110  
3111  
3112  
3113  
3114  
3115  
3116  
3117  
3118  
3119  
3120  
3121  
3122  
3123  
3124  
3125  
3126  
3127  
3128  
3129  
3130  
3131  
3132  
3133  
3134  
3135  
3136  
3137  
3138  
3139  
3140  
3141  
3142  
3143

**New Functions in Issue 6**

<i>posix_trace_attr_setmaxdatasize()</i>	<i>pthread_barrier_destroy()</i>	<i>signbit()</i>
<i>posix_trace_attr_setname()</i>	<i>pthread_barrier_init()</i>	<i>sinf()</i>
<i>posix_trace_attr_setstreamfullpolicy()</i>	<i>pthread_barrier_wait()</i>	<i>sinhf()</i>
<i>posix_trace_attr_setstreamsize()</i>	<i>pthread_barrierattr_destroy()</i>	<i>sinhl()</i>
<i>posix_trace_clear()</i>	<i>pthread_barrierattr_getpshared()</i>	<i>sinl()</i>
<i>posix_trace_close()</i>	<i>pthread_barrierattr_init()</i>	<i>sqrtf()</i>
<i>posix_trace_create_withlog()</i>	<i>pthread_barrierattr_setpshared()</i>	<i>sqrtl()</i>
<i>posix_trace_event()</i>	<i>pthread_condattr_getclock()</i>	<i>strerror_r()</i>
<i>posix_trace_eventid_equal()</i>	<i>pthread_condattr_setclock()</i>	<i>stroull()</i>
<i>posix_trace_eventid_get_name()</i>	<i>pthread_getcpuclockid()</i>	<i>strtoimax()</i>
<i>posix_trace_eventid_open()</i>	<i>pthread_mutex_timedlock()</i>	<i>strtoll()</i>
<i>posix_trace_eventset_add()</i>	<i>pthread_rwlock_timedrdlock()</i>	<i>strtoumax()</i>
<i>posix_trace_eventset_del()</i>	<i>pthread_rwlock_timedwrlock()</i>	<i>tanf()</i>
<i>posix_trace_eventset_empty()</i>	<i>pthread_schedsetprio()</i>	<i>tanhf()</i>
<i>posix_trace_eventset_fill()</i>	<i>pthread_spin_destroy()</i>	<i>tanh1()</i>
<i>posix_trace_eventset_ismember()</i>	<i>pthread_spin_init()</i>	<i>tanl()</i>
<i>posix_trace_eventtypelist_getnext_id()</i>	<i>pthread_spin_lock()</i>	<i>tgamma()</i>
<i>posix_trace_eventtypelist_rewind()</i>	<i>pthread_spin_trylock()</i>	<i>tgammaf()</i>
<i>posix_trace_flush()</i>	<i>pthread_spin_unlock()</i>	<i>tgammal()</i>
<i>posix_trace_get_attr()</i>	<i>remainderf()</i>	<i>trunc()</i>
<i>posix_trace_get_filter()</i>	<i>remainderl()</i>	<i>truncf()</i>
<i>posix_trace_get_status()</i>	<i>remquo()</i>	<i>truncl()</i>
<i>posix_trace_getnext_event()</i>	<i>remquof()</i>	<i>unsetenv()</i>
<i>posix_trace_open()</i>	<i>remquol()</i>	<i>vfprintf()</i>
<i>posix_trace_rewind()</i>	<i>rintf()</i>	<i>vfscanf()</i>
<i>posix_trace_set_filter()</i>	<i>rintl()</i>	<i>vwscanf()</i>
<i>posix_trace_shutdown()</i>	<i>round()</i>	<i>vprintf()</i>
<i>posix_trace_start()</i>	<i>roundf()</i>	<i>vscanf()</i>
<i>posix_trace_stop()</i>	<i>roundl()</i>	<i>vsnprintf()</i>
<i>posix_trace_timedgetnext_event()</i>	<i>scalbln()</i>	<i>vsprintf()</i>
<i>posix_trace_trid_eventid_open()</i>	<i>scalblnf()</i>	<i>vsscanf()</i>
<i>posix_trace_trygetnext_event()</i>	<i>scalblnl()</i>	<i>vswscanf()</i>
<i>posix_typed_mem_get_info()</i>	<i>scalbn()</i>	<i>vwscanf()</i>
<i>posix_typed_mem_open()</i>	<i>scalbnf()</i>	<i>wcstoimax()</i>
<i>powf()</i>	<i>scalbnl()</i>	<i>wcstoll()</i>
<i>powl()</i>	<i>sem_timedwait()</i>	<i>wcstoull()</i>
<i>pselect()</i>	<i>setegid()</i>	<i>wcstoumax()</i>
<i>pthread_attr_getstack()</i>	<i>setenv()</i>	
<i>pthread_attr_setstack()</i>	<i>seteuid()</i>	

3144 The following new headers are introduced in Issue 6:

3145

3146

3147

3148

3149

New Headers in Issue 6		
<complex.h>	<spawn.h>	<tgmath.h>
<fenv.h>	<stdbool.h>	<trace.h>
<net/if.h>	<stdint.h>	

3150 The following table lists the functions and symbols from the XSI extension. These are new since  
 3151 the ISO POSIX-1: 1996 standard.

3152  
 3153

New XSI Functions and Symbols in Issue 6				
3154	<i>_longjmp()</i>	<i>getcontext()</i>	<i>msgget()</i>	<i>setutxent()</i>
3155	<i>_setjmp()</i>	<i>getdate()</i>	<i>msgrcv()</i>	<i>shmat()</i>
3156	<i>_tolower()</i>	<i>getgrent()</i>	<i>msgsnd()</i>	<i>shmctl()</i>
3157	<i>_toupper()</i>	<i>gethostid()</i>	<i>nftw()</i>	<i>shmdt()</i>
3158	<i>a64l()</i>	<i>getitimer()</i>	<i>nice()</i>	<i>shmget()</i>
3159	<i>basename()</i>	<i>getpgid()</i>	<i>nl_langinfo()</i>	<i>sigaltstack()</i>
3160	<i>bcmp()</i>	<i>getpmsg()</i>	<i>rand48()</i>	<i>sighold()</i>
3161	<i>bcopy()</i>	<i>getpriority()</i>	<i>openlog()</i>	<i>sigignore()</i>
3162	<i>bzero()</i>	<i>getpwent()</i>	<i>poll()</i>	<i>siginterrupt()</i>
3163	<i>catclose()</i>	<i>getrlimit()</i>	<i>pread()</i>	<i>sigpause()</i>
3164	<i>catgets()</i>	<i>getrusage()</i>	<i>pthread_attr_getguardsize()</i>	<i>sigrelse()</i>
3165	<i>catopen()</i>	<i>getsid()</i>	<i>pthread_attr_setguardsize()</i>	<i>sigset()</i>
3166	<i>closelog()</i>	<i>getsubopt()</i>	<i>pthread_attr_setstack()</i>	<i>srand48()</i>
3167	<i>crypt()</i>	<i>gettimeofday()</i>	<i>pthread_getconcurrency()</i>	<i>srandom()</i>
3168	<i>daylight</i>	<i>getutxent()</i>	<i>pthread_mutexattr_gettype()</i>	<i>statvfs()</i>
3169	<i>dbm_clearerr()</i>	<i>getutxid()</i>	<i>pthread_mutexattr_settype()</i>	<i>strcasecmp()</i>
3170	<i>dbm_close()</i>	<i>getutxline()</i>	<i>pthread_rwlockattr_init()</i>	<i>strdup()</i>
3171	<i>dbm_delete()</i>	<i>getwd()</i>	<i>pthread_rwlockattr_setpshared()</i>	<i>strfmon()</i>
3172	<i>dbm_error()</i>	<i>grantpt()</i>	<i>pthread_setconcurrency()</i>	<i>strncasecmp()</i>
3173	<i>dbm_fetch()</i>	<i>hcreate()</i>	<i>ptsname()</i>	<i>strptime()</i>
3174	<i>dbm_firstkey()</i>	<i>hdestroy()</i>	<i>putenv()</i>	<i>swab()</i>
3175	<i>dbm_nextkey()</i>	<i>hsearch()</i>	<i>pututxline()</i>	<i>swapcontext()</i>
3176	<i>dbm_open()</i>	<i>iconv()</i>	<i>pwrite()</i>	<i>sync()</i>
3177	<i>dbm_store()</i>	<i>iconv_close()</i>	<i>random()</i>	<i>syslog()</i>
3178	<i>dirname()</i>	<i>iconv_open()</i>	<i>readv()</i>	<i>tcgetsid()</i>
3179	<i>dlclose()</i>	<i>index()</i>	<i>realpath()</i>	<i>tdelete()</i>
3180	<i>dlderror()</i>	<i>initstate()</i>	<i>remque()</i>	<i>telldir()</i>
3181	<i>dlopen()</i>	<i>insque()</i>	<i>rindex()</i>	<i>tempnam()</i>
3182	<i>dlsym()</i>	<i>isascii()</i>	<i>seed48()</i>	<i>tfind()</i>
3183	<i>drand48()</i>	<i>jrand48()</i>	<i>seekdir()</i>	<i>timezone</i>
3184	<i>ecvt()</i>	<i>killpg()</i>	<i>semctl()</i>	<i>toascii()</i>
3185	<i>encrypt()</i>	<i>l64a()</i>	<i>semget()</i>	<i>truncate()</i>
3186	<i>endgrent()</i>	<i>lchown()</i>	<i>semop()</i>	<i>tsearch()</i>
3187	<i>endpwent()</i>	<i>lcong48()</i>	<i>setcontext()</i>	<i>twalk()</i>
3188	<i>endutxent()</i>	<i>lfind()</i>	<i>setgrent()</i>	<i>ulimit()</i>
3189	<i>erand48()</i>	<i>lockf()</i>	<i>setitimer()</i>	<i>unlockpt()</i>
3190	<i>fchdir()</i>	<i>lrand48()</i>	<i>setkey()</i>	<i>utimes()</i>
3191	<i>fcvt()</i>	<i>lsearch()</i>	<i>setlogmask()</i>	<i>waitid()</i>
3192	<i>ffs()</i>	<i>makecontext()</i>	<i>setpggrp()</i>	<i>wcswcs()</i>
3193	<i>fntmsg()</i>	<i>memccpy()</i>	<i>setpriority()</i>	<i>wcswidth()</i>
3194	<i>fstatvfs()</i>	<i>mknod()</i>	<i>setpwent()</i>	<i>wcwidth()</i>
3195	<i>ftime()</i>	<i>mkstemp()</i>	<i>setregid()</i>	<i>writev()</i>
3196	<i>ftok()</i>	<i>mktemp()</i>	<i>setreuid()</i>	
3197	<i>ftw()</i>	<i>mrnd48()</i>	<i>setrlimit()</i>	
3198	<i>gcvt()</i>	<i>msgctl()</i>	<i>setstate()</i>	

3199 The following table lists the headers from the XSI extension. These are new since the  
 3200 ISO POSIX-1: 1996 standard.

3201  
3202  
3203  
3204  
3205  
3206  
3207  
3208  
3209  
3210  
3211  
3212

New XSI Headers in Issue 6		
<cpio.h>	<poll.h>	<sys/statvfs.h>
<dlfcn.h>	<search.h>	<sys/time.h>
<fmtmsg.h>	<strings.h>	<sys/timeb.h>
<ftw.h>	<stropts.h>	<sys/uio.h>
<iconv.h>	<sys/ipc.h>	<syslog.h>
<langinfo.h>	<sys/mman.h>	<ucontext.h>
<libgen.h>	<sys/msg.h>	<ulimit.h>
<monetary.h>	<sys/resource.h>	<utmpx.h>
<ndbm.h>	<sys/sem.h>	
<nl_types.h>	<sys/shm.h>	

### 3213 **B.1.5 Terminology**

3214 Refer to Section A.1.4 (on page 3295).

### 3215 **B.1.6 Definitions**

3216 Refer to Section A.3 (on page 3302).

### 3217 **B.1.7 Relationship to Other Formal Standards**

3218 There is no additional rationale provided for this section.

### 3219 **B.1.8 Portability**

3220 Refer to Section A.1.5 (on page 3298).

#### 3221 *B.1.8.1 Codes*

3222 Refer to Section A.1.5.1 (on page 3298).

### 3223 **B.1.9 Format of Entries**

3224 Each system interface reference page has a common layout of sections describing the interface.  
3225 This layout is similar to the manual page or “man” page format shipped with most UNIX  
3226 systems, and each header has sections describing the SYNOPSIS, DESCRIPTION, RETURN  
3227 VALUE, and ERRORS. These are the four sections that relate to conformance.

3228 Additional sections are informative, and add considerable information for the application  
3229 developer. EXAMPLES sections provide example usage. APPLICATION USAGE sections  
3230 provide additional caveats, issues, and recommendations to the developer. RATIONALE  
3231 sections give additional information on the decisions made in defining the interface.

3232 FUTURE DIRECTIONS sections act as pointers to related work that may impact the interface in  
3233 the future, and often cautions the developer to architect the code to account for a change in this  
3234 area. Note that a future directions statement should not be taken as a commitment to adopt a  
3235 feature or interface in the future.

3236 The CHANGE HISTORY section describes when the interface was introduced, and how it has  
3237 changed.

3238 Option labels and margin markings in the page can be useful in guiding the application  
3239 developer.

## 3240 B.2 General Information

### 3241 B.2.1 Use and Implementation of Functions

3242 The information concerning the use of functions was adapted from a description in the ISO C  
 3243 standard. Here is an example of how an application program can protect itself from functions  
 3244 that may or may not be macros, rather than true functions:

3245 The *atoi()* function may be used in any of several ways:

- 3246 • By use of its associated header (possibly generating a macro expansion):

```
3247     #include <stdlib.h>
3248     /* ... */
3249     i = atoi(str);
```

- 3250 • By use of its associated header (assuredly generating a true function call):

```
3251     #include <stdlib.h>
3252     #undef atoi
3253     /* ... */
3254     i = atoi(str);
```

3255 or:

```
3256     #include <stdlib.h>
3257     /* ... */
3258     i = (atoi) (str);
```

- 3259 • By explicit declaration:

```
3260     extern int atoi (const char *);
3261     /* ... */
3262     i = atoi(str);
```

- 3263 • By implicit declaration:

```
3264     /* ... */
3265     i = atoi(str);
```

3266 (Assuming no function prototype is in scope. This is not allowed by the ISO C standard for  
 3267 functions with variable arguments; furthermore, parameter type conversion “widening” is  
 3268 subject to different rules in this case.)

3269 Note that the ISO C standard reserves names starting with ‘\_’ for the compiler. Therefore, the  
 3270 compiler could, for example, implement an intrinsic, built-in function *\_asm\_builtin\_atoi()*, which  
 3271 it recognized and expanded into inline assembly code. Then, in *<stdlib.h>*, there could be the  
 3272 following:

```
3273     #define atoi(X) _asm_builtin_atoi(X)
```

3274 The user’s “normal” call to *atoi()* would then be expanded inline, but the implementor would  
 3275 also be required to provide a callable function named *atoi()* for use when the application  
 3276 requires it; for example, if its address is to be stored in a function pointer variable.

3277 **B.2.2 The Compilation Environment**3278 *B.2.2.1 POSIX.1 Symbols*

3279 This and the following section address the issue of “name space pollution”. The ISO C standard  
 3280 requires that the name space beyond what it reserves not be altered except by explicit action of  
 3281 the application writer. This section defines the actions to add the POSIX.1 symbols for those  
 3282 headers where both the ISO C standard and POSIX.1 need to define symbols, and also where the  
 3283 XSI Extension extends the base standard.

3284 When headers are used to provide symbols, there is a potential for introducing symbols that the  
 3285 application writer cannot predict. Ideally, each header should only contain one set of symbols,  
 3286 but this is not practical for historical reasons. Thus, the concept of feature test macros is  
 3287 included. Two feature test macros are explicitly defined by IEEE Std 1003.1-200x; it is expected  
 3288 that future revisions may add to this.

3289 **Note:** Feature test macros allow an application to announce to the implementation its desire to have  
 3290 certain symbols and prototypes exposed. They should not be confused with the version test  
 3291 macros and constants for options in `<unistd.h>` which are the implementation’s way of  
 3292 announcing functionality to the application.

3293 It is further intended that these feature test macros apply only to the headers specified by  
 3294 IEEE Std 1003.1-200x. Implementations are expressly permitted to make visible symbols not  
 3295 specified by IEEE Std 1003.1-200x, within both POSIX.1 and other headers, under the control of  
 3296 feature test macros that are not defined by IEEE Std 1003.1-200x.

3297 **The `_POSIX_C_SOURCE` Feature Test Macro**

3298 Since `_POSIX_SOURCE` specified by the POSIX.1-1990 standard did not have a value associated  
 3299 with it, the `_POSIX_C_SOURCE` macro replaces it, allowing an application to inform the system  
 3300 of the revision of the standard to which it conforms. This symbol will allow implementations to  
 3301 support various revisions of IEEE Std 1003.1-200x simultaneously. For instance, when either  
 3302 `_POSIX_SOURCE` is defined or `_POSIX_C_SOURCE` is defined as 1, the system should make  
 3303 visible the same name space as permitted and required by the POSIX.1-1990 standard. When  
 3304 `_POSIX_C_SOURCE` is defined, the state of `_POSIX_SOURCE` is completely irrelevant.

3305 It is expected that C bindings to future POSIX standards will define new values for  
 3306 `_POSIX_C_SOURCE`, with each new value reserving the name space for that new standard, plus  
 3307 all earlier POSIX standards.

3308 **The `_XOPEN_SOURCE` Feature Test Macro**

3309 The feature test macro `_XOPEN_SOURCE` is provided as the announcement mechanism for the  
 3310 application that it requires functionality from the Single UNIX Specification. `_XOPEN_SOURCE`  
 3311 must be defined to the value 600 before the inclusion of any header to enable the functionality in  
 3312 the Single UNIX Specification. Its definition subsumes the use of `_POSIX_SOURCE` and  
 3313 `_POSIX_C_SOURCE`.

3314 An extract of code from a conforming application, that appears before any `#include` statements,  
 3315 is given below:

```
3316 #define _XOPEN_SOURCE 600 /* Single UNIX Specification, Version 3 */
3317 #include ...
```

3318 Note that the definition of `_XOPEN_SOURCE` with the value 600 makes the definition of  
 3319 `_POSIX_C_SOURCE` redundant and it can safely be omitted.

## 3320 B.2.2.2 The Name Space

3321 The reservation of identifiers is paraphrased from the ISO C standard. The text is included  
3322 because it needs to be part of IEEE Std 1003.1-200x, regardless of possible changes in future  
3323 versions of the ISO C standard.

3324 These identifiers may be used by implementations, particularly for feature test macros.  
3325 Implementations should not use feature test macro names that might be reasonably used by a  
3326 standard.

3327 Including headers more than once is a reasonably common practice, and it should be carried  
3328 forward from the ISO C standard. More significantly, having definitions in more than one  
3329 header is explicitly permitted. Where the potential declaration is “benign” (the same definition  
3330 twice) the declaration can be repeated, if that is permitted by the compiler. (This is usually true  
3331 of macros, for example.) In those situations where a repetition is not benign (for example,  
3332 **typedefs**), conditional compilation must be used. The situation actually occurs both within the  
3333 ISO C standard and within POSIX.1: **time\_t** should be in **<sys/types.h>**, and the ISO C standard  
3334 mandates that it be in **<time.h>**.

3335 The area of name space pollution *versus* additions to structures is difficult because of the macro  
3336 structure of C. The following discussion summarizes all the various problems with and  
3337 objections to the issue.

3338 Note the phrase “user-defined macro”. Users are not permitted to define macro names (or any  
3339 other name) beginning with “\_**[A-Z]**”. Thus, the conflict cannot occur for symbols reserved  
3340 to the vendor’s name space, and the permission to add fields automatically applies, without  
3341 qualification, to those symbols.

3342 1. Data structures (and unions) need to be defined in headers by implementations to meet  
3343 certain requirements of POSIX.1 and the ISO C standard.

3344 2. The structures defined by POSIX.1 are typically minimal, and any practical  
3345 implementation would wish to add fields to these structures either to hold additional  
3346 related information or for backwards-compatibility (or both). Future standards (and *de*  
3347 *facto* standards) would also wish to add to these structures. Issues of field alignment make  
3348 it impractical (at least in the general case) to simply omit fields when they are not defined  
3349 by the particular standard involved.

3350 Struct **dirent** is an example of such a minimal structure (although one could argue about  
3351 whether the other fields need visible names). The *st\_rdev* field of most implementations’  
3352 **stat** structure is a common example where extension is needed and where a conflict could  
3353 occur.

3354 3. Fields in structures are in an independent name space, so the addition of such fields  
3355 presents no problem to the C language itself in that such names cannot interact with  
3356 identically named user symbols because access is qualified by the specific structure name.

3357 4. There is an exception to this: macro processing is done at a lexical level. Thus, symbols  
3358 added to a structure might be recognized as user-provided macro names at the location  
3359 where the structure is declared. This only can occur if the user-provided name is declared  
3360 as a macro before the header declaring the structure is included. The user’s use of the name  
3361 after the declaration cannot interfere with the structure because the symbol is hidden and  
3362 only accessible through access to the structure. Presumably, the user would not declare  
3363 such a macro if there was an intention to use that field name.

3364 5. Macros from the same or a related header might use the additional fields in the structure,  
3365 and those field names might also collide with user macros. Although this is a less frequent  
3366 occurrence, since macros are expanded at the point of use, no constraint on the order of use

3367 of names can apply.

3368 6. An “obvious” solution of using names in the reserved name space and then redefining  
 3369 them as macros when they should be visible does not work because this has the effect of  
 3370 exporting the symbol into the general name space. For example, given a (hypothetical)  
 3371 system-provided header `<h.h>`, and two parts of a C program in `a.c` and `b.c`, in header  
 3372 `<h.h>`:

```
3373     struct foo {
3374         int __i;
3375     }
3376
3377     #ifdef _FEATURE_TEST
3378     #define i __i;
3379     #endif
```

3379 In file `a.c`:

```
3380     #include h.h
3381     extern int i;
3382     ...
```

3383 In file `b.c`:

```
3384     extern int i;
3385     ...
```

3386 The symbol that the user thinks of as `i` in both files has an external name of `__i` in `a.c`; the  
 3387 same symbol `i` in `b.c` has an external name `i` (ignoring any hidden manipulations the  
 3388 compiler might perform on the names). This would cause a mysterious name resolution  
 3389 problem when `a.o` and `b.o` are linked.

3390 Simply avoiding definition then causes alignment problems in the structure.

3391 A structure of the form:

```
3392     struct foo {
3393         union {
3394             int __i;
3395             #ifdef _FEATURE_TEST
3396             int i;
3397             #endif
3398         } __ii;
3399     }
```

3400 does not work because the name of the logical field `i` is `__ii.i`, and introduction of a macro  
 3401 to restore the logical name immediately reintroduces the problem discussed previously  
 3402 (although its manifestation might be more immediate because a syntax error would result  
 3403 if a recursive macro did not cause it to fail first).

3404 7. A more workable solution would be to declare the structure:

```

3405     struct foo {
3406         #ifdef _FEATURE_TEST
3407             int i;
3408         #else
3409             int __i;
3410         #endif
3411     }

```

3412 However, if a macro (particularly one required by a standard) is to be defined that uses  
 3413 this field, two must be defined: one that uses *i*, the other that uses *\_\_i*. If more than one  
 3414 additional field is used in a macro and they are conditional on distinct combinations of  
 3415 features, the complexity goes up as  $2^n$ .

3416 All this leaves a difficult situation: vendors must provide very complex headers to deal with  
 3417 what is conceptually simple and safe—adding a field to a structure. It is the possibility of user-  
 3418 provided macros with the same name that makes this difficult.

3419 Several alternatives were proposed that involved constraining the user's access to part of the  
 3420 name space available to the user (as specified by the ISO C standard). In some cases, this was  
 3421 only until all the headers had been included. There were two proposals discussed that failed to  
 3422 achieve consensus:

- 3423 1. Limiting it for the whole program.
- 3424 2. Restricting the use of identifiers containing only uppercase letters until after all system  
 3425 headers had been included. It was also pointed out that because macros might wish to  
 3426 access fields of a structure (and macro expansion occurs totally at point of use) restricting  
 3427 names in this way would not protect the macro expansion, and thus the solution was  
 3428 inadequate.

3429 It was finally decided that reservation of symbols would occur, but as constrained.

3430 The current wording also allows the addition of fields to a structure, but requires that user  
 3431 macros of the same name not interfere. This allows vendors to do one of the following:

- 3432 • Not create the situation (do not extend the structures with user-accessible names or use the  
 3433 solution in (7) above)
- 3434 • Extend their compilers to allow some way of adding names to structures and macros safely

3435 There are at least two ways that the compiler might be extended: add new preprocessor  
 3436 directives that turn off and on macro expansion for certain symbols (without changing the value  
 3437 of the macro) and a function or lexical operation that suppresses expansion of a word. The latter  
 3438 seems more flexible, particularly because it addresses the problem in macros as well as in  
 3439 declarations.

3440 The following seems to be a possible implementation extension to the C language that will do  
 3441 this: any token that during macro expansion is found to be preceded by three '#' symbols shall  
 3442 not be further expanded in exactly the same way as described for macros that expand to their  
 3443 own name as in Section 3.8.3.4 of the ISO C standard. A vendor may also wish to implement this  
 3444 as an operation that is lexically a function, which might be implemented as:

```

3445     #define __safe_name(x) ###x

```

3446 Using a function notation would insulate vendors from changes in standards until such a  
 3447 functionality is standardized (if ever). Standardization of such a function would be valuable  
 3448 because it would then permit third parties to take advantage of it portably in software they may  
 3449 supply.

3450 The symbols that are “explicitly permitted, but not required by IEEE Std 1003.1-200x” include  
 3451 those classified below. (That is, the symbols classified below might, but are not required to, be  
 3452 present when `_POSIX_C_SOURCE` is defined to have the value 20010xL.)

- 3453 • Symbols in `<limits.h>` and `<unistd.h>` that are defined to indicate support for options or  
 3454 limits that are constant at compile-time.
- 3455 • Symbols in the name space reserved for the implementation by the ISO C standard.
- 3456 • Symbols in a name space reserved for a particular type of extension (for example, type names  
 3457 ending with `_t` in `<sys/types.h>`).
- 3458 • Additional members of structures or unions whose names do not reduce the name space  
 3459 reserved for applications.

3460 Since both implementations and future revisions of IEEE Std 1003.1-200x and other POSIX  
 3461 standards may use symbols in the reserved spaces described in these tables, there is a potential  
 3462 for name space clashes. To avoid future name space clashes when adding symbols,  
 3463 implementations should not use the `posix_`, `POSIX_`, or `_POSIX_` prefixes.

### 3464 B.2.3 Error Numbers

3465 It was the consensus of the standard developers that to allow the conformance document to  
 3466 state that an error occurs and under what conditions, but to disallow a statement that it never  
 3467 occurs, does not make sense. It could be implied by the current wording that this is allowed, but  
 3468 to reduce the possibility of future interpretation requests, it is better to make an explicit  
 3469 statement.

3470 The ISO C standard requires that `errno` be an assignable lvalue. Originally, the definition in |  
 3471 POSIX.1 was stricter than that in the ISO C standard, `extern int errno`, in order to support |  
 3472 historical usage. In a multi-threaded environment, implementing `errno` as a global variable  
 3473 results in non-deterministic results when accessed. It is required, however, that `errno` work as a  
 3474 per-thread error reporting mechanism. In order to do this, a separate `errno` value has to be  
 3475 maintained for each thread. The following section discusses the various alternative solutions  
 3476 that were considered.

3477 In order to avoid this problem altogether for new functions, these functions avoid using `errno`  
 3478 and, instead, return the error number directly as the function return value; a return value of zero  
 3479 indicates that no error was detected.

3480 For any function that can return errors, the function return value is not used for any purpose  
 3481 other than for reporting errors. Even when the output of the function is scalar, it is passed  
 3482 through a function argument. While it might have been possible to allow some scalar outputs to  
 3483 be coded as negative function return values and mixed in with positive error status returns, this  
 3484 was rejected—using the return value for a mixed purpose was judged to be of limited use and  
 3485 error prone.

3486 Checking the value of `errno` alone is not sufficient to determine the existence or type of an error,  
 3487 since it is not required that a successful function call clear `errno`. The variable `errno` should only  
 3488 be examined when the return value of a function indicates that the value of `errno` is meaningful.  
 3489 In that case, the function is required to set the variable to something other than zero.

3490 The variable `errno` shall never be set to zero by any function call; to do so would contradict the  
 3491 ISO C standard.

3492 POSIX.1 requires (in the ERRORS sections of function descriptions) certain error values to be set  
 3493 in certain conditions because many existing applications depend on them. Some error numbers,  
 3494 such as [EFAULT], are entirely implementation-defined and are noted as such in their

3495 description in the ERRORS section. This section otherwise allows wide latitude to the  
 3496 implementation in handling error reporting.

3497 Some of the ERRORS sections in IEEE Std 1003.1-200x have two subsections. The first:

3498 “The function shall fail if:”

3499 could be called the “mandatory” section.

3500 The second:

3501 “The function may fail if:”

3502 could be informally known as the “optional” section.

3503 Attempting to infer the quality of an implementation based on whether it detects optional error  
 3504 conditions is not useful.

3505 Following each one-word symbolic name for an error, there is a description of the error. The  
 3506 rationale for some of the symbolic names follows:

3507 [ECANCELED] This spelling was chosen as being more common.

3508 [EFAULT] Most historical implementations do not catch an error and set *errno* when an  
 3509 invalid address is given to the functions *wait()*, *time()*, or *times()*. Some  
 3510 implementations cannot reliably detect an invalid address. And most systems  
 3511 that detect invalid addresses will do so only for a system call, not for a library  
 3512 routine.

3513 [EFTYPE] This error code was proposed in earlier proposals as “Inappropriate operation  
 3514 for file type”, meaning that the operation requested is not appropriate for the  
 3515 file specified in the function call. This code was proposed, although the same  
 3516 idea was covered by [ENOTTY], because the connotations of the name would  
 3517 be misleading. It was pointed out that the *fcntl()* function uses the error code  
 3518 [EINVAL] for this notion, and hence all instances of [EFTYPE] were changed  
 3519 to this code.

3520 [EINTR] POSIX.1 prohibits conforming implementations from restarting interrupted  
 3521 system calls. However, it does not require that [EINTR] be returned when  
 3522 another legitimate value may be substituted; for example, a partial transfer  
 3523 count when *read()* or *write()* are interrupted. This is only given when the  
 3524 signal catching function returns normally as opposed to returns by  
 3525 mechanisms like *longjmp()* or *siglongjmp()*.

3526 [ELOOP] In specifying conditions under which implementations would generate this  
 3527 error, the following goals were considered:

3528 • To ensure that actual loops are detected, including loops that result from  
 3529 symbolic links across distributed file systems.

3530 • To ensure that during pathname resolution an application can rely on the  
 3531 ability to follow at least {SYMLOOP\_MAX} symbolic links in the absence  
 3532 of a loop.

3533 • To allow implementations to provide the capability of traversing more  
 3534 than {SYMLOOP\_MAX} symbolic links in the absence of a loop.

3535 • To allow implementations to detect loops and generate the error prior to  
 3536 encountering {SYMLOOP\_MAX} symbolic links.

3537	[ENAMETOOLONG]	
3538		When a symbolic link is encountered during pathname resolution, the
3539		contents of that symbolic link are used to create a new pathname. The
3540		standard developers intended to allow, but not require, that implementations
3541		enforce the restriction of {PATH_MAX} on the result of this pathname
3542		substitution.
3543	[ENOMEM]	The term <i>main memory</i> is not used in POSIX.1 because it is implementation-
3544		defined.
3545	[ENOTSUP]	This error code is to be used when an implementation chooses to implement
3546		the required functionality of IEEE Std 1003.1-200x but does not support
3547		optional facilities defined by IEEE Std 1003.1-200x. The return of [ENOSYS] is
3548		to be taken to indicate that the function of the interface is not supported at all;
3549		the function will always fail with this error code.
3550	[ENOTTY]	The symbolic name for this error is derived from a time when device control
3551		was done by <i>ioctl()</i> and that operation was only permitted on a terminal
3552		interface. The term <i>TTY</i> is derived from <i>teletypewriter</i> , the devices to which
3553		this error originally applied.
3554	[EOVERFLOW]	Most of the uses of this error code are related to large file support. Typically,
3555		these cases occur on systems which support multiple programming
3556		environments with different sizes for <i>off_t</i> , but they may also occur in
3557		connection with remote file systems.
3558		In addition, when different programming environments have different widths
3559		for types such as <i>int</i> and <i>uid_t</i> , several functions may encounter a condition
3560		where a value in a particular environment is too wide to be represented. In
3561		that case, this error should be raised. For example, suppose the currently
3562		running process has 64-bit <i>int</i> , and file descriptor 9223372036854775807 is
3563		open and does not have the close-on-exec flag set. If the process then uses
3564		<i>execl()</i> to <i>exec</i> a file compiled in a programming environment with 32-bit <i>int</i> ,
3565		the call to <i>execl()</i> can fail with <i>errno</i> set to [EOVERFLOW]. A similar failure
3566		can occur with <i>execl()</i> if any of the user IDs or any of the group IDs to be
3567		assigned to the new process image are out of range for the executed file's
3568		programming environment.
3569		Note, however, that this condition cannot occur for functions that are
3570		explicitly described as always being successful, such as <i>getpid()</i> .
3571	[EPIPE]	This condition normally generates the signal SIGPIPE; the error is returned if
3572		the signal does not terminate the process.
3573	[EROFS]	In historical implementations, attempting to <i>unlink()</i> or <i>rmdir()</i> a mount point
3574		would generate an [EBUSY] error. An implementation could be envisioned
3575		where such an operation could be performed without error. In this case, if
3576		<i>either</i> the directory entry or the actual data structures reside on a read-only file
3577		system, [EROFS] is the appropriate error to generate. (For example, changing
3578		the link count of a file on a read-only file system could not be done, as is
3579		required by <i>unlink()</i> , and thus an error should be reported.)
3580		Three error numbers, [EDOM], [EILSEQ], and [ERANGE], were added to this section primarily
3581		for consistency with the ISO C standard.

3582 **Alternative Solutions for Per-Thread *errno***

3583 The usual implementation of *errno* as a single global variable does not work in a multi-threaded  
 3584 environment. In such an environment, a thread may make a POSIX.1 call and get a -1 error  
 3585 return, but before that thread can check the value of *errno*, another thread might have made a  
 3586 second POSIX.1 call that also set *errno*. This behavior is unacceptable in robust programs. There  
 3587 were a number of alternatives that were considered for handling the *errno* problem:

- 3588 • Implement *errno* as a per-thread integer variable.
- 3589 • Implement *errno* as a service that can access the per-thread error number.
- 3590 • Change all POSIX.1 calls to accept an extra status argument and avoid setting *errno*.
- 3591 • Change all POSIX.1 calls to raise a language exception.

3592 The first option offers the highest level of compatibility with existing practice but requires  
 3593 special support in the linker, compiler, and/or virtual memory system to support the new  
 3594 concept of thread private variables. When compared with current practice, the third and fourth  
 3595 options are much cleaner, more efficient, and encourage a more robust programming style, but  
 3596 they require new versions of all of the POSIX.1 functions that might detect an error. The second  
 3597 option offers compatibility with existing code that uses the `<errno.h>` header to define the  
 3598 symbol *errno*. In this option, *errno* may be a macro defined:

```
3599     #define errno  (*__errno())
3600     extern int    *__errno();
```

3601 This option may be implemented as a per-thread variable whereby an *errno* field is allocated in  
 3602 the user space object representing a thread, and whereby the function `__errno()` makes a system  
 3603 call to determine the location of its user space object and returns the address of the *errno* field of  
 3604 that object. Another implementation, one that avoids calling the kernel, involves allocating  
 3605 stacks in chunks. The stack allocator keeps a side table indexed by chunk number containing a  
 3606 pointer to the thread object that uses that chunk. The `__errno()` function then looks at the stack  
 3607 pointer, determines the chunk number, and uses that as an index into the chunk table to find its  
 3608 thread object and thus its private value of *errno*. On most architectures, this can be done in four  
 3609 to five instructions. Some compilers may wish to implement `__errno()` inline to improve  
 3610 performance.

3611 **Disallowing Return of the [EINTR] Error Code**

3612 Many blocking interfaces defined by IEEE Std 1003.1-200x may return [EINTR] if interrupted  
 3613 during their execution by a signal handler. Blocking interfaces introduced under the Threads  
 3614 option do not have this property. Instead, they require that the interface appear to be atomic  
 3615 with respect to interruption. In particular, clients of blocking interfaces need not handle any  
 3616 possible [EINTR] return as a special case since it will never occur. If it is necessary to restart  
 3617 operations or complete incomplete operations following the execution of a signal handler, this is  
 3618 handled by the implementation, rather than by the application.

3619 Requiring applications to handle [EINTR] errors on blocking interfaces has been shown to be a  
 3620 frequent source of often unreproducible bugs, and it adds no compelling value to the available  
 3621 functionality. Thus, blocking interfaces introduced for use by multi-threaded programs do not  
 3622 use this paradigm. In particular, in none of the functions `flockfile()`, `pthread_cond_timedwait()`,  
 3623 `pthread_cond_wait()`, `pthread_join()`, `pthread_mutex_lock()`, and `sigwait()` did providing [EINTR]  
 3624 returns add value, or even particularly make sense. Thus, these functions do not provide for an  
 3625 [EINTR] return, even when interrupted by a signal handler. The same arguments can be applied  
 3626 to `sem_wait()`, `sem_trywait()`, `sigwaitinfo()`, and `sigtimedwait()`, but implementations are  
 3627 permitted to return [EINTR] error codes for these functions for compatibility with earlier

3628 versions of IEEE Std 1003.1-200x. Applications cannot rely on calls to these functions returning  
3629 [EINTR] error codes when signals are delivered to the calling thread, but they should allow for  
3630 the possibility.

#### 3631 *B.2.3.1 Additional Error Numbers*

3632 The ISO C standard defines the name space for implementations to add additional error  
3633 numbers.

### 3634 **B.2.4 Signal Concepts**

3635 Historical implementations of signals, using the *signal()* function, have shortcomings that make  
3636 them unreliable for many application uses. Because of this, a new signal mechanism, based very  
3637 closely on the one of 4.2 BSD and 4.3 BSD, was added to POSIX.1.

#### 3638 **Signal Names**

3639 The restriction on the actual type used for **sigset\_t** is intended to guarantee that these objects can  
3640 always be assigned, have their address taken, and be passed as parameters by value. It is not  
3641 intended that this type be a structure including pointers to other data structures, as that could  
3642 impact the portability of applications performing such operations. A reasonable implementation  
3643 could be a structure containing an array of some integer type.

3644 The signals described in IEEE Std 1003.1-200x must have unique values so that they may be  
3645 named as parameters of **case** statements in the body of a C language **switch** clause. However,  
3646 implementation-defined signals may have values that overlap with each other or with signals  
3647 specified in IEEE Std 1003.1-200x. An example of this is SIGABRT, which traditionally overlaps  
3648 some other signal, such as SIGIOT.

3649 SIGKILL, SIGTERM, SIGUSR1, and SIGUSR2 are ordinarily generated only through the explicit  
3650 use of the *kill()* function, although some implementations generate SIGKILL under  
3651 extraordinary circumstances. SIGTERM is traditionally the default signal sent by the *kill*  
3652 command.

3653 The signals SIGBUS, SIGEMT, SIGIOT, SIGTRAP, and SIGSYS were omitted from POSIX.1  
3654 because their behavior is implementation-defined and could not be adequately categorized.  
3655 Conforming implementations may deliver these signals, but must document the circumstances  
3656 under which they are delivered and note any restrictions concerning their delivery. The signals  
3657 SIGFPE, SIGILL, and SIGSEGV are similar in that they also generally result only from  
3658 programming errors. They were included in POSIX.1 because they do indicate three relatively  
3659 well-categorized conditions. They are all defined by the ISO C standard and thus would have to  
3660 be defined by any system with a ISO C standard binding, even if not explicitly included in  
3661 POSIX.1.

3662 There is very little that a Conforming POSIX.1 Application can do by catching, ignoring, or  
3663 masking any of the signals SIGILL, SIGTRAP, SIGIOT, SIGEMT, SIGBUS, SIGSEGV, SIGSYS, or  
3664 SIGFPE. They will generally be generated by the system only in cases of programming errors.  
3665 While it may be desirable for some robust code (for example, a library routine) to be able to  
3666 detect and recover from programming errors in other code, these signals are not nearly sufficient  
3667 for that purpose. One portable use that does exist for these signals is that a command interpreter  
3668 can recognize them as the cause of a process' termination (with *wait()*) and print an appropriate  
3669 message. The mnemonic tags for these signals are derived from their PDP-11 origin.

3670 The signals SIGSTOP, SIGTSTP, SIGTTIN, SIGTTOU, and SIGCONT are provided for job control  
3671 and are unchanged from 4.2 BSD. The signal SIGCHLD is also typically used by job control  
3672 shells to detect children that have terminated or, as in 4.2 BSD, stopped.

3673 Some implementations, including System V, have a signal named SIGCLD, which is similar to  
3674 SIGCHLD in 4.2 BSD. POSIX.1 permits implementations to have a single signal with both  
3675 names. POSIX.1 carefully specifies ways in which conforming applications can avoid the  
3676 semantic differences between the two different implementations. The name SIGCHLD was  
3677 chosen for POSIX.1 because most current application usages of it can remain unchanged in  
3678 conforming applications. SIGCLD in System V has more cases of semantics that POSIX.1 does  
3679 not specify, and thus applications using it are more likely to require changes in addition to the  
3680 name change.

3681 The signals SIGUSR1 and SIGUSR2 are commonly used by applications for notification of  
3682 exceptional behavior and are described as “reserved as application-defined” so that such use is  
3683 not prohibited. Implementations should not generate SIGUSR1 or SIGUSR2, except when  
3684 explicitly requested by *kill()*. It is recommended that libraries not use these two signals, as such  
3685 use in libraries could interfere with their use by applications calling the libraries. If such use is  
3686 unavoidable, it should be documented. It is prudent for non-portable libraries to use non-  
3687 standard signals to avoid conflicts with use of standard signals by portable libraries.

3688 There is no portable way for an application to catch or ignore non-standard signals. Some  
3689 implementations define the range of signal numbers, so applications can install signal-catching  
3690 functions for all of them. Unfortunately, implementation-defined signals often cause problems  
3691 when caught or ignored by applications that do not understand the reason for the signal. While  
3692 the desire exists for an application to be more robust by handling all possible signals (even those  
3693 only generated by *kill()*), no existing mechanism was found to be sufficiently portable to include  
3694 in POSIX.1. The value of such a mechanism, if included, would be diminished given that  
3695 SIGKILL would still not be catchable.

3696 A number of new signal numbers are reserved for applications because the two user signals  
3697 defined by POSIX.1 are insufficient for many realtime applications. A range of signal numbers is  
3698 specified, rather than an enumeration of additional reserved signal names, because different  
3699 applications and application profiles will require a different number of application signals. It is  
3700 not desirable to burden all application domains and therefore all implementations with the  
3701 maximum number of signals required by all possible applications. Note that in this context,  
3702 signal numbers are essentially different signal priorities.

3703 The relatively small number of required additional signals, `{_POSIX_RTSIG_MAX}`, was chosen  
3704 so as not to require an unreasonably large signal mask/set. While this number of signals defined  
3705 in POSIX.1 will fit in a single 32-bit word signal mask, it is recognized that most existing  
3706 implementations define many more signals than are specified in POSIX.1 and, in fact, many  
3707 implementations have already exceeded 32 signals (including the “null signal”). Support of  
3708 `{_POSIX_RTSIG_MAX}` additional signals may push some implementation over the single 32-bit  
3709 word line, but is unlikely to push any implementations that are already over that line beyond the  
3710 64-signal line.

#### 3711 B.2.4.1 Signal Generation and Delivery

3712 The terms defined in this section are not used consistently in documentation of historical  
3713 systems. Each signal can be considered to have a lifetime beginning with *generation* and ending  
3714 with *delivery* or *acceptance*. The POSIX.1 definition of *delivery* does not exclude ignored signals;  
3715 this is considered a more consistent definition. This revised text in several parts of  
3716 IEEE Std 1003.1-200x clarifies the distinct semantics of asynchronous signal *delivery* and  
3717 synchronous signal *acceptance*. The previous wording attempted to categorize both under the  
3718 term *delivery*, which led to conflicts over whether the effects of asynchronous signal delivery  
3719 applied to synchronous signal acceptance.

3720 Signals generated for a process are delivered to only one thread. Thus, if more than one thread is  
3721 eligible to receive a signal, one has to be chosen. The choice of threads is left entirely up to the  
3722 implementation both to allow the widest possible range of conforming implementations and to  
3723 give implementations the freedom to deliver the signal to the “easiest possible” thread should  
3724 there be differences in ease of delivery between different threads.

3725 Note that should multiple delivery among cooperating threads be required by an application,  
3726 this can be trivially constructed out of the provided single-delivery semantics. The construction  
3727 of a *sigwait\_multiple()* function that accomplishes this goal is presented with the rationale for  
3728 *sigwaitinfo()*.

3729 Implementations should deliver unblocked signals as soon after they are generated as possible.  
3730 However, it is difficult for POSIX.1 to make specific requirements about this, beyond those in  
3731 *kill()* and *sigprocmask()*. Even on systems with prompt delivery, scheduling of higher priority  
3732 processes is always likely to cause delays.

3733 In general, the interval between the generation and delivery of unblocked signals cannot be  
3734 detected by an application. Thus, references to pending signals generally apply to blocked,  
3735 pending signals. An implementation registers a signal as pending on the process when no thread  
3736 has the signal unblocked and there are no threads blocked in a *sigwait()* function for that signal.  
3737 Thereafter, the implementation delivers the signal to the first thread that unblocks the signal or  
3738 calls a *sigwait()* function on a signal set containing this signal rather than choosing the recipient  
3739 thread at the time the signal is sent.

3740 In the 4.3 BSD system, signals that are blocked and set to SIG\_IGN are discarded immediately  
3741 upon generation. For a signal that is ignored as its default action, if the action is SIG\_DFL and  
3742 the signal is blocked, a generated signal remains pending. In the 4.1 BSD system and in  
3743 System V, Release 3, two other implementations that support a somewhat similar signal  
3744 mechanism, all ignored, blocked signals remain pending if generated. Because it is not normally  
3745 useful for an application to simultaneously ignore and block the same signal, it was unnecessary  
3746 for POSIX.1 to specify behavior that would invalidate any of the historical implementations.

3747 There is one case in some historical implementations where an unblocked, pending signal does  
3748 not remain pending until it is delivered. In the System V implementation of *signal()*, pending  
3749 signals are discarded when the action is set to SIG\_DFL or a signal-catching routine (as well as to  
3750 SIG\_IGN). Except in the case of setting SIGCHLD to SIG\_DFL, implementations that do this do  
3751 not conform completely to POSIX.1. Some earlier proposals for POSIX.1 explicitly stated this,  
3752 but these statements were redundant due to the requirement that functions defined by POSIX.1  
3753 not change attributes of processes defined by POSIX.1 except as explicitly stated.

3754 POSIX.1 specifically states that the order in which multiple, simultaneously pending signals are  
3755 delivered is unspecified. This order has not been explicitly specified in historical  
3756 implementations, but has remained quite consistent and been known to those familiar with the  
3757 implementations. Thus, there have been cases where applications (usually system utilities) have  
3758 been written with explicit or implicit dependencies on this order. Implementors and others  
3759 porting existing applications may need to be aware of such dependencies.

3760 When there are multiple pending signals that are not blocked, implementations should arrange  
3761 for the delivery of all signals at once, if possible. Some implementations stack calls to all pending  
3762 signal-catching routines, making it appear that each signal-catcher was interrupted by the next  
3763 signal. In this case, the implementation should ensure that this stacking of signals does not  
3764 violate the semantics of the signal masks established by *sigaction()*. Other implementations  
3765 process at most one signal when the operating system is entered, with remaining signals saved  
3766 for later delivery. Although this practice is widespread, this behavior is neither standardized  
3767 nor endorsed. In either case, implementations should attempt to deliver signals associated with  
3768 the current state of the process (for example, SIGFPE) before other signals, if possible.

3769 In 4.2 BSD and 4.3 BSD, it is not permissible to ignore or explicitly block SIGCONT, because if  
3770 blocking or ignoring this signal prevented it from continuing a stopped process, such a process  
3771 could never be continued (only killed by SIGKILL). However, 4.2 BSD and 4.3 BSD do block  
3772 SIGCONT during execution of its signal-catching function when it is caught, creating exactly  
3773 this problem. A proposal was considered to disallow catching SIGCONT in addition to ignoring  
3774 and blocking it, but this limitation led to objections. The consensus was to require that  
3775 SIGCONT always continue a stopped process when generated. This removed the need to  
3776 disallow ignoring or explicit blocking of the signal; note that SIG\_IGN and SIG\_DFL are  
3777 equivalent for SIGCONT .

#### 3778 B.2.4.2 Realtime Signal Generation and Delivery

3779 The Realtime Signals Extension option to POSIX.1 signal generation and delivery behavior is  
3780 required for the following reasons:

- 3781 • The **sigevent** structure is used by other POSIX.1 functions that result in asynchronous event  
3782 notifications to specify the notification mechanism to use and other information needed by  
3783 the notification mechanism. IEEE Std 1003.1-200x defines only three symbolic values for the  
3784 notification mechanism. SIGEV\_NONE is used to indicate that no notification is required  
3785 when the event occurs. This is useful for applications that use asynchronous I/O with polling  
3786 for completion. SIGEV\_SIGNAL indicates that a signal shall be generated when the event  
3787 occurs. SIGEV\_NOTIFY provides for “callback functions” for asynchronous notifications  
3788 done by a function call within the context of a new thread. This provides a multi-threaded  
3789 process a more natural means of notification than signals. The primary difficulty with  
3790 previous notification approaches has been to specify the environment of the notification  
3791 routine.
  - 3792 — One approach is to limit the notification routine to call only functions permitted in a  
3793 signal handler. While the list of permissible functions is clearly stated, this is overly  
3794 restrictive.
  - 3795 — A second approach is to define a new list of functions or classes of functions that are  
3796 explicitly permitted or not permitted. This would give a programmer more lists to deal  
3797 with, which would be awkward.
  - 3798 — The third approach is to define completely the environment for execution of the  
3799 notification function. A clear definition of an execution environment for notification is  
3800 provided by executing the notification function in the environment of a newly created  
3801 thread.

3802 Implementations may support additional notification mechanisms by defining new values  
3803 for *sigev\_notify*.

3804 For a notification type of SIGEV\_SIGNAL, the other members of the **sigevent** structure  
3805 defined by IEEE Std 1003.1-200x specify the realtime signal—that is, the signal number and  
3806 application-defined value that differentiates between occurrences of signals with the same  
3807 number—that will be generated when the event occurs. The structure is defined in  
3808 `<signal.h>`, even though the structure is not directly used by any of the signal functions,  
3809 because it is part of the signals interface used by the POSIX.1b “client functions”. When the  
3810 client functions include `<signal.h>` to define the signal names, the **sigevent** structure will  
3811 also be defined.

3812 An application-defined value passed to the signal handler is used to differentiate between  
3813 different “events” instead of requiring that the application use different signal numbers for  
3814 several reasons:

- 3815 — Realtime applications potentially handle a very large number of different events.  
 3816 Requiring that implementations support a correspondingly large number of distinct  
 3817 signal numbers will adversely impact the performance of signal delivery because the  
 3818 signal masks to be manipulated on entry and exit to the handlers will become large.
- 3819 — Event notifications are prioritized by signal number (the rationale for this is explained in  
 3820 the following paragraphs) and the use of different signal numbers to differentiate  
 3821 between the different event notifications overloads the signal number more than has  
 3822 already been done. It also requires that the application writer make arbitrary assignments  
 3823 of priority to events that are logically of equal priority.
- 3824 A union is defined for the application-defined value so that either an integer constant or a  
 3825 pointer can be portably passed to the signal-catching function. On some architectures a  
 3826 pointer cannot be cast to an `int` and *vice versa*.
- 3827 Use of a structure here with an explicit notification type discriminant rather than explicit  
 3828 parameters to realtime functions, or embedded in other realtime structures, provides for  
 3829 future extensions to IEEE Std 1003.1-200x. Additional, perhaps more efficient, notification  
 3830 mechanisms can be supported for existing realtime function interfaces, such as timers and  
 3831 asynchronous I/O, by extending the `sigevent` structure appropriately. The existing realtime  
 3832 function interfaces will not have to be modified to use any such new notification mechanism.  
 3833 The revised text concerning the `SIGEV_SIGNAL` value makes consistent the semantics of the  
 3834 members of the `sigevent` structure, particularly in the definitions of `lio_listio()` and  
 3835  `aio_fsync()`. For uniformity, other revisions cause this specification to be referred to rather  
 3836 than inaccurately duplicated in the descriptions of functions and structures using the  
 3837 `sigevent` structure. The revised wording does not relax the requirement that the signal  
 3838 number be in the range `SIGRTMIN` to `SIGRTMAX` to guarantee queuing and passing of the  
 3839 application value, since that requirement is still implied by the signal names.
- 3840 • IEEE Std 1003.1-200x is intentionally vague on whether “non-realtime” signal-generating  
 3841 mechanisms can result in a `siginfo_t` being supplied to the handler on delivery. In one  
 3842 existing implementation, a `siginfo_t` is posted on signal generation, even though the  
 3843 implementation does not support queuing of multiple occurrences of a signal. It is not the  
 3844 intent of IEEE Std 1003.1-200x to preclude this, independent of the mandate to define signals  
 3845 that do support queuing. Any interpretation that appears to preclude this is a mistake in the  
 3846 reading or writing of the standard.
  - 3847 • Signals handled by realtime signal handlers might be generated by functions or conditions  
 3848 that do not allow the specification of an application-defined value and do not queue.  
 3849 IEEE Std 1003.1-200x specifies the `si_code` member of the `siginfo_t` structure used in existing  
 3850 practice and defines additional codes so that applications can detect whether an application-  
 3851 defined value is present or not. The code `SI_USER` for `kill()`-generated signals is adopted  
 3852 from existing practice.
  - 3853 • The `sigaction()` `sa_flags` value `SA_SIGINFO` tells the implementation that the signal-catching  
 3854 function expects two additional arguments. When the flag is not set, a single argument, the  
 3855 signal number, is passed as specified by IEEE Std 1003.1-200x. Although IEEE Std 1003.1-200x  
 3856 does not explicitly allow the `info` argument to the handler function to be `NULL`, this is  
 3857 existing practice. This provides for compatibility with programs whose signal-catching  
 3858 functions are not prepared to accept the additional arguments. IEEE Std 1003.1-200x is  
 3859 explicitly unspecified as to whether signals actually queue when `SA_SIGINFO` is not set for a  
 3860 signal, as there appear to be no benefits to applications in specifying one behavior or another.  
 3861 One existing implementation queues a `siginfo_t` on each signal generation, unless the signal  
 3862 is already pending, in which case the implementation discards the new `siginfo_t`; that is, the  
 3863 queue length is never greater than one. This implementation only examines `SA_SIGINFO` on

3864 signal delivery, discarding the queued `siginfo_t` if its delivery was not requested.

3865 IEEE Std 1003.1-200x specifies several new values for the `si_code` member of the `siginfo_t`  
3866 structure. In existing practice, a `si_code` value of less than or equal to zero indicates that the  
3867 signal was generated by a process via the `kill()` function. In existing practice, values of `si_code`  
3868 that provide additional information for implementation-generated signals, such as SIGFPE or  
3869 SIGSEGV, are all positive. Thus, if implementations define the new constants specified in  
3870 IEEE Std 1003.1-200x to be negative numbers, programs written to use existing practice will  
3871 not break. IEEE Std 1003.1-200x chose not to attempt to specify existing practice values of  
3872 `si_code` other than SI\_USER both because it was deemed beyond the scope of  
3873 IEEE Std 1003.1-200x and because many of the values in existing practice appear to be  
3874 platform and implementation-defined. But, IEEE Std 1003.1-200x does specify that if an  
3875 implementation—for example, one that does not have existing practice in this area—chooses  
3876 to define additional values for `si_code`, these values have to be different from the values of the  
3877 symbols specified by IEEE Std 1003.1-200x. This will allow conforming applications to  
3878 differentiate between signals generated by one of the POSIX.1b asynchronous events and  
3879 those generated by other implementation events in a manner compatible with existing  
3880 practice.

3881 The unique values of `si_code` for the POSIX.1b asynchronous events have implications for  
3882 implementations of, for example, asynchronous I/O or message passing in user space library  
3883 code. Such an implementation will be required to provide a hidden interface to the signal  
3884 generation mechanism that allows the library to specify the standard values of `si_code`.

3885 Existing practice also defines additional members of `siginfo_t`, such as the process ID and  
3886 user ID of the sending process for `kill()`-generated signals. These members were deemed not  
3887 necessary to meet the requirements of realtime applications and are not specified by  
3888 IEEE Std 1003.1-200x. Neither are they precluded.

3889 The third argument to the signal-catching function, `context`, is left undefined by  
3890 IEEE Std 1003.1-200x, but is specified in the interface because it matches existing practice for  
3891 the SA\_SIGINFO flag. It was considered undesirable to require a separate implementation  
3892 for SA\_SIGINFO for POSIX conformance on implementations that already support the two  
3893 additional parameters.

- 3894 • The requirement to deliver lower numbered signals in the range SIGRTMIN to SIGRTMAX  
3895 first, when multiple unblocked signals are pending, results from several considerations:
  - 3896 — A method is required to prioritize event notifications. The signal number was chosen  
3897 instead of, for instance, associating a separate priority with each request, because an  
3898 implementation has to check pending signals at various points and select one for delivery  
3899 when more than one is pending. Specifying a selection order is the minimal additional  
3900 semantic that will achieve prioritized delivery. If a separate priority were to be associated  
3901 with queued signals, it would be necessary for an implementation to search all non-  
3902 empty, non-blocked signal queues and select from among them the pending signal with  
3903 the highest priority. This would significantly increase the cost of and decrease the  
3904 determinism of signal delivery.
  - 3905 — Given the specified selection of the lowest numeric unblocked pending signal,  
3906 preemptive priority signal delivery can be achieved using signal numbers and signal  
3907 masks by ensuring that the `sa_mask` for each signal number blocks all signals with a  
3908 higher numeric value.

3909 For realtime applications that want to use only the newly defined realtime signal numbers  
3910 without interference from the standard signals, this can be achieved by blocking all of the  
3911 standard signals in the process signal mask and in the `sa_mask` installed by the signal

3912 action for the realtime signal handlers.  
 3913 IEEE Std 1003.1-200x explicitly leaves unspecified the ordering of signals outside of the range  
 3914 of realtime signals and the ordering of signals within this range with respect to those outside  
 3915 the range. It was believed that this would unduly constrain implementations or standards in  
 3916 the future definition of new signals.

#### 3917 B.2.4.3 Signal Actions

3918 Early proposals mentioned SIGCONT as a second exception to the rule that signals are not  
 3919 delivered to stopped processes until continued. Because IEEE Std 1003.1-200x now specifies that  
 3920 SIGCONT causes the stopped process to continue when it is generated, delivery of SIGCONT is  
 3921 not prevented because a process is stopped, even without an explicit exception to this rule.

3922 Ignoring a signal by setting the action to SIG\_IGN (or SIG\_DFL for signals whose default action  
 3923 is to ignore) is not the same as installing a signal-catching function that simply returns. Invoking  
 3924 such a function will interrupt certain system functions that block processes (for example, *wait()*,  
 3925 *sigsuspend()*, *pause()*, *read()*, *write()*) while ignoring a signal has no such effect on the process.

3926 Historical implementations discard pending signals when the action is set to SIG\_IGN.  
 3927 However, they do not always do the same when the action is set to SIG\_DFL and the default  
 3928 action is to ignore the signal. IEEE Std 1003.1-200x requires this for the sake of consistency and  
 3929 also for completeness, since the only signal this applies to is SIGCHLD, and  
 3930 IEEE Std 1003.1-200x disallows setting its action to SIG\_IGN.

3931 Some implementations (System V, for example) assign different semantics for SIGCLD |  
 3932 depending on whether the action is set to SIG\_IGN or SIG\_DFL. Since POSIX.1 requires that the |  
 3933 default action for SIGCHLD be to ignore the signal, applications should always set the action to |  
 3934 SIG\_DFL in order to avoid SIGCHLD.

3935 Whether or not an implementation allows SIG\_IGN as a SIGCHLD disposition to be inherited |  
 3936 across a call to one of the *exec* family of functions or *posix\_spawn()* is explicitly left as |  
 3937 unspecified. This change was made as a result of IEEE PASC Interpretation 1003.1 #132, and |  
 3938 permits the implementation to decide between the following alternatives: |

- 3939 • Unconditionally leave SIGCHLD set to SIG\_IGN, in which case the implementation would |  
 3940 not allow applications that assume inheritance of SIG\_DFL to conform to |  
 3941 IEEE Std 1003.1-200x without change. The implementation would, however, retain an ability |  
 3942 to control applications that create child processes but never call on the *wait* family of |  
 3943 functions, potentially filling up the process table. |
- 3944 • Unconditionally reset SIGCHLD to SIG\_DFL, in which case the implementation would allow |  
 3945 applications that assume inheritance of SIG\_DFL to conform. The implementation would, |  
 3946 however, lose an ability to control applications that spawn child processes but never reap |  
 3947 them. |
- 3948 • Provide some mechanism, not specified in IEEE Std 1003.1-200x, to control inherited |  
 3949 SIGCHLD dispositions. |

3950 Some implementations (System V, for example) will deliver a SIGCLD signal immediately when |  
 3951 a process establishes a signal-catching function for SIGCLD when that process has a child that |  
 3952 has already terminated. Other implementations, such as 4.3 BSD, do not generate a new |  
 3953 SIGCHLD signal in this way. In general, a process should not attempt to alter the signal action |  
 3954 for the SIGCHLD signal while it has any outstanding children. However, it is not always |  
 3955 possible for a process to avoid this; for example, shells sometimes start up processes in pipelines |  
 3956 with other processes from the pipeline as children. Processes that cannot ensure that they have |  
 3957 no children when altering the signal action for SIGCHLD thus need to be prepared for, but not

3958 depend on, generation of an immediate SIGCHLD signal.

3959 The default action of the stop signals (SIGSTOP , SIGTSTP, SIGTTIN, SIGTTOU) is to stop a  
 3960 process that is executing. If a stop signal is delivered to a process that is already stopped, it has  
 3961 no effect. In fact, if a stop signal is generated for a stopped process whose signal mask blocks the  
 3962 signal, the signal will never be delivered to the process since the process must receive a  
 3963 SIGCONT, which discards all pending stop signals, in order to continue executing.

3964 The SIGCONT signal shall continue a stopped process even if SIGCONT is blocked (or ignored).  
 3965 However, if a signal-catching routine has been established for SIGCONT, it will not be entered  
 3966 until SIGCONT is unblocked.

3967 If a process in an orphaned process group stops, it is no longer under the control of a job control  
 3968 shell and hence would not normally ever be continued. Because of this, orphaned processes that  
 3969 receive terminal-related stop signals (SIGTSTP , SIGTTIN, SIGTTOU, but not SIGSTOP ) must  
 3970 not be allowed to stop. The goal is to prevent stopped processes from languishing forever. (As  
 3971 SIGSTOP is sent only via *kill()*, it is assumed that the process or user sending a SIGSTOP can  
 3972 send a SIGCONT when desired.) Instead, the system must discard the stop signal. As an  
 3973 extension, it may also deliver another signal in its place. 4.3 BSD sends a SIGKILL, which is  
 3974 overly effective because SIGKILL is not catchable. Another possible choice is SIGHUP. 4.3 BSD  
 3975 also does this for orphaned processes (processes whose parent has terminated) rather than for  
 3976 members of orphaned process groups; this is less desirable because job control shells manage  
 3977 process groups. POSIX.1 also prevents SIGTTIN and SIGTTOU signals from being generated for  
 3978 processes in orphaned process groups as a direct result of activity on a terminal, preventing  
 3979 infinite loops when *read()* and *write()* calls generate signals that are discarded; see Section  
 3980 A.11.1.4 (on page 3357). A similar restriction on the generation of SIGTSTP was considered, but  
 3981 that would be unnecessary and more difficult to implement due to its asynchronous nature.

3982 Although POSIX.1 requires that signal-catching functions be called with only one argument,  
 3983 there is nothing to prevent conforming implementations from extending POSIX.1 to pass  
 3984 additional arguments, as long as Strictly Conforming POSIX.1 Applications continue to compile  
 3985 and execute correctly. Most historical implementations do, in fact, pass additional, signal-  
 3986 specific arguments to certain signal-catching routines.

3987 There was a proposal to change the declared type of the signal handler to:

```
3988     void func (int sig, ...);
```

3989 The usage of ellipses ("...") is ISO C standard syntax to indicate a variable number of  
 3990 arguments. Its use was intended to allow the implementation to pass additional information to  
 3991 the signal handler in a standard manner.

3992 Unfortunately, this construct would require all signal handlers to be defined with this syntax  
 3993 because the ISO C standard allows implementations to use a different parameter passing  
 3994 mechanism for variable parameter lists than for non-variable parameter lists. Thus, all existing  
 3995 signal handlers in all existing applications would have to be changed to use the variable syntax  
 3996 in order to be standard and portable. This is in conflict with the goal of Minimal Changes to  
 3997 Existing Application Code.

3998 When terminating a process from a signal-catching function, processes should be aware of any  
 3999 interpretation that their parent may make of the status returned by *wait()* or *waitpid()*. In  
 4000 particular, a signal-catching function should not call *exit(0)* or *\_exit(0)* unless it wants to indicate  
 4001 successful termination. A non-zero argument to *exit()* or *\_exit()* can be used to indicate  
 4002 unsuccessful termination. Alternatively, the process can use *kill()* to send itself a fatal signal  
 4003 (first ensuring that the signal is set to the default action and not blocked). See also the  
 4004 RATIONALE section of the *\_exit()* function.

4005 The behavior of *unsafe* functions, as defined by this section, is undefined when they are invoked  
4006 from signal-catching functions in certain circumstances. The behavior of reentrant functions, as  
4007 defined by this section, is as specified by POSIX.1, regardless of invocation from a signal-  
4008 catching function. This is the only intended meaning of the statement that reentrant functions  
4009 may be used in signal-catching functions without restriction. Applications must still consider all  
4010 effects of such functions on such things as data structures, files, and process state. In particular,  
4011 application writers need to consider the restrictions on interactions when interrupting *sleep()*  
4012 (see *sleep()*) and interactions among multiple handles for a file description. The fact that any  
4013 specific function is listed as reentrant does not necessarily mean that invocation of that function  
4014 from a signal-catching function is recommended.

4015 In order to prevent errors arising from interrupting non-reentrant function calls, applications  
4016 should protect calls to these functions either by blocking the appropriate signals or through the  
4017 use of some programmatic semaphore. POSIX.1 does not address the more general problem of  
4018 synchronizing access to shared data structures. Note in particular that even the “safe” functions  
4019 may modify the global variable *errno*; the signal-catching function may want to save and restore  
4020 its value. The same principles apply to the reentrancy of application routines and asynchronous  
4021 data access.

4022 Note that *longjmp()* and *siglongjmp()* are not in the list of reentrant functions. This is because the  
4023 code executing after *longjmp()* or *siglongjmp()* can call any unsafe functions with the same  
4024 danger as calling those unsafe functions directly from the signal handler. Applications that use  
4025 *longjmp()* or *siglongjmp()* out of signal handlers require rigorous protection in order to be  
4026 portable. Many of the other functions that are excluded from the list are traditionally  
4027 implemented using either the C language *malloc()* or *free()* functions or the ISO C standard I/O  
4028 library, both of which traditionally use data structures in a non-reentrant manner. Because any  
4029 combination of different functions using a common data structure can cause reentrancy  
4030 problems, POSIX.1 does not define the behavior when any unsafe function is called in a signal  
4031 handler that interrupts any unsafe function.

4032 The only realtime extension to signal actions is the addition of the additional parameters to the  
4033 signal-catching function. This extension has been explained and motivated in the previous  
4034 section. In making this extension, though, developers of POSIX.1b ran into issues relating to  
4035 function prototypes. In response to input from the POSIX.1 standard developers, members were  
4036 added to the **sigaction** structure to specify function prototypes for the newer signal-catching  
4037 function specified by POSIX.1b. These members follow changes that are being made to POSIX.1.  
4038 Note that IEEE Std 1003.1-200x explicitly states that these fields may overlap so that a union can  
4039 be defined. This will enable existing implementations of POSIX.1 to maintain binary-  
4040 compatibility when these extensions are added.

4041 The **siginfo\_t** structure was adopted for passing the application-defined value to match existing  
4042 practice, but the existing practice has no provision for an application-defined value, so this was  
4043 added. Note that POSIX normally reserves the “\_t” type designation for opaque types. The  
4044 **siginfo\_t** structure breaks with this convention to follow existing practice and thus promote  
4045 portability. Standardization of the existing practice for the other members of this structure may  
4046 be addressed in the future.

4047 Although it is not explicitly visible to applications, there are additional semantics for signal  
4048 actions implied by queued signals and their interaction with other POSIX.1b realtime functions.  
4049 Specifically:

- 4050 • It is not necessary to queue signals whose action is SIG\_IGN.
- 4051 • For implementations that support POSIX.1b timers, some interaction with the timer functions  
4052 at signal delivery is implied to manage the timer overrun count.

4053 **B.2.4.4** *Signal Effects on Other Functions*

4054 The most common behavior of an interrupted function after a signal-catching function returns is  
4055 for the interrupted function to give an [EINTR] error. However, there are a number of specific  
4056 exceptions, including *sleep()* and certain situations with *read()* and *write()*.

4057 The historical implementations of many functions defined by IEEE Std 1003.1-200x are not  
4058 interruptible, but delay delivery of signals generated during their execution until after they  
4059 complete. This is never a problem for functions that are guaranteed to complete in a short  
4060 (imperceptible to a human) period of time. It is normally those functions that can suspend a  
4061 process indefinitely or for long periods of time (for example, *wait()*, *pause()*, *sigsuspend()*, *sleep()*,  
4062 or *read()/write()* on a slow device like a terminal] that are interruptible. This permits  
4063 applications to respond to interactive signals or to set timeouts on calls to most such functions  
4064 with *alarm()*. Therefore, implementations should generally make such functions (including ones  
4065 defined as extensions) interruptible.

4066 Functions not mentioned explicitly as interruptible may be so on some implementations,  
4067 possibly as an extension where the function gives an [EINTR] error. There are several functions  
4068 (for example, *getpid()*, *getuid()*) that are specified as never returning an error, which can thus  
4069 never be extended in this way.

4070 **B.2.5** **Standard I/O Streams**4071 **B.2.5.1** *Interaction of File Descriptors and Standard I/O Streams*

4072 There is no additional rationale provided for this section.

4073 **B.2.5.2** *Stream Orientation and Encoding Rules*

4074 There is no additional rationale provided for this section.

4075 **B.2.6** **STREAMS**

4076 STREAMS are introduced into IEEE Std 1003.1-200x as part of the alignment with the Single  
4077 UNIX Specification, but marked as an option in recognition that not all systems may wish to  
4078 implement the facility. The option within IEEE Std 1003.1-200x is denoted by the XSR margin  
4079 marker. The standard developers made this option independent of the XSI option.

4080 STREAMS are a method of implementing network services and other character-based  
4081 input/output mechanisms, with the STREAM being a full-duplex connection between a process  
4082 and a device. STREAMS provides direct access to protocol modules, and optional protocol  
4083 modules can be interposed between the process-end of the STREAM and the device-driver at the  
4084 device-end of the STREAM. Pipes can be implemented using the STREAMS mechanism, so they  
4085 can provide process-to-process as well as process-to-device communications.

4086 This section introduces STREAMS I/O, the message types used to control them, an overview of  
4087 the priority mechanism, and the interfaces used to access them.

4088 **B.2.6.1** *Accessing STREAMS*

4089 There is no additional rationale provided for this section.

**4090 B.2.7 XSI Interprocess Communication**

4091 There are two forms of IPC supported as options in IEEE Std 1003.1-200x. The traditional  
4092 System V IPC routines derived from the SVID—that is, the *msg\**(), *sem\**(), and *shm\**()  
4093 interfaces—are mandatory on XSI-conformant systems. Thus, all XSI-conformant systems  
4094 provide the same mechanisms for manipulating messages, shared memory, and semaphores.

4095 In addition, the POSIX Realtime Extension provides an alternate set of routines for those systems  
4096 supporting the appropriate options.

4097 The application writer is presented with a choice: the System V interfaces or the POSIX  
4098 interfaces (loosely derived from the Berkeley interfaces). The XSI profile prefers the System V  
4099 interfaces, but the POSIX interfaces may be more suitable for realtime or other performance-  
4100 sensitive applications.

**4101 B.2.7.1 IPC General Information**

4102 General information that is shared by all three mechanisms is described in this section. The  
4103 common permissions mechanism is briefly introduced, describing the mode bits, and how they  
4104 are used to determine whether or not a process has access to read or write/alter the appropriate  
4105 instance of one of the IPC mechanisms. All other relevant information is contained in the  
4106 reference pages themselves.

4107 The semaphore type of IPC allows processes to communicate through the exchange of  
4108 semaphore values. A semaphore is a positive integer. Since many applications require the use of  
4109 more than one semaphore, XSI-conformant systems have the ability to create sets or arrays of  
4110 semaphores.

4111 Calls to support semaphores include:

4112 *semctl*(), *semget*(), *semop*()

4113 Semaphore sets are created by using the *semget*() function.

4114 The message type of IPC allows process to communicate through the exchange of data stored in  
4115 buffers. This data is transmitted between processes in discrete portions known as messages.

4116 Calls to support message queues include:

4117 *msgctl*(), *msgget*(), *msgrcv*(), *msgsnd*()

4118 The share memory type of IPC allows two or more processes to share memory and consequently  
4119 the data contained therein. This is done by allowing processes to set up access to a common  
4120 memory address space. This sharing of memory provides a fast means of exchange of data  
4121 between processes.

4122 Calls to support shared memory include:

4123 *shmctl*(), *shmdt*(), *shmget*()

4124 The *ftok*() interface is also provided.

4125 **B.2.8 Realtime**4126 **Advisory Information**

4127 POSIX.1b contains an Informative Annex with proposed interfaces for “real-time files”. These  
 4128 interfaces could determine groups of the exact parameters required to do “direct I/O” or  
 4129 “extents”. These interfaces were objected to by a significant portion of the balloting group as too  
 4130 complex. A conforming application had little chance of correctly navigating the large parameter  
 4131 space to match its desires to the system. In addition, they only applied to a new type of file  
 4132 (realtime files) and they told the implementation exactly what to do as opposed to advising the  
 4133 implementation on application behavior and letting it optimize for the system the (portable)  
 4134 application was running on. For example, it was not clear how a system that had a disk array  
 4135 should set its parameters.

4136 There seemed to be several overall goals:

- 4137 • Optimizing sequential access
- 4138 • Optimizing caching behavior
- 4139 • Optimizing I/O data transfer
- 4140 • Preallocation

4141 The advisory interfaces, *posix\_fadvise()* and *posix\_madvise()*, satisfy the first two goals. The  
 4142 `POSIX_FADV_SEQUENTIAL` and `POSIX_MADV_SEQUENTIAL` advice tells the  
 4143 implementation to expect serial access. Typically the system will prefetch the next several serial  
 4144 accesses in order to overlap I/O. It may also free previously accessed serial data if memory is  
 4145 tight. If the application is not doing serial access it can use `POSIX_FADV_WILLNEED` and  
 4146 `POSIX_MADV_WILLNEED` to accomplish I/O overlap, as required. When the application  
 4147 advises `POSIX_FADV_RANDOM` or `POSIX_MADV_RANDOM` behavior, the implementation  
 4148 usually tries to fetch a minimum amount of data with each request and it does not expect much  
 4149 locality. `POSIX_FADV_DONTNEED` and `POSIX_MADV_DONTNEED` allow the system to free  
 4150 up caching resources as the data will not be required in the near future.

4151 `POSIX_FADV_NOREUSE` tells the system that caching the specified data is not optimal. For file  
 4152 I/O, the transfer should go directly to the user buffer instead of being cached internally by the  
 4153 implementation. To portably perform direct disk I/O on all systems, the application must  
 4154 perform its I/O transfers according to the following rules:

- 4155 1. The user buffer should be aligned according to the `{POSIX_REC_XFER_ALIGN} pathconf()`  
 4156 variable.
- 4157 2. The number of bytes transferred in an I/O operation should be a multiple of the  
 4158 `{POSIX_ALLOC_SIZE_MIN} pathconf()` variable.
- 4159 3. The offset into the file at the start of an I/O operation should be a multiple of the  
 4160 `{POSIX_ALLOC_SIZE_MIN} pathconf()` variable.
- 4161 4. The application should ensure that all threads which open a given file specify  
 4162 `POSIX_FADV_NOREUSE` to be sure that there is no unexpected interaction between  
 4163 threads using buffered I/O and threads using direct I/O to the same file.

4164 In some cases, a user buffer must be properly aligned in order to be transferred directly to/from  
 4165 the device. The `{POSIX_REC_XFER_ALIGN} pathconf()` variable tells the application the proper  
 4166 alignment.

4167 The preallocation goal is met by the space control function, *posix\_fallocate()*. The application can  
 4168 use *posix\_fallocate()* to guarantee no [ENOSPC] errors and to improve performance by prepaying

4169 any overhead required for block allocation.

4170 Implementations may use information conveyed by a previous *posix\_fadvise()* call to influence  
4171 the manner in which allocation is performed. For example, if an application did the following  
4172 calls:

```
4173     fd = open("file");
4174     posix_fadvise(fd, offset, len, POSIX_FADV_SEQUENTIAL);
4175     posix_fallocate(fd, len, size);
```

4176 an implementation might allocate the file contiguously on disk.

4177 Finally, the *pathconf()* variables {*POSIX\_REC\_MIN\_XFER\_SIZE*},  
4178 {*POSIX\_REC\_MAX\_XFER\_SIZE*}, and {*POSIX\_REC\_INCR\_XFER\_SIZE*} tell the application a  
4179 range of transfer sizes that are recommended for best I/O performance.

4180 Where bounded response time is required, the vendor can supply the appropriate settings of the  
4181 advisories to achieve a guaranteed performance level.

4182 The interfaces meet the goals while allowing applications using regular files to take advantage of  
4183 performance optimizations. The interfaces tell the implementation expected application  
4184 behavior which the implementation can use to optimize performance on a particular system  
4185 with a particular dynamic load.

4186 The *posix\_memalign()* function was added to allow for the allocation of specifically aligned  
4187 buffers; for example, for {*POSIX\_REC\_XFER\_ALIGN*}.

4188 The working group also considered the alternative of adding a function which would return an  
4189 aligned pointer to memory within a user supplied buffer. This was not considered to be the best  
4190 method, because it potentially wastes large amounts of memory when buffers need to be aligned  
4191 on large alignment boundaries.

## 4192 **Message Passing**

4193 This section provides the rationale for the definition of the message passing interface in  
4194 IEEE Std 1003.1-200x. This is presented in terms of the objectives, models, and requirements  
4195 imposed upon this interface.

### 4196 • Objectives

4197 Many applications, including both realtime and database applications, require a means of  
4198 passing arbitrary amounts of data between cooperating processes comprising the overall  
4199 application on one or more processors. Many conventional interfaces for interprocess  
4200 communication are insufficient for realtime applications in that efficient and deterministic  
4201 data passing methods cannot be implemented. This has prompted the definition of message  
4202 passing interfaces providing these facilities:

- 4203 — Open a message queue.
- 4204 — Send a message to a message queue.
- 4205 — Receive a message from a queue, either synchronously or asynchronously.
- 4206 — Alter message queue attributes for flow and resource control.

4207 It is assumed that an application may consist of multiple cooperating processes and that  
4208 these processes may wish to communicate and coordinate their activities. The message  
4209 passing facility described in IEEE Std 1003.1-200x allows processes to communicate through  
4210 system-wide queues. These message queues are accessed through names that may be  
4211 pathnames. A message queue can be opened for use by multiple sending and/or multiple

- 4212 receiving processes.
- 4213 • Background on Embedded Applications
- 4214 Interprocess communication utilizing message passing is a key facility for the construction of  
4215 deterministic, high-performance realtime applications. The facility is present in all realtime  
4216 systems and is the framework upon which the application is constructed. The performance of  
4217 the facility is usually a direct indication of the performance of the resulting application.
- 4218 Realtime applications, especially for embedded systems, are typically designed around the  
4219 performance constraints imposed by the message passing mechanisms. Applications for  
4220 embedded systems are typically very tightly constrained. Application writers expect to  
4221 design and control the entire system. In order to minimize system costs, the writer will  
4222 attempt to use all resources to their utmost and minimize the requirement to add additional  
4223 memory or processors.
- 4224 The embedded applications usually share address spaces and only a simple message passing  
4225 mechanism is required. The application can readily access common data incurring only  
4226 mutual-exclusion overheads. The models desired are the simplest possible with the  
4227 application building higher-level facilities only when needed.
- 4228 • Requirements
- 4229 The following requirements determined the features of the message passing facilities defined  
4230 in IEEE Std 1003.1-200x:
- 4231 — Naming of Message Queues
- 4232 The mechanism for gaining access to a message queue is a pathname evaluated in a  
4233 context that is allowed to be a file system name space, or it can be independent of any file  
4234 system. This is a specific attempt to allow implementations based on either method in  
4235 order to address both embedded systems and to also allow implementation in larger  
4236 systems.
- 4237 The interface of *mq\_open()* is defined to allow but not require the access control and name  
4238 conflicts resulting from utilizing a file system for name resolution. All required behavior  
4239 is specified for the access control case. Yet a conforming implementation, such as an  
4240 embedded system kernel, may define that there are no distinctions between users and  
4241 may define that all process have all access privileges.
- 4242 — Embedded System Naming
- 4243 Embedded systems need to be able to utilize independent name spaces for accessing the  
4244 various system objects. They typically do not have a file system, precluding its utilization  
4245 as a common name resolution mechanism. The modularity of an embedded system limits  
4246 the connections between separate mechanisms that can be allowed.
- 4247 Embedded systems typically do not have any access protection. Since the system does not  
4248 support the mixing of applications from different areas, and usually does not even have  
4249 the concept of an authorization entity, access control is not useful.
- 4250 — Large System Naming
- 4251 On systems with more functionality, the name resolution must support the ability to use  
4252 the file system as the name resolution mechanism/object storage medium and to have  
4253 control over access to the objects. Utilizing the pathname space can result in further errors  
4254 when the names conflict with other objects.
- 4255 — Fixed Size of Messages

4256 The interfaces impose a fixed upper bound on the size of messages that can be sent to a  
4257 specific message queue. The size is set on an individual queue basis and cannot be  
4258 changed dynamically.

4259 The purpose of the fixed size is to increase the ability of the system to optimize the  
4260 implementation of *mq\_send()* and *mq\_receive()*. With fixed sizes of messages and fixed  
4261 numbers of messages, specific message blocks can be pre-allocated. This eliminates a  
4262 significant amount of checking for errors and boundary conditions. Additionally, an  
4263 implementation can optimize data copying to maximize performance. Finally, with a  
4264 restricted range of message sizes, an implementation is better able to provide  
4265 deterministic operations.

#### 4266 — Prioritization of Messages

4267 Message prioritization allows the application to determine the order in which messages  
4268 are received. Prioritization of messages is a key facility that is provided by most realtime  
4269 kernels and is heavily utilized by the applications. The major purpose of having priorities  
4270 in message queues is to avoid priority inversions in the message system, where a high-  
4271 priority message is delayed behind one or more lower-priority messages. This allows the  
4272 applications to be designed so that they do not need to be interrupted in order to change  
4273 the flow of control when exceptional conditions occur. The prioritization does add  
4274 additional overhead to the message operations in those cases it is actually used but a  
4275 clever implementation can optimize for the FIFO case to make that more efficient.

#### 4276 — Asynchronous Notification

4277 The interface supports the ability to have a task asynchronously notified of the  
4278 availability of a message on the queue. The purpose of this facility is to allow the task to  
4279 perform other functions and yet still be notified that a message has become available on  
4280 the queue.

4281 To understand the requirement for this function, it is useful to understand two models of  
4282 application design: a single task performing multiple functions and multiple tasks  
4283 performing a single function. Each of these models has advantages.

4284 Asynchronous notification is required to build the model of a single task performing  
4285 multiple operations. This model typically results from either the expectation that  
4286 interruption is less expensive than utilizing a separate task or from the growth of the  
4287 application to include additional functions.

### 4288 **Semaphores**

4289 Semaphores are a high-performance process synchronization mechanism. Semaphores are  
4290 named by null-terminated strings of characters.

4291 A semaphore is created using the *sem\_init()* function or the *sem\_open()* function with the  
4292 *O\_CREAT* flag set in *oflag*.

4293 To use a semaphore, a process has to first initialize the semaphore or inherit an open descriptor  
4294 for the semaphore via *fork()*.

4295 A semaphore preserves its state when the last reference is closed. For example, if a semaphore  
4296 has a value of 13 when the last reference is closed, it will have a value of 13 when it is next  
4297 opened.

4298 When a semaphore is created, an initial state for the semaphore has to be provided. This value is  
4299 a non-negative integer. Negative values are not possible since they indicate the presence of  
4300 blocked processes. The persistence of any of these objects across a system crash or a system

- 4301 reboot is undefined. Conforming applications shall not depend on any sort of persistence across  
4302 a system reboot or a system crash.
- 4303 • Models and Requirements
- 4304 A realtime system requires synchronization and communication between the processes  
4305 comprising the overall application. An efficient and reliable synchronization mechanism has  
4306 to be provided in a realtime system that will allow more than one schedulable process  
4307 mutually-exclusive access to the same resource. This synchronization mechanism has to  
4308 allow for the optimal implementation of synchronization or systems implementors will  
4309 define other, more cost-effective methods.
- 4310 At issue are the methods whereby multiple processes (tasks) can be designed and  
4311 implemented to work together in order to perform a single function. This requires  
4312 interprocess communication and synchronization. A semaphore mechanism is the lowest  
4313 level of synchronization that can be provided by an operating system.
- 4314 A semaphore is defined as an object that has an integral value and a set of blocked processes  
4315 associated with it. If the value is positive or zero, then the set of blocked processes is empty;  
4316 otherwise, the size of the set is equal to the absolute value of the semaphore value. The value  
4317 of the semaphore can be incremented or decremented by any process with access to the  
4318 semaphore and must be done as an indivisible operation. When a semaphore value is less  
4319 than or equal to zero, any process that attempts to lock it again will block or be informed that  
4320 it is not possible to perform the operation.
- 4321 A semaphore may be used to guard access to any resource accessible by more than one  
4322 schedulable task in the system. It is a global entity and not associated with any particular  
4323 process. As such, a method of obtaining access to the semaphore has to be provided by the  
4324 operating system. A process that wants access to a critical resource (section) has to wait on  
4325 the semaphore that guards that resource. When the semaphore is locked on behalf of a  
4326 process, it knows that it can utilize the resource without interference by any other  
4327 cooperating process in the system. When the process finishes its operation on the resource,  
4328 leaving it in a well-defined state, it posts the semaphore, indicating that some other process  
4329 may now obtain the resource associated with that semaphore.
- 4330 In this section, mutexes and condition variables are specified as the synchronization  
4331 mechanisms between threads.
- 4332 These primitives are typically used for synchronizing threads that share memory in a single  
4333 process. However, this section provides an option allowing the use of these synchronization  
4334 interfaces and objects between processes that share memory, regardless of the method for  
4335 sharing memory.
- 4336 Much experience with semaphores shows that there are two distinct uses of synchronization:  
4337 locking, which is typically of short duration; and waiting, which is typically of long or  
4338 unbounded duration. These distinct usages map directly onto mutexes and condition  
4339 variables, respectively.
- 4340 Semaphores are provided in IEEE Std 1003.1-200x primarily to provide a means of  
4341 synchronization for processes; these processes may or may not share memory. Mutexes and  
4342 condition variables are specified as synchronization mechanisms between threads; these  
4343 threads always share (some) memory. Both are synchronization paradigms that have been in  
4344 widespread use for a number of years. Each set of primitives is particularly well matched to  
4345 certain problems.
- 4346 With respect to binary semaphores, experience has shown that condition variables and  
4347 mutexes are easier to use for many synchronization problems than binary semaphores. The

4348 primary reason for this is the explicit appearance of a Boolean predicate that specifies when  
4349 the condition wait is satisfied. This Boolean predicate terminates a loop, including the call to  
4350 *pthread\_cond\_wait()*. As a result, extra wakeups are benign since the predicate governs  
4351 whether the thread will actually proceed past the condition wait. With stateful primitives,  
4352 such as binary semaphores, the wakeup in itself typically means that the wait is satisfied. The  
4353 burden of ensuring correctness for such waits is thus placed on *all* signalers of the semaphore  
4354 rather than on an *explicitly coded* Boolean predicate located at the condition wait. Experience  
4355 has shown that the latter creates a major improvement in safety and ease-of-use.

4356 Counting semaphores are well matched to dealing with producer/consumer problems,  
4357 including those that might exist between threads of different processes, or between a signal  
4358 handler and a thread. In the former case, there may be little or no memory shared by the  
4359 processes; in the latter case, one is not communicating between co-equal threads, but  
4360 between a thread and an interruptlike entity. It is for these reasons that IEEE Std 1003.1-200x  
4361 allows semaphores to be used by threads.

4362 Mutexes and condition variables have been effectively used with and without priority  
4363 inheritance, priority ceiling, and other attributes to synchronize threads that share memory.  
4364 The efficiency of their implementation is comparable to or better than that of other  
4365 synchronization primitives that are sometimes harder to use (for example, binary  
4366 semaphores). Furthermore, there is at least one known implementation of Ada tasking that  
4367 uses these primitives. Mutexes and condition variables together constitute an appropriate,  
4368 sufficient, and complete set of interthread synchronization primitives.

4369 Efficient multi-threaded applications require high-performance synchronization primitives.  
4370 Considerations of efficiency and generality require a small set of primitives upon which more  
4371 sophisticated synchronization functions can be built.

#### 4372 • Standardization Issues

4373 It is possible to implement very high-performance semaphores using test-and-set  
4374 instructions on shared memory locations. The library routines that implement such a high-  
4375 performance interface has to properly ensure that a *sem\_wait()* or *sem\_trywait()* operation  
4376 that cannot be performed will issue a blocking semaphore system call or properly report the  
4377 condition to the application. The same interface to the application program would be  
4378 provided by a high-performance implementation.

### 4379 *B.2.8.1 Realtime Signals*

#### 4380 **Realtime Signals Extension**

4381 This portion of the rationale presents models, requirements, and standardization issues relevant  
4382 to the Realtime Signals Extension. This extension provides the capability required to support  
4383 reliable, deterministic, asynchronous notification of events. While a new mechanism,  
4384 unencumbered by the historical usage and semantics of POSIX.1 signals, might allow for a more  
4385 efficient implementation, the application requirements for event notification can be met with a  
4386 small number of extensions to signals. Therefore, a minimal set of extensions to signals to  
4387 support the application requirements is specified.

4388 The realtime signal extensions specified in this section are used by other realtime functions  
4389 requiring asynchronous notification:

#### 4390 • Models

4391 The model supported is one of multiple cooperating processes, each of which handles  
4392 multiple asynchronous external events. Events represent occurrences that are generated as

- 4393 the result of some activity in the system. Examples of occurrences that can constitute an  
4394 event include:
- 4395 — Completion of an asynchronous I/O request
  - 4396 — Expiration of a POSIX.1b timer
  - 4397 — Arrival of an interprocess message
  - 4398 — Generation of a user-defined event
- 4399 Processing of these events may occur synchronously via polling for event notifications or  
4400 asynchronously via a software interrupt mechanism. Existing practice for this model is well  
4401 established for traditional proprietary realtime operating systems, realtime executives, and  
4402 realtime extended POSIX-like systems.
- 4403 A contrasting model is that of “cooperating sequential processes” where each process  
4404 handles a single priority of events via polling. Each process blocks while waiting for events,  
4405 and each process depends on the preemptive, priority-based process scheduling mechanism  
4406 to arbitrate between events of different priority that need to be processed concurrently.  
4407 Existing practice for this model is also well established for small realtime executives that  
4408 typically execute in an unprotected physical address space, but it is just emerging in the  
4409 context of a fuller function operating system with multiple virtual address spaces.
- 4410 It could be argued that the cooperating sequential process model, and the facilities supported  
4411 by the POSIX Threads Extension obviate a software interrupt model. But, even with the  
4412 cooperating sequential process model, the need has been recognized for a software interrupt  
4413 model to handle exceptional conditions and process aborting, so the mechanism must be  
4414 supported in any case. Furthermore, it is not the purview of IEEE Std 1003.1-200x to attempt  
4415 to convince realtime practitioners that their current application models based on software  
4416 interrupts are “broken” and should be replaced by the cooperating sequential process model.  
4417 Rather, it is the charter of IEEE Std 1003.1-200x to provide standard extensions to  
4418 mechanisms that support existing realtime practice.
- 4419 • Requirements
- 4420 This section discusses the following realtime application requirements for asynchronous  
4421 event notification:
- 4422 — Reliable delivery of asynchronous event notification
- 4423 The events notification mechanism shall guarantee delivery of an event notification.  
4424 Asynchronous operations (such as asynchronous I/O and timers) that complete  
4425 significantly after they are invoked have to guarantee that delivery of the event  
4426 notification can occur at the time of completion.
- 4427 — Prioritized handling of asynchronous event notifications
- 4428 The events notification mechanism shall support the assigning of a user function as an  
4429 event notification handler. Furthermore, the mechanism shall support the preemption of  
4430 an event handler function by a higher priority event notification and shall support the  
4431 selection of the highest priority pending event notification when multiple notifications (of  
4432 different priority) are pending simultaneously.
- 4433 The model here is based on hardware interrupts. Asynchronous event handling allows  
4434 the application to ensure that time-critical events are immediately processed when  
4435 delivered, without the indeterminism of being at a random location within a polling loop.  
4436 Use of handler priority allows the specification of how handlers are interrupted by other  
4437 higher priority handlers.

- 4438 — Differentiation between multiple occurrences of event notifications of the same type
- 4439 The events notification mechanism shall pass an application-defined value to the event  
4440 handler function. This value can be used for a variety of purposes, such as enabling the  
4441 application to identify which of several possible events of the same type (for example,  
4442 timer expirations) has occurred.
- 4443 — Polled reception of asynchronous event notifications
- 4444 The events notification mechanism shall support blocking and non-blocking polls for  
4445 asynchronous event notification.
- 4446 The polled mode of operation is often preferred over the interrupt mode by those  
4447 practitioners accustomed to this model. Providing support for this model facilitates the  
4448 porting of applications based on this model to POSIX.1b conforming systems.
- 4449 — Deterministic response to asynchronous event notifications
- 4450 The events notification mechanism shall not preclude implementations that provide  
4451 deterministic event dispatch latency and shall minimize the number of system calls  
4452 needed to use the event facilities during realtime processing.
- 4453 • Rationale for Extension
- 4454 POSIX.1 signals have many of the characteristics necessary to support the asynchronous  
4455 handling of event notifications, and the Realtime Signals Extension addresses the following  
4456 deficiencies in the POSIX.1 signal mechanism:
- 4457 — Signals do not support reliable delivery of event notification. Subsequent occurrences of  
4458 a pending signal are not guaranteed to be delivered.
- 4459 — Signals do not support prioritized delivery of event notifications. The order of signal  
4460 delivery when multiple unblocked signals are pending is undefined.
- 4461 — Signals do not support the differentiation between multiple signals of the same type.

#### 4462 *B.2.8.2 Asynchronous I/O*

4463 Many applications need to interact with the I/O subsystem in an asynchronous manner. The  
4464 asynchronous I/O mechanism provides the ability to overlap application processing and I/O  
4465 operations initiated by the application. The asynchronous I/O mechanism allows a single  
4466 process to perform I/O simultaneously to a single file multiple times or to multiple files  
4467 multiple times.

#### 4468 **Overview**

4469 Asynchronous I/O operations proceed in logical parallel with the processing done by the  
4470 application after the asynchronous I/O has been initiated. Other than this difference,  
4471 asynchronous I/O behaves similarly to normal I/O using *read()*, *write()*, *lseek()*, and *fsync()*.  
4472 The effect of issuing an asynchronous I/O request is as if a separate thread of execution were to  
4473 perform atomically the implied *lseek()* operation, if any, and then the requested I/O operation  
4474 (either *read()*, *write()*, or *fsync()*). There is no seek implied with a call to *aio\_fsync()*. Concurrent  
4475 asynchronous operations and synchronous operations applied to the same file update the file as  
4476 if the I/O operations had proceeded serially.

4477 When asynchronous I/O completes, a signal can be delivered to the application to indicate the  
4478 completion of the I/O. This signal can be used to indicate that buffers and control blocks used  
4479 for asynchronous I/O can be reused. Signal delivery is not required for an asynchronous  
4480 operation and may be turned off on a per-operation basis by the application. Signals may also be

4481 synchronously polled using *aio\_suspend()*, *sigtimedwait()*, or *sigwaitinfo()*.

4482 Normal I/O has a return value and an error status associated with it. Asynchronous I/O returns  
4483 a value and an error status when the operation is first submitted, but that only relates to whether  
4484 the operation was successfully queued up for servicing. The I/O operation itself also has a  
4485 return status and an error value. To allow the application to retrieve the return status and the  
4486 error value, functions are provided that, given the address of an asynchronous I/O control  
4487 block, yield the return and error status associated with the operation. Until an asynchronous I/O  
4488 operation is done, its error status shall be [EINPROGRESS]. Thus, an application can poll for  
4489 completion of an asynchronous I/O operation by waiting for the error status to become equal to  
4490 a value other than [EINPROGRESS]. The return status of an asynchronous I/O operation is  
4491 undefined so long as the error status is equal to [EINPROGRESS].

4492 Storage for asynchronous operation return and error status may be limited. Submission of  
4493 asynchronous I/O operations may fail if this storage is exceeded. When an application retrieves  
4494 the return status of a given asynchronous operation, therefore, any system-maintained storage  
4495 used for this status and the error status may be reclaimed for use by other asynchronous  
4496 operations.

4497 Asynchronous I/O can be performed on file descriptors that have been enabled for POSIX.1b  
4498 synchronized I/O. In this case, the I/O operation still occurs asynchronously, as defined herein;  
4499 however, the asynchronous operation I/O in this case is not completed until the I/O has reached  
4500 either the state of synchronized I/O data integrity completion or synchronized I/O file integrity  
4501 completion, depending on the sort of synchronized I/O that is enabled on the file descriptor.

## 4502 **Models**

4503 Three models illustrate the use of asynchronous I/O: a journalization model, a data acquisition  
4504 model, and a model of the use of asynchronous I/O in supercomputing applications.

- 4505 • **Journalization Model**

4506 Many realtime applications perform low-priority journalizing functions. Journalizing  
4507 requires that logging records be queued for output without blocking the initiating process.

- 4508 • **Data Acquisition Model**

4509 A data acquisition process may also serve as a model. The process has two or more channels  
4510 delivering intermittent data that must be read within a certain time. The process issues one  
4511 asynchronous read on each channel. When one of the channels needs data collection, the  
4512 process reads the data and posts it through an asynchronous write to secondary memory for  
4513 future processing.

- 4514 • **Supercomputing Model**

4515 The supercomputing community has used asynchronous I/O much like that specified herein  
4516 for many years. This community requires the ability to perform multiple I/O operations to  
4517 multiple devices with a minimal number of entries to “the system”; each entry to “the  
4518 system” provokes a major delay in operations when compared to the normal progress made  
4519 by the application. This existing practice motivated the use of combined *lseek()* and *read()* or  
4520 *write()* calls, as well as the *lio\_listio()* call. Another common practice is to disable signal  
4521 notification for I/O completion, and simply poll for I/O completion at some interval by  
4522 which the I/O should be completed. Likewise, interfaces like *aio\_cancel()* have been in  
4523 successful commercial use for many years. Note also that an underlying implementation of  
4524 asynchronous I/O will require the ability, at least internally, to cancel outstanding  
4525 asynchronous I/O, at least when the process exits. (Consider an asynchronous read from a  
4526 terminal, when the process intends to exit immediately.)

4527 **Requirements**

4528 Asynchronous input and output for realtime implementations have these requirements:

- 4529 • The ability to queue multiple asynchronous read and write operations to a single open  
4530 instance. Both sequential and random access should be supported.
- 4531 • The ability to queue asynchronous read and write operations to multiple open instances.
- 4532 • The ability to obtain completion status information by polling and/or asynchronous event  
4533 notification.
- 4534 • Asynchronous event notification on asynchronous I/O completion is optional.
- 4535 • It has to be possible for the application to associate the event with the *aiocbp* for the operation  
4536 that generated the event.
- 4537 • The ability to cancel queued requests.
- 4538 • The ability to wait upon asynchronous I/O completion in conjunction with other types of  
4539 events.
- 4540 • The ability to accept an *aio\_read()* and an *aio\_cancel()* for a device that accepts a *read()*, and  
4541 the ability to accept an *aio\_write()* and an *aio\_cancel()* for a device that accepts a *write()*. This  
4542 does not imply that the operation is asynchronous.

4543 **Standardization Issues**

4544 The following issues are addressed by the standardization of asynchronous I/O:

## 4545 • Rationale for New Interface

4546 Non-blocking I/O does not satisfy the needs of either realtime or high-performance  
4547 computing models; these models require that a process overlap program execution and I/O  
4548 processing. Realtime applications will often make use of direct I/O to or from the address  
4549 space of the process, or require synchronized (unbuffered) I/O; they also require the ability  
4550 to overlap this I/O with other computation. In addition, asynchronous I/O allows an  
4551 application to keep a device busy at all times, possibly achieving greater throughput.  
4552 Supercomputing and database architectures will often have specialized hardware that can  
4553 provide true asynchrony underlying the logical asynchrony provided by this interface. In  
4554 addition, asynchronous I/O should be supported by all types of files and devices in the same  
4555 manner.

## 4556 • Effect of Buffering

4557 If asynchronous I/O is performed on a file that is buffered prior to being actually written to  
4558 the device, it is possible that asynchronous I/O will offer no performance advantage over  
4559 normal I/O; the cycles *stolen* to perform the asynchronous I/O will be taken away from the  
4560 running process and the I/O will occur at interrupt time. This potential lack of gain in  
4561 performance in no way obviates the need for asynchronous I/O by realtime applications,  
4562 which very often will use specialized hardware support; multiple processors; and/or  
4563 unbuffered, synchronized I/O.

4564 **B.2.8.3** *Memory Management*

4565 All memory management and shared memory definitions are located in the `<sys/mman.h>`  
4566 header. This is for alignment with historical practice.

4567 **Memory Locking Functions**

4568 This portion of the rationale presents models, requirements, and standardization issues relevant  
4569 to process memory locking.

## 4570 • Models

4571 Realtime systems that conform to IEEE Std 1003.1-200x are expected (and desired) to be  
4572 supported on systems with demand-paged virtual memory management, non-paged  
4573 swapping memory management, and physical memory systems with no memory  
4574 management hardware. The general case, however, is the demand-paged, virtual memory  
4575 system with each POSIX process running in a virtual address space. Note that this includes  
4576 architectures where each process resides in its own virtual address space and architectures  
4577 where the address space of each process is only a portion of a larger global virtual address  
4578 space.

4579 The concept of memory locking is introduced to eliminate the indeterminacy introduced by  
4580 paging and swapping, and to support an upper bound on the time required to access the  
4581 memory mapped into the address space of a process. Ideally, this upper bound will be the  
4582 same as the time required for the processor to access “main memory”, including any address  
4583 translation and cache miss overheads. But some implementations—primarily on  
4584 mainframes—will not actually force locked pages to be loaded and held resident in main  
4585 memory. Rather, they will handle locked pages so that accesses to these pages will meet the  
4586 performance metrics for locked process memory in the implementation. Also, although it is  
4587 not, for example, the intention that this interface, as specified, be used to lock process  
4588 memory into “cache”, it is conceivable that an implementation could support a large static  
4589 RAM memory and define this as “main memory” and use a large[r] dynamic RAM as  
4590 “backing store”. These interfaces could then be interpreted as supporting the locking of  
4591 process memory into the static RAM. Support for multiple levels of backing store would  
4592 require extensions to these interfaces.

4593 Implementations may also use memory locking to guarantee a fixed translation between  
4594 virtual and physical addresses where such is beneficial to improving determinacy for  
4595 direct-to/from-process input/output. IEEE Std 1003.1-200x does not guarantee to the  
4596 application that the virtual-to-physical address translations, if such exist, are fixed, because  
4597 such behavior would not be implementable on all architectures on which implementations of  
4598 IEEE Std 1003.1-200x are expected. But IEEE Std 1003.1-200x does mandate that an  
4599 implementation define, for the benefit of potential users, whether or not locking guarantees  
4600 fixed translations.

4601 Memory locking is defined with respect to the address space of a process. Only the pages  
4602 mapped into the address space of a process may be locked by the process, and when the  
4603 pages are no longer mapped into the address space—for whatever reason—the locks  
4604 established with respect to that address space are removed. Shared memory areas warrant  
4605 special mention, as they may be mapped into more than one address space or mapped more  
4606 than once into the address space of a process; locks may be established on pages within these  
4607 areas with respect to several of these mappings. In such a case, the lock state of the  
4608 underlying physical pages is the logical OR of the lock state with respect to each of the  
4609 mappings. Only when all such locks have been removed are the shared pages considered  
4610 unlocked.

4611 In recognition of the page granularity of Memory Management Units (MMU), and in order to  
 4612 support locking of ranges of address space, memory locking is defined in terms of “page”  
 4613 granularity. That is, for the interfaces that support an address and size specification for the  
 4614 region to be locked, the address must be on a page boundary, and all pages mapped by the  
 4615 specified range are locked, if valid. This means that the length is implicitly rounded up to a  
 4616 multiple of the page size. The page size is implementation-defined and is available to  
 4617 applications as a compile time symbolic constant or at runtime via *sysconf()*.

4618 A “real memory” POSIX.1b implementation that has no MMU could elect not to support  
 4619 these interfaces, returning [ENOSYS]. But an application could easily interpret this as  
 4620 meaning that the implementation would unconditionally page or swap the application when  
 4621 such is not the case. It is the intention of IEEE Std 1003.1-200x that such a system could define  
 4622 these interfaces as “NO-OPs”, returning success without actually performing any function  
 4623 except for mandated argument checking.

#### 4624 • Requirements

4625 For realtime applications, memory locking is generally considered to be required as part of  
 4626 application initialization. This locking is performed after an application has been loaded (that  
 4627 is, *exec'd*) and the program remains locked for its entire lifetime. But to support applications  
 4628 that undergo major mode changes where, in one mode, locking is required, but in another it  
 4629 is not, the specified interfaces allow repeated locking and unlocking of memory within the  
 4630 lifetime of a process.

4631 When a realtime application locks its address space, it should not be necessary for the  
 4632 application to then “touch” all of the pages in the address space to guarantee that they are  
 4633 resident or else suffer potential paging delays the first time the page is referenced. Thus,  
 4634 IEEE Std 1003.1-200x requires that the pages locked by the specified interfaces be resident  
 4635 when the locking functions return successfully.

4636 Many architectures support system-managed stacks that grow automatically when the  
 4637 current extent of the stack is exceeded. A realtime application has a requirement to be able to  
 4638 “preallocate” sufficient stack space and lock it down so that it will not suffer page faults to  
 4639 grow the stack during critical realtime operation. There was no consensus on a portable way  
 4640 to specify how much stack space is needed, so IEEE Std 1003.1-200x supports no specific  
 4641 interface for preallocating stack space. But an application can portably lock down a specific  
 4642 amount of stack space by specifying *MCL\_FUTURE* in a call to *memlockall()* and then calling  
 4643 a dummy function that declares an automatic array of the desired size.

4644 Memory locking for realtime applications is also generally considered to be an “all or  
 4645 nothing” proposition. That is, the entire process, or none, is locked down. But, for  
 4646 applications that have well-defined sections that need to be locked and others that do not,  
 4647 IEEE Std 1003.1-200x supports an optional set of interfaces to lock or unlock a range of  
 4648 process addresses. Reasons for locking down a specific range include:

- 4649 — An asynchronous event handler function that must respond to external events in a  
 4650 deterministic manner such that page faults cannot be tolerated
- 4651 — An input/output “buffer” area that is the target for direct-to-process I/O, and the  
 4652 overhead of implicit locking and unlocking for each I/O call cannot be tolerated

4653 Finally, locking is generally viewed as an “application-wide” function. That is, the  
 4654 application is globally aware of which regions are locked and which are not over time. This is  
 4655 in contrast to a function that is used temporarily within a “third party” library routine whose  
 4656 function is unknown to the application, and therefore must have no “side effects”. The  
 4657 specified interfaces, therefore, do not support “lock stacking” or “lock nesting” within a  
 4658 process. But, for pages that are shared between processes or mapped more than once into a

4659 process address space, “lock stacking” is essentially mandated by the requirement that  
4660 unlocking of pages that are mapped by more than one process or more than once by the same  
4661 process does not affect locks established on the other mappings.

4662 There was some support for “lock stacking” so that locking could be transparently used in  
4663 functions or opaque modules. But the consensus was not to burden all implementations with  
4664 lock stacking (and reference counting), and an implementation option was proposed. There  
4665 were strong objections to the option because applications would have to support both  
4666 options in order to remain portable. The consensus was to eliminate lock stacking altogether,  
4667 primarily through overwhelming support for the System V “m[un]lock[all]” interface on  
4668 which IEEE Std 1003.1-200x is now based.

4669 Locks are not inherited across *fork()*s because some implementations implement *fork()* by  
4670 creating new address spaces for the child. In such an implementation, requiring locks to be  
4671 inherited would lead to new situations in which a fork would fail due to the inability of the  
4672 system to lock sufficient memory to lock both the parent and the child. The consensus was  
4673 that there was no benefit to such inheritance. Note that this does not mean that locks are  
4674 removed when, for instance, a thread is created in the same address space.

4675 Similarly, locks are not inherited across *exec* because some implementations implement *exec*  
4676 by unmapping all of the pages in the address space (which, by definition, removes the locks  
4677 on these pages), and maps in pages of the *exec*'d image. In such an implementation, requiring  
4678 locks to be inherited would lead to new situations in which *exec* would fail. Reporting this  
4679 failure would be very cumbersome to detect in time to report to the calling process, and no  
4680 appropriate mechanism exists for informing the *exec*'d process of its status.

4681 It was determined that, if the newly loaded application required locking, it was the  
4682 responsibility of that application to establish the locks. This is also in keeping with the  
4683 general view that it is the responsibility of the application to be aware of all locks that are  
4684 established.

4685 There was one request to allow (not mandate) locks to be inherited across *fork()*, and a  
4686 request for a flag, MCL\_INHERIT, that would specify inheritance of memory locks across  
4687 *exec*s. Given the difficulties raised by this and the general lack of support for the feature in  
4688 IEEE Std 1003.1-200x, it was not added. IEEE Std 1003.1-200x does not preclude an  
4689 implementation from providing this feature for administrative purposes, such as a “run”  
4690 command that will lock down and execute specified program. Additionally, the rationale for  
4691 the objection equated *fork()* with creating a thread in the address space. IEEE Std 1003.1-200x  
4692 does not mandate releasing locks when creating additional threads in an existing process.

#### 4693 • Standardization Issues

4694 One goal of IEEE Std 1003.1-200x is to define a set of primitives that provide the necessary  
4695 functionality for realtime applications, with consideration for the needs of other application  
4696 domains where such were identified, which is based to the extent possible on existing  
4697 industry practice.

4698 The Memory Locking option is required by many realtime applications to tune performance.  
4699 Such a facility is accomplished by placing constraints on the virtual memory system to limit  
4700 paging of time of the process or of critical sections of the process. This facility should not be  
4701 used by most non-realtime applications.

4702 Optional features provided in IEEE Std 1003.1-200x allow applications to lock selected  
4703 address ranges with the caveat that the process is responsible for being aware of the page  
4704 granularity of locking and the un-nested nature of the locks.

4705 **Mapped Files Functions**

4706 The Memory Mapped Files option provides a mechanism that allows a process to access files by  
4707 directly incorporating file data into its address space. Once a file is “mapped” into a process  
4708 address space, the data can be manipulated by instructions as memory. The use of mapped files  
4709 can significantly reduce I/O data movement since file data does not have to be copied into  
4710 process data buffers as in *read()* and *write()*. If more than one process maps a file, its contents  
4711 are shared among them. This provides a low overhead mechanism by which processes can  
4712 synchronize and communicate.

## 4713 • Historical Perspective

4714 Realtime applications have historically been implemented using a collection of cooperating  
4715 processes or tasks. In early systems, these processes ran on bare hardware (that is, without an  
4716 operating system) with no memory relocation or protection. The application paradigms that  
4717 arose from this environment involve the sharing of data between the processes.

4718 When realtime systems were implemented on top of vendor-supplied operating systems, the  
4719 paradigm or performance benefits of direct access to data by multiple processes was still  
4720 deemed necessary. As a result, operating systems that claim to support realtime applications  
4721 must support the shared memory paradigm.

4722 Additionally, a number of realtime systems provide the ability to map specific sections of the  
4723 physical address space into the address space of a process. This ability is required if an  
4724 application is to obtain direct access to memory locations that have specific properties (for  
4725 example, refresh buffers or display devices, dual ported memory locations, DMA target  
4726 locations). The use of this ability is common enough to warrant some degree of  
4727 standardization of its interface. This ability overlaps the general paradigm of shared  
4728 memory in that, in both instances, common global objects are made addressable by  
4729 individual processes or tasks.

4730 Finally, a number of systems also provide the ability to map process addresses to files. This  
4731 provides both a general means of sharing persistent objects, and using files in a manner that  
4732 optimizes memory and swapping space usage.

4733 Simple shared memory is clearly a special case of the more general file mapping capability.  
4734 In addition, there is relatively widespread agreement and implementation of the file  
4735 mapping interface. In these systems, many different types of objects can be mapped (for  
4736 example, files, memory, devices, and so on) using the same mapping interfaces. This  
4737 approach both minimizes interface proliferation and maximizes the generality of programs  
4738 using the mapping interfaces.

## 4739 • Memory Mapped Files Usage

4740 A memory object can be concurrently mapped into the address space of one or more  
4741 processes. The *mmap()* and *munmap()* functions allow a process to manipulate their address  
4742 space by mapping portions of memory objects into it and removing them from it. When  
4743 multiple processes map the same memory object, they can share access to the underlying  
4744 data. Implementations may restrict the size and alignment of mappings to be on *page*-size  
4745 boundaries. The page size, in bytes, is the value of the system-configurable variable  
4746 {PAGESIZE}, typically accessed by calling *sysconf()* with a *name* argument of  
4747 \_SC\_PAGESIZE. If an implementation has no restrictions on size or alignment, it may  
4748 specify a 1-byte page size.

4749 To map memory, a process first opens a memory object. The *ftruncate()* function can be used  
4750 to contract or extend the size of the memory object even when the object is currently  
4751 mapped. If the memory object is extended, the contents of the extended areas are zeros.

4752 After opening a memory object, the application maps the object into its address space using  
4753 the *mmap()* function call. Once a mapping has been established, it remains mapped until  
4754 unmapped with *munmap()*, even if the memory object is closed. The *mprotect()* function can  
4755 be used to change the memory protections initially established by *mmap()*.

4756 A *close()* of the file descriptor, while invalidating the file descriptor itself, does not unmap  
4757 any mappings established for the memory object. The address space, including all mapped  
4758 regions, is inherited on *fork()*. The entire address space is unmapped on process termination  
4759 or by successful calls to any of the *exec* family of functions.

4760 The *msync()* function is used to force mapped file data to permanent storage.

4761 • Effects on Other Functions

4762 When the Memory Mapped Files option is supported, the operation of the *open()*, *creat()*, and  
4763 *unlink()* functions are a natural result of using the file system name space to map the global  
4764 names for memory objects.

4765 The *ftruncate()* function can be use to set the length of a sharable memory object.

4766 The meaning of *stat()* fields other than the size and protection information is undefined on  
4767 implementations where memory objects are not implemented using regular files. When  
4768 regular files are used, the times reflect when the implementation updated the file image of  
4769 the data, not when a process updated the data in memory.

4770 The operations of *fdopen()*, *write()*, *read()*, and *lseek()* were made unspecified for objects  
4771 opened with *shm\_open()*, so that implementations that did not implement memory objects as  
4772 regular files would not have to support the operation of these functions on shared memory  
4773 objects.

4774 The behavior of memory objects with respect to *close()*, *dup()*, *dup2()*, *open()*, *close()*, *fork()*,  
4775 *\_exit()*, and the *exec* family of functions is the same as the behavior of the existing practice of  
4776 the *mmap()* function.

4777 A memory object can still be referenced after a close. That is, any mappings made to the file  
4778 are still in effect, and reads and writes that are made to those mappings are still valid and are  
4779 shared with other processes that have the same mapping. Likewise, the memory object can  
4780 still be used if any references remain after its name(s) have been deleted. Any references that  
4781 remain after a close must not appear to the application as file descriptors.

4782 This is existing practice for *mmap()* and *close()*. In addition, there are already mappings  
4783 present (text, data, stack) that do not have open file descriptors. The text mapping in  
4784 particular is considered a reference to the file containing the text. The desire was to treat all  
4785 mappings by the process uniformly. Also, many modern implementations use *mmap()* to  
4786 implement shared libraries, and it would not be desirable to keep file descriptors for each of  
4787 the many libraries an application can use. It was felt there were many other existing  
4788 programs that used this behavior to free a file descriptor, and thus IEEE Std 1003.1-200x  
4789 could not forbid it and still claim to be using existing practice.

4790 For implementations that implement memory objects using memory only, memory objects  
4791 will retain the memory allocated to the file after the last close and will use that same memory  
4792 on the next open. Note that closing the memory object is not the same as deleting the name,  
4793 since the memory object is still defined in the memory object name space.

4794 The locks of *fcntl()* do not block any read or write operation, including read or write access to  
4795 shared memory or mapped files. In addition, implementations that only support shared  
4796 memory objects should not be required to implement record locks. The reference to *fcntl()* is  
4797 added to make this point explicitly. The other *fcntl()* commands are useful with shared

4798 memory objects.

4799 The size of pages that mapping hardware may be able to support may be a configurable  
4800 value, or it may change based on hardware implementations. The addition of the  
4801 `_SC_PAGESIZE` parameter to the `sysconf()` function is provided for determining the mapping  
4802 page size at runtime.

### 4803 **Shared Memory Functions**

4804 Implementations may support the Shared Memory Objects option without supporting a general  
4805 Memory Mapped Files option. Shared memory objects are named regions of storage that may be  
4806 independent of the file system and can be mapped into the address space of one or more  
4807 processes to allow them to share the associated memory.

#### 4808 • Requirements

4809 Shared memory is used to share data among several processes, each potentially running at  
4810 different priority levels, responding to different inputs, or performing separate tasks. Shared  
4811 memory is not just simply providing common access to data, it is providing the fastest  
4812 possible communication between the processes. With one memory write operation, a process  
4813 can pass information to as many processes as have the memory region mapped.

4814 As a result, shared memory provides a mechanism that can be used for all other interprocess  
4815 communications facilities. It may also be used by an application for implementing more  
4816 sophisticated mechanisms than semaphores and message queues.

4817 The need for a shared memory interface is obvious for virtual memory systems, where the  
4818 operating system is directly preventing processes from accessing each other's data. However,  
4819 in unprotected systems, such as those found in some embedded controllers, a shared  
4820 memory interface is needed to provide a portable mechanism to allocate a region of memory  
4821 to be shared and then to communicate the address of that region to other processes.

4822 This, then, provides the minimum functionality that a shared memory interface must have in  
4823 order to support realtime applications: to allocate and name an object to be mapped into  
4824 memory for potential sharing (`open()` or `shm_open()`), and to make the memory object  
4825 available within the address space of a process (`mmap()`). To complete the interface, a  
4826 mechanism to release the claim of a process on a shared memory object (`munmap()`) is also  
4827 needed, as well as a mechanism for deleting the name of a sharable object that was  
4828 previously created (`unlink()` or `shm_unlink()`).

4829 After a mapping has been established, an implementation should not have to provide  
4830 services to maintain that mapping. All memory writes into that area will appear immediately  
4831 in the memory mapping of that region by any other processes.

4832 Thus, requirements include:

4833 — Support creation of sharable memory objects and the mapping of these objects into the  
4834 address space of a process.

4835 — Sharable memory objects should be accessed by global names accessible from all  
4836 processes.

4837 — Support the mapping of specific sections of physical address space (such as a memory  
4838 mapped device) into the address space of a process. This should not be done by the  
4839 process specifying the actual address, but again by an implementation-defined global  
4840 name (such as a special device name) dedicated to this purpose.

4841 — Support the mapping of discrete portions of these memory objects.

- 4842 — Support for minimum hardware configurations that contain no physical media on which  
4843 to store shared memory contents permanently.
- 4844 — The ability to preallocate the entire shared memory region so that minimum hardware  
4845 configurations without virtual memory support can guarantee contiguous space.
- 4846 — The maximizing of performance by not requiring functionality that would require  
4847 implementation interaction above creating the shared memory area and returning the  
4848 mapping.
- 4849 Note that the above requirements do not preclude:
- 4850 — The sharable memory object from being implemented using actual files on an actual file  
4851 system.
- 4852 — The global name that is accessible from all processes being restricted to a file system area  
4853 that is dedicated to handling shared memory.
- 4854 — An implementation not providing implementation-defined global names for the purpose  
4855 of physical address mapping.
- 4856 • Shared Memory Objects Usage
- 4857 If the Shared Memory Objects option is supported, a shared memory object may be created,  
4858 or opened if it already exists, with the *shm\_open()* function. If the shared memory object is  
4859 created, it has a length of zero. The *truncate()* function can be used to set the size of the  
4860 shared memory object after creation. The *shm\_unlink()* function removes the name for a  
4861 shared memory object created by *shm\_open()*.
- 4862 • Shared Memory Overview
- 4863 The shared memory facility defined by IEEE Std 1003.1-200x usually results in memory  
4864 locations being added to the address space of the process. The implementation returns the  
4865 address of the new space to the application by means of a pointer. This works well in  
4866 languages like C. However, in languages without pointer types it will not work. In the  
4867 bindings for such a language, either a special COMMON section will need to be defined  
4868 (which is unlikely), or the binding will have to allow existing structures to be mapped. The  
4869 implementation will likely have to place restrictions on the size and alignment of such  
4870 structures or will have to map a suitable region of the address space of the process into the  
4871 memory object, and thus into other processes. These are issues for that particular language  
4872 binding. For IEEE Std 1003.1-200x, however, the practice will not be forbidden, merely  
4873 undefined.
- 4874 Two potentially different name spaces are used for naming objects that may be mapped into  
4875 process address spaces. When the Memory Mapped Files option is supported, files may be  
4876 accessed via *open()*. When the Shared Memory Objects option is supported, sharable  
4877 memory objects that might not be files may be accessed via the *shm\_open()* function. These  
4878 options are not mutually-exclusive.
- 4879 Some implementations supporting the Shared Memory Objects option may choose to |  
4880 implement the shared memory object name space as part of the file system name space. |  
4881 There are several reasons for this: |
- 4882 — It allows applications to prevent name conflicts by use of the directory structure.
- 4883 — It uses an existing mechanism for accessing global objects and prevents the creation of a  
4884 new mechanism for naming global objects.
- 4885 In such implementations, memory objects can be implemented using regular files, if that is  
4886 what the implementation chooses. The *shm\_open()* function can be implemented as an *open()*

4887 call in a fixed directory followed by a call to *fcntl()* to set *FD\_CLOEXEC*. The *shm\_unlink()*  
4888 function can be implemented as an *unlink()* call.

4889 On the other hand, it is also expected that small embedded systems that support the Shared  
4890 Memory Objects option may wish to implement shared memory without having any file  
4891 systems present. In this case, the implementations may choose to use a simple string valued  
4892 name space for shared memory regions. The *shm\_open()* function permits either type of  
4893 implementation.

4894 Some implementations have hardware that supports protection of mapped data from certain |  
4895 classes of access and some do not. Systems that supply this functionality can support the |  
4896 Memory Protection option. |

4897 Some implementations restrict size, alignment, and protections to be on *page*-size  
4898 boundaries. If an implementation has no restrictions on size or alignment, it may specify a 1-  
4899 byte page size. Applications on implementations that do support larger pages must be  
4900 cognizant of the page size since this is the alignment and protection boundary.

4901 Simple embedded implementations may have a 1-byte page size and only support the Shared  
4902 Memory Objects option. This provides simple shared memory between processes without  
4903 requiring mapping hardware.

4904 IEEE Std 1003.1-200x specifically allows a memory object to remain referenced after a close  
4905 because that is existing practice for the *mmap()* function.

#### 4906 **Typed Memory Functions**

4907 Implementations may support the Typed Memory Objects option without supporting either the  
4908 Shared Memory option or the Memory Mapped Files option. Typed memory objects are pools of  
4909 specialized storage, different from the main memory resource normally used by a processor to  
4910 hold code and data, that can be mapped into the address space of one or more processes.

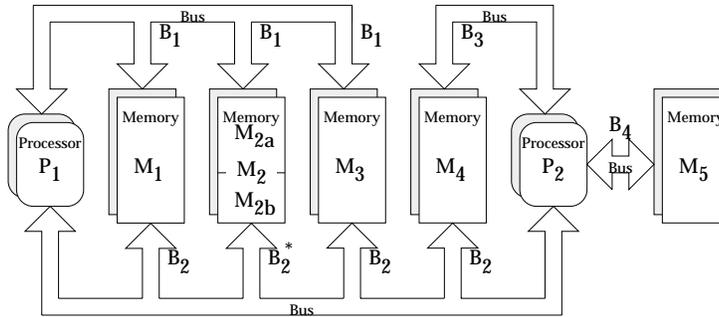
##### 4911 • Model

4912 Realtime systems conforming to one of the POSIX.13 realtime profiles are expected (and  
4913 desired) to be supported on systems with more than one type or pool of memory (for  
4914 example, SRAM, DRAM, ROM, EPROM, EEPROM), where each type or pool of memory may  
4915 be accessible by one or more processors via one or more busses (ports). Memory mapped  
4916 files, shared memory objects, and the language-specific storage allocation operators (*malloc()*  
4917 for the ISO C standard, *new* for ISO Ada) fail to provide application program interfaces  
4918 versatile enough to allow applications to control their utilization of such diverse memory  
4919 resources. The typed memory interfaces *posix\_typed\_mem\_open()*, *posix\_mem\_offset()*,  
4920 *posix\_typed\_mem\_get\_info()*, *mmap()*, and *munmap()* defined herein support the model of  
4921 typed memory described below.

4922 For purposes of this model, a system comprises several processors (for example, P1 and P2),  
4923 several physical memory pools (for example, M1, M2, M2a, M2b, M3, M4, and M5), and  
4924 several busses or “ports” (for example, B1, B2, B3, and B4) interconnecting the various  
4925 processors and memory pools in some system-specific way. Notice that some memory pools  
4926 may be contained in others (for example, M2a and M2b are contained in M2).

4927 Figure B-1 (on page 3412) shows an example of such a model. In a system like this, an  
4928 application should be able to perform the following operations:

4929



\* All addresses in pool M<sub>2</sub> (comprising pools M<sub>2a</sub> and M<sub>2b</sub>) accessible via port B<sub>1</sub>.  
 Addresses in pool M<sub>2b</sub> are also accessible via port B<sub>2</sub>.  
 Addresses in pool M<sub>2a</sub> are *not* accessible via port B<sub>2</sub>.

4930

**Figure B-1** Example of a System with Typed Memory

4931

— Typed Memory Allocation

4932

An application should be able to allocate memory dynamically from the desired pool using the desired bus, and map it into a process' address space. For example, processor P1 can allocate some portion of memory pool M1 through port B1, treating all unmapped subareas of M1 as a heap-storage resource from which memory may be allocated. This portion of memory is mapped into the process' address space, and subsequently deallocated when unmapped from all processes.

4933

4934

4935

4936

4937

4938

— Using the Same Storage Region from Different Busses

4939

An application process with a mapped region of storage that is accessed from one bus should be able to map that same storage area at another address (subject to page size restrictions detailed in *mmap()*), to allow it to be accessed from another bus. For example, processor P1 may wish to access the same region of memory pool M2b both through ports B1 and B2.

4940

4941

4942

4943

4944

— Sharing Typed Memory Regions

4945

Several application processes running on the same or different processors may wish to share a particular region of a typed memory pool. Each process or processor may wish to access this region through different busses. For example, processor P1 may want to share a region of memory pool M4 with processor P2, and they may be required to use busses B2 and B3, respectively, to minimize bus contention. A problem arises here when a process allocates and maps a portion of fragmented memory and then wants to share this region of memory with another process, either in the same processor or different processors. The solution adopted is to allow the first process to find out the memory map (offsets and lengths) of all the different fragments of memory that were mapped into its address space, by repeatedly calling *posix\_mem\_offset()*. Then, this process can pass the offsets and lengths obtained to the second process, which can then map the same memory fragments into its address space.

4946

4947

4948

4949

4950

4951

4952

4953

4954

4955

4956

4957

— Contiguous Allocation

4958

The problem of finding the memory map of the different fragments of the memory pool that were mapped into logically contiguous addresses of a given process, can be solved by requesting contiguous allocation. For example, a process in P1 can allocate 10 Kbytes of physically contiguous memory from M3-B1, and obtain the offset (within pool M3) of

4959

4960

4961

4962 this block of memory. Then, it can pass this offset (and the length) to a process in P2 using  
 4963 some interprocess communication mechanism. The second process can map the same  
 4964 block of memory by using the offset transferred and specifying M3-B2.

4965 — Unallocated Mapping

4966 Any subarea of a memory pool that is mapped to a process, either as the result of an  
 4967 allocation request or an explicit mapping, is normally unavailable for allocation. Special  
 4968 processes such as debuggers, however, may need to map large areas of a typed memory  
 4969 pool, yet leave those areas available for allocation.

4970 Typed memory allocation and mapping has to coexist with storage allocation operators like  
 4971 *malloc()*, but systems are free to choose how to implement this coexistence. For example, it  
 4972 may be system configuration-dependent if all available system memory is made part of one  
 4973 of the typed memory pools or if some part will be restricted to conventional allocation  
 4974 operators. Equally system configuration-dependent may be the availability of operators like  
 4975 *malloc()* to allocate storage from certain typed memory pools. It is not excluded to configure  
 4976 a system such that a given named pool, P1, is in turn split into non-overlapping named  
 4977 subpools. For example, M1-B1, M2-B1, and M3-B1 could also be accessed as one common  
 4978 pool M123-B1. A call to *malloc()* on P1 could work on such a larger pool while full  
 4979 optimization of memory usage by P1 would require typed memory allocation at the subpool  
 4980 level.

4981 • Existing Practice

4982 OS-9 provides for the naming (numbering) and prioritization of memory types by a system  
 4983 administrator. It then provides APIs to request memory allocation of typed (colored)  
 4984 memory by number, and to generate a bus address from a mapped memory address  
 4985 (translate). When requesting colored memory, the user can specify type 0 to signify allocation  
 4986 from the first available type in priority order.

4987 HP-RT presents interfaces to map different kinds of storage regions that are visible through a  
 4988 VME bus, although it does not provide allocation operations. It also provides functions to  
 4989 perform address translation between VME addresses and virtual addresses. It represents a  
 4990 VME-bus unique solution to the general problem.

4991 The PSOS approach is similar (that is, based on a pre-established mapping of bus address  
 4992 ranges to specific memories) with a concept of segments and regions (regions dynamically  
 4993 allocated from a heap which is a special segment). Therefore, PSOS does not fully address the  
 4994 general allocation problem either. PSOS does not have a “process”-based model, but more of  
 4995 a “thread”-only-based model of multi-tasking. So mapping to a process address space is not  
 4996 an issue.

4997 QNX (a Canadian OS vendor specializing in realtime embedded systems on 80x86-based  
 4998 processors) uses the System V approach of opening specially named devices (shared memory  
 4999 segments) and using *mmap()* to then gain access from the process. They do not address  
 5000 allocation directly, but once typed shared memory can be mapped, an “allocation manager”  
 5001 process could be written to handle requests for allocation.

5002 The System V approach also included allocation, implemented by opening yet other special  
 5003 “devices” which allocate, rather than appearing as a whole memory object.

5004 The Orkid realtime kernel interface definition has operations to manage memory “regions”  
 5005 and “pools”, which are areas of memory that may reflect the differing physical nature of the  
 5006 memory. Operations to allocate memory from these regions and pools are also provided. |

- 5007 • Requirements
- 5008 Existing practice in SVID-derived UNIX systems relies on functionality similar to *mmap()*
- 5009 and its related interfaces to achieve mapping and allocation of typed memory. However, the
- 5010 issue of sharing typed memory (allocated or mapped) and the complication of multiple ports
- 5011 are not addressed in any consistent way by existing UNIX system practice. Part of this
- 5012 functionality is existing practice in specialized realtime operating systems. In order to
- 5013 solidify the capabilities implied by the model above, the following requirements are imposed
- 5014 on the interface:
- 5015 — Identification of Typed Memory Pools and Ports
- 5016 All processes (running in all processors) in the system shall be able to identify a particular
- 5017 (system configured) typed memory pool accessed through a particular (system
- 5018 configured) port by a name. That name shall be a member of a name space common to all
- 5019 these processes, but need not be the same name space as that containing ordinary
- 5020 filenames. The association between memory pools/ports and corresponding names is
- 5021 typically established when the system is configured. The “open” operation for typed
- 5022 memory objects should be distinct from the *open()* function, for consistency with other
- 5023 similar services, but implementable on top of *open()*. This implies that the handle for a
- 5024 typed memory object will be a file descriptor.
- 5025 — Allocation and Mapping of Typed Memory
- 5026 Once a typed memory object has been identified by a process, it shall be possible to both
- 5027 map user-selected subareas of that object into process address space and to map system-
- 5028 selected (that is, dynamically allocated) subareas of that object, with user-specified
- 5029 length, into process address space. It shall also be possible to determine the maximum
- 5030 length of memory allocation that may be requested from a given typed memory object.
- 5031 — Sharing Typed Memory
- 5032 Two or more processes shall be able to share portions of typed memory, either user-
- 5033 selected or dynamically allocated. This requirement applies also to dynamically allocated
- 5034 regions of memory that are composed of several non-contiguous pieces.
- 5035 — Contiguous Allocation
- 5036 For dynamic allocation, it shall be the user’s option whether the system is required to
- 5037 allocate a contiguous subarea within the typed memory object, or whether it is permitted
- 5038 to allocate discontinuous fragments which appear contiguous in the process mapping.
- 5039 Contiguous allocation simplifies the process of sharing allocated typed memory, while
- 5040 discontinuous allocation allows for potentially better recovery of deallocated typed
- 5041 memory.
- 5042 — Accessing Typed Memory Through Different Ports
- 5043 Once a subarea of a typed memory object has been mapped, it shall be possible to
- 5044 determine the location and length corresponding to a user-selected portion of that object
- 5045 within the memory pool. This location and length can then be used to remap that portion
- 5046 of memory for access from another port. If the referenced portion of typed memory was
- 5047 allocated discontinuously, the length thus determined may be shorter than anticipated,
- 5048 and the user code shall adapt to the value returned.
- 5049 — Deallocation
- 5050 When a previously mapped subarea of typed memory is no longer mapped by any
- 5051 process in the system—as a result of a call or calls to *munmap()*—that subarea shall
- 5052 become potentially reusable for dynamic allocation; actual reuse of the subarea is a

- 5053 function of the dynamic typed memory allocation policy.
- 5054 — Unallocated Mapping
- 5055 It shall be possible to map user-selected subareas of a typed memory object without  
5056 marking that subarea as unavailable for allocation. This option is not the default behavior,  
5057 and shall require appropriate privilege.
- 5058 • Scenario
- 5059 The following scenario will serve to clarify the use of the typed memory interfaces.
- 5060 Process A running on P1 (see Figure B-1 (on page 3412)) wants to allocate some memory  
5061 from memory pool M2, and it wants to share this portion of memory with process B running  
5062 on P2. Since P2 only has access to the lower part of M2, both processes will use the memory  
5063 pool named M2b which is the part of M2 that is accessible both from P1 and P2. The  
5064 operations that both processes need to perform are shown below:
- 5065 — Allocating Typed Memory
- 5066 Process A calls *posix\_typed\_mem\_open()* with the name */typed.m2b-b1* and a *tflag* of  
5067 *POSIX\_TYPED\_MEM\_ALLOCATE* to get a file descriptor usable for allocating from pool  
5068 M2b accessed through port B1. It then calls *mmap()* with this file descriptor requesting a  
5069 length of 4 096 bytes. The system allocates two discontinuous blocks of sizes 1 024 and  
5070 3 072 bytes within M2b. The *mmap()* function returns a pointer to a 4 096 byte array in  
5071 process A's logical address space, mapping the allocated blocks contiguously. Process A  
5072 can then utilize the array, and store data in it.
- 5073 — Determining the Location of the Allocated Blocks
- 5074 Process A can determine the lengths and offsets (relative to M2b) of the two blocks  
5075 allocated, by using the following procedure: First, process A calls *posix\_mem\_offset()*  
5076 with the address of the first element of the array and length 4 096. Upon return, the offset and  
5077 length (1 024 bytes) of the first block are returned. A second call to *posix\_mem\_offset()* is  
5078 then made using the address of the first element of the array plus 1 024 (the length of the  
5079 first block), and a new length of 4 096–1 024. If there were more fragments allocated, this  
5080 procedure could have been continued within a loop until the offsets and lengths of all the  
5081 blocks were obtained. Notice that this relatively complex procedure can be avoided if  
5082 contiguous allocation is requested (by opening the typed memory object with the *tflag*  
5083 *POSIX\_TYPED\_MEM\_ALLOCATE\_CONTIG*).
- 5084 — Sharing Data Across Processes
- 5085 Process A passes the two offset values and lengths obtained from the *posix\_mem\_offset()*  
5086 calls to process B running on P2, via some form of interprocess communication. Process B  
5087 can gain access to process A's data by calling *posix\_typed\_mem\_open()* with the name  
5088 */typed.m2b-b2* and a *tflag* of zero, then using two *mmap()* calls on the resulting file  
5089 descriptor to map the two subareas of that typed memory object to its own address space.
- 5090 • Rationale for no *mem\_alloc()* and *mem\_free()*
- 5091 The standard developers had originally proposed a pair of new flags to *mmap()* which, when  
5092 applied to a typed memory object descriptor, would cause *mmap()* to allocate dynamically  
5093 from an unallocated and unmapped area of the typed memory object. Deallocation was  
5094 similarly accomplished through the use of *munmap()*. This was rejected by the ballot group  
5095 because it excessively complicated the (already rather complex) *mmap()* interface and  
5096 introduced semantics useful only for typed memory, to a function which must also map  
5097 shared memory and files. They felt that a memory allocator should be built on top of *mmap()*  
5098 instead of being incorporated within the same interface, much as the ISO C standard libraries

5099 build *malloc()* on top of the virtual memory mapping functions *brk()* and *sbrk()*. This would  
 5100 eliminate the complicated semantics involved with unmapping only part of an allocated  
 5101 block of typed memory.

5102 To attempt to achieve ballot group consensus, typed memory allocation and deallocation was  
 5103 first migrated from *mmap()* and *munmap()* to a pair of complementary functions modeled on  
 5104 the ISO C standard *malloc()* and *free()*. The *mem\_alloc()* function specified explicitly the  
 5105 typed memory object (typed memory pool/access port) from which allocation takes place,  
 5106 unlike *malloc()* where the memory pool and port are unspecified. The *mem\_free()* function  
 5107 handled deallocation. These new semantics still met all of the requirements detailed above  
 5108 without modifying the behavior of *mmap()* except to allow it to map specified areas of typed  
 5109 memory objects. An implementation would have been free to implement *mem\_alloc()* and  
 5110 *mem\_free()* over *mmap()*, through *mmap()*, or independently but cooperating with *mmap()*.

5111 The ballot group was queried to see if this was an acceptable alternative, and while there was  
 5112 some agreement that it achieved the goal of removing the complicated semantics of  
 5113 allocation from the *mmap()* interface, several balloters realized that it just created two  
 5114 additional functions that behaved, in great part, like *mmap()*. These balloters proposed an  
 5115 alternative which has been implemented here in place of a separate *mem\_alloc()* and  
 5116 *mem\_free()*. This alternative is based on four specific suggestions:

- 5117 1. The *posix\_typed\_mem\_open()* function should provide a flag which specifies “allocate  
 5118 on *mmap()*” (otherwise, *mmap()* just maps the underlying object). This allows things  
 5119 roughly similar to **/dev/zero** versus **/dev/swap**. Two such flags have been implemented,  
 5120 one of which forces contiguous allocation.
- 5121 2. The *posix\_mem\_offset()* function is acceptable because it can be applied usefully to  
 5122 mapped objects in general. It should return the file descriptor of the underlying object.
- 5123 3. The *mem\_get\_info()* function in an earlier draft should be renamed  
 5124 *posix\_typed\_mem\_get\_info()* because it is not generally applicable to memory objects. It  
 5125 should probably return the file descriptor’s allocation attribute. We have implemented  
 5126 the renaming of the function, but reject having it return a piece of information which is  
 5127 readily known by an application without this function. Its whole purpose is to query  
 5128 the typed memory object for attributes that are not user-specified, but determined by  
 5129 the implementation.
- 5130 4. There should be no separate *mem\_alloc()* or *mem\_free()* functions. Instead, using  
 5131 *mmap()* on a typed memory object opened with an “allocate on *mmap()*” flag should be  
 5132 used to force allocation. These are precisely the semantics defined in the current draft.

5133 • Rationale for no Typed Memory Access Management

5134 The working group had originally defined an additional interface (and an additional kind of  
 5135 object: typed memory master) to establish and dissolve mappings to typed memory on  
 5136 behalf of devices or processors which were independent of the operating system and had no  
 5137 inherent capability to directly establish mappings on their own. This was to have provided  
 5138 functionality similar to device driver interfaces such as *physio()* and their underlying bus-  
 5139 specific interfaces (for example, *mballoc()*) which serve to set up and break down DMA  
 5140 pathways, and derive mapped addresses for use by hardware devices and processor cards.

5141 The ballot group felt that this was beyond the scope of POSIX.1 and its amendments.  
 5142 Furthermore, the removal of interrupt handling interfaces from a preceding amendment (the  
 5143 IEEE Std 1003.1d-1999) during its balloting process renders these typed memory access  
 5144 management interfaces an incomplete solution to portable device management from a user  
 5145 process; it would be possible to initiate a device transfer to/from typed memory, but  
 5146 impossible to handle the transfer-complete interrupt in a portable way.

5147 To achieve ballot group consensus, all references to typed memory access management  
5148 capabilities were removed. The concept of portable interfaces from a device driver to both  
5149 operating system and hardware is being addressed by the Uniform Driver Interface (UDI)  
5150 industry forum, with formal standardization deferred until proof of concept and industry-  
5151 wide acceptance and implementation.

#### 5152 B.2.8.4 Process Scheduling

5153 IEEE PASC Interpretation 1003.1 #96 has been applied, adding the `pthread_setschedprio()` |  
5154 function. This was added since previously there was no way for a thread to lower its own |  
5155 priority without going to the tail of the threads list for its new priority. This capability is |  
5156 necessary to bound the duration of priority inversion encountered by a thread. |

5157 The following portion of the rationale presents models, requirements, and standardization issues |  
5158 relevant to process scheduling; see also Section B.2.9.4 (on page 3456). |

5159 In an operating system supporting multiple concurrent processes, the system determines the |  
5160 order in which processes execute to meet implementation-defined goals. For time-sharing |  
5161 systems, the goal is to enhance system throughput and promote fairness; the application is |  
5162 provided little or no control over this sequencing function. While this is acceptable and desirable |  
5163 behavior in a time-sharing system, it is inappropriate in a realtime system; realtime applications |  
5164 must specifically control the execution sequence of their concurrent processes in order to meet |  
5165 externally defined response requirements.

5166 In IEEE Std 1003.1-200x, the control over process sequencing is provided using a concept of |  
5167 scheduling policies. These policies, described in detail in this section, define the behavior of the |  
5168 system whenever processor resources are to be allocated to competing processes. Only the |  
5169 behavior of the policy is defined; conforming implementations are free to use any mechanism |  
5170 desired to achieve the described behavior.

#### 5171 • Models

5172 In an operating system supporting multiple concurrent processes, the system determines the |  
5173 order in which processes execute and might force long-running processes to yield to other |  
5174 processes at certain intervals. Typically, the scheduling code is executed whenever an event |  
5175 occurs that might alter the process to be executed next.

5176 The simplest scheduling strategy is a “first-in, first-out” (FIFO) dispatcher. Whenever a |  
5177 process becomes runnable, it is placed on the end of a ready list. The process at the front of |  
5178 the ready list is executed until it exits or becomes blocked, at which point it is removed from |  
5179 the list. This scheduling technique is also known as “run-to-completion” or “run-to-block”.

5180 A natural extension to this scheduling technique is the assignment of a “non-migrating |  
5181 priority” to each process. This policy differs from strict FIFO scheduling in only one respect: |  
5182 whenever a process becomes runnable, it is placed at the end of the list of processes runnable |  
5183 at that priority level. When selecting a process to run, the system always selects the first |  
5184 process from the highest priority queue with a runnable process. Thus, when a process |  
5185 becomes unblocked, it will preempt a running process of lower priority without otherwise |  
5186 altering the ready list. Further, if a process elects to alter its priority, it is removed from the |  
5187 ready list and reinserted, using its new priority, according to the policy above.

5188 While the above policy might be considered unfriendly in a time-sharing environment in |  
5189 which multiple users require more balanced resource allocation, it could be ideal in a |  
5190 realtime environment for several reasons. The most important of these is that it is |  
5191 deterministic: the highest-priority process is always run and, among processes of equal |  
5192 priority, the process that has been runnable for the longest time is executed first. Because of |  
5193 this determinism, cooperating processes can implement more complex scheduling simply by

- 5194 altering their priority. For instance, if processes at a single priority were to reschedule  
5195 themselves at fixed time intervals, a time-slice policy would result.
- 5196 In a dedicated operating system in which all processes are well-behaved realtime  
5197 applications, non-migrating priority scheduling is sufficient. However, many existing  
5198 implementations provide for more complex scheduling policies.
- 5199 IEEE Std 1003.1-200x specifies a linear scheduling model. In this model, every process in the  
5200 system has a priority. The system scheduler always dispatches a process that has the highest  
5201 (generally the most time-critical) priority among all runnable processes in the system. As  
5202 long as there is only one such process, the dispatching policy is trivial. When multiple  
5203 processes of equal priority are eligible to run, they are ordered according to a strict run-to-  
5204 completion (FIFO) policy.
- 5205 The priority is represented as a positive integer and is inherited from the parent process. For  
5206 processes running under a fixed priority scheduling policy, the priority is never altered  
5207 except by an explicit function call.
- 5208 It was determined arbitrarily that larger integers correspond to “higher priorities”.
- 5209 Certain implementations might impose restrictions on the priority ranges to which processes  
5210 can be assigned. There also can be restrictions on the set of policies to which processes can be  
5211 set.
- 5212 • Requirements
- 5213 Realtime processes require that scheduling be fast and deterministic, and that it guarantees  
5214 to preempt lower priority processes.
- 5215 Thus, given the linear scheduling model, realtime processes require that they be run at a  
5216 priority that is higher than other processes. Within this framework, realtime processes are  
5217 free to yield execution resources to each other in a completely portable and implementation-  
5218 defined manner.
- 5219 As there is a generally perceived requirement for processes at the same priority level to share  
5220 processor resources more equitably, provisions are made by providing a scheduling policy  
5221 (that is, SCHED\_RR) intended to provide a timeslice-like facility.
- 5222 **Note:** The following topics assume that low numeric priority implies low scheduling criticality  
5223 and *vice versa*.
- 5224 • Rationale for New Interface
- 5225 Realtime applications need to be able to determine when processes will run in relation to  
5226 each other. It must be possible to guarantee that a critical process will run whenever it is  
5227 runnable; that is, whenever it wants to for as long as it needs. SCHED\_FIFO satisfies this  
5228 requirement. Additionally, SCHED\_RR was defined to meet a realtime requirement for a  
5229 well-defined time-sharing policy for processes at the same priority.
- 5230 It would be possible to use the BSD *setpriority()* and *getpriority()* functions by redefining the  
5231 meaning of the “nice” parameter according to the scheduling policy currently in use by the  
5232 process. The System V *nice()* interface was felt to be undesirable for realtime because it  
5233 specifies an adjustment to the “nice” value, rather than setting it to an explicit value.  
5234 Realtime applications will usually want to set priority to an explicit value. Also, System V  
5235 *nice()* does not allow for changing the priority of another process.
- 5236 With the POSIX.1b interfaces, the traditional “nice” value does not affect the SCHED\_FIFO  
5237 or SCHED\_RR scheduling policies. If a “nice” value is supported, it is implementation-  
5238 defined whether it affects the SCHED\_OTHER policy.

5239 An important aspect of IEEE Std 1003.1-200x is the explicit description of the queuing and  
5240 preemption rules. It is critical, to achieve deterministic scheduling, that such rules be stated  
5241 clearly in IEEE Std 1003.1-200x.

5242 IEEE Std 1003.1-200x does not address the interaction between priority and swapping. The  
5243 issues involved with swapping and virtual memory paging are extremely implementation-  
5244 defined and would be nearly impossible to standardize at this point. The proposed  
5245 scheduling paradigm, however, fully describes the scheduling behavior of runnable  
5246 processes, of which one criterion is that the working set be resident in memory. Assuming  
5247 the existence of a portable interface for locking portions of a process in memory, paging  
5248 behavior need not affect the scheduling of realtime processes.

5249 IEEE Std 1003.1-200x also does not address the priorities of “system” processes. In general,  
5250 these processes should always execute in low-priority ranges to avoid conflict with other  
5251 realtime processes. Implementations should document the priority ranges in which system  
5252 processes run.

5253 The default scheduling policy is not defined. The effect of I/O interrupts and other system  
5254 processing activities is not defined. The temporary lending of priority from one process to  
5255 another (such as for the purposes of affecting freeing resources) by the system is not  
5256 addressed. Preemption of resources is not addressed. Restrictions on the ability of a process  
5257 to affect other processes beyond a certain level (influence levels) is not addressed.

5258 The rationale used to justify the simple time-quantum scheduler is that it is common practice  
5259 to depend upon this type of scheduling to assure “fair” distribution of processor resources  
5260 among portions of the application that must interoperate in a serial fashion. Note that  
5261 IEEE Std 1003.1-200x is silent with respect to the setting of this time quantum, or whether it is  
5262 a system-wide value or a per-process value, although it appears that the prevailing realtime  
5263 practice is for it to be a system-wide value.

5264 In a system with  $N$  processes at a given priority, all processor-bound, in which the time  
5265 quantum is equal for all processes at a specific priority level, the following assumptions are  
5266 made of such a scheduling policy:

- 5267 1. A time quantum  $Q$  exists and the current process will own control of the processor for  
5268 at least a duration of  $Q$  and will have the processor for a duration of  $Q$ .
- 5269 2. The  $N$ th process at that priority will control a processor within a duration of  $(N-1) \times Q$ .

5270 These assumptions are necessary to provide equal access to the processor and bounded  
5271 response from the application.

5272 The assumptions hold for the described scheduling policy only if no system overhead, such  
5273 as interrupt servicing, is present. If the interrupt servicing load is non-zero, then one of the  
5274 two assumptions becomes fallacious, based upon how  $Q$  is measured by the system.

5275 If  $Q$  is measured by clock time, then the assumption that the process obtains a duration  $Q$   
5276 processor time is false if interrupt overhead exists. Indeed, a scenario can be constructed with  
5277  $N$  processes in which a single process undergoes complete processor starvation if a  
5278 peripheral device, such as an analog-to-digital converter, generates significant interrupt  
5279 activity periodically with a period of  $N \times Q$ .

5280 If  $Q$  is measured as actual processor time, then the assumption that the  $N$ th process runs in  
5281 within the duration  $(N-1) \times Q$  is false.

5282 It should be noted that SCHED\_FIFO suffers from interrupt-based delay as well. However,  
5283 for SCHED\_FIFO, the implied response of the system is “as soon as possible”, so that the  
5284 interrupt load for this case is a vendor selection and not a compliance issue.

5285 With this in mind, it is necessary either to complete the definition by including bounds on the  
5286 interrupt load, or to modify the assumptions that can be made about the scheduling policy.

5287 Since the motivation of inclusion of the policy is common usage, and since current  
5288 applications do not enjoy the luxury of bounded interrupt load, item (2) above is sufficient to  
5289 express existing application needs and is less restrictive in the standard definition. No  
5290 difference in interface is necessary.

5291 In an implementation in which the time quantum is equal for all processes at a specific  
5292 priority, our assumptions can then be restated as:

5293 — A time quantum  $Q$  exists, and a processor-bound process will be rescheduled after a  
5294 duration of, at most,  $Q$ . Time quantum  $Q$  may be defined in either wall clock time or  
5295 execution time.

5296 — In general, the  $N$ th process of a priority level should wait no longer than  $(N-1) \times Q$  time  
5297 to execute, assuming no processes exist at higher priority levels.

5298 — No process should wait indefinitely.

5299 For implementations supporting per-process time quanta, these assumptions can be readily  
5300 extended.

### 5301 **Sporadic Server Scheduling Policy**

5302 The sporadic server is a mechanism defined for scheduling aperiodic activities in time-critical  
5303 realtime systems. This mechanism reserves a certain bounded amount of execution capacity for  
5304 processing aperiodic events at a high priority level. Any aperiodic events that cannot be  
5305 processed within the bounded amount of execution capacity are executed in the background at a  
5306 low priority level. Thus, a certain amount of execution capacity can be guaranteed to be  
5307 available for processing periodic tasks, even under burst conditions in the arrival of aperiodic  
5308 processing requests (that is, a large number of requests in a short time interval). The sporadic  
5309 server also simplifies the schedulability analysis of the realtime system, because it allows  
5310 aperiodic processes or threads to be treated as if they were periodic. The sporadic server was  
5311 first described by Sprunt, et al.

5312 The key concept of the sporadic server is to provide and limit a certain amount of computation  
5313 capacity for processing aperiodic events at their assigned normal priority, during a time interval  
5314 called the *replenishment period*. Once the entity controlled by the sporadic server mechanism is  
5315 initialized with its period and execution-time budget attributes, it preserves its execution  
5316 capacity until an aperiodic request arrives. The request will be serviced (if there are no higher  
5317 priority activities pending) as long as there is execution capacity left. If the request is completed,  
5318 the actual execution time used to service it is subtracted from the capacity, and a replenishment  
5319 of this amount of execution time is scheduled to happen one replenishment period after the  
5320 arrival of the aperiodic request. If the request is not completed, because there is no execution  
5321 capacity left, then the aperiodic process or thread is assigned a lower background priority. For  
5322 each portion of consumed execution capacity the execution time used is replenished after one  
5323 replenishment period. At the time of replenishment, if the sporadic server was executing at a  
5324 background priority level, its priority is elevated to the normal level. Other similar  
5325 replenishment policies have been defined, but the one presented here represents a compromise  
5326 between efficiency and implementation complexity.

5327 The interface that appears in this section defines a new scheduling policy for threads and  
5328 processes that behaves according to the rules of the sporadic server mechanism. Scheduling  
5329 attributes are defined and functions are provided to allow the user to set and get the parameters  
5330 that control the scheduling behavior of this mechanism, namely the normal and low priority, the  
5331 replenishment period, the maximum number of pending replenishment operations, and the

- 5332 initial execution-time budget.
- 5333 • Scheduling Aperiodic Activities
- 5334 Virtually all realtime applications are required to process aperiodic activities. In many cases,  
5335 there are tight timing constraints that the response to the aperiodic events must meet. Usual  
5336 timing requirements imposed on the response to these events are:
- 5337 — The effects of an aperiodic activity on the response time of lower priority activities must  
5338 be controllable and predictable.
  - 5339 — The system must provide the fastest possible response time to aperiodic events.
  - 5340 — It must be possible to take advantage of all the available processing bandwidth not  
5341 needed by time-critical activities to enhance average-case response times to aperiodic  
5342 events.
- 5343 Traditional methods for scheduling aperiodic activities are background processing, polling  
5344 tasks, and direct event execution:
- 5345 — Background processing consists of assigning a very low priority to the processing of  
5346 aperiodic events. It utilizes all the available bandwidth in the system that has not been  
5347 consumed by higher priority threads. However, it is very difficult, or impossible, to meet  
5348 requirements on average-case response time, because the aperiodic entity has to wait for  
5349 the execution of all other entities which have higher priority.
  - 5350 — Polling consists of creating a periodic process or thread for servicing aperiodic requests.  
5351 At regular intervals, the polling entity is started and it services accumulated pending  
5352 aperiodic requests. If no aperiodic requests are pending, the polling entity suspends itself  
5353 until its next period. Polling allows the aperiodic requests to be processed at a higher  
5354 priority level. However, worst and average-case response times of polling entities are a  
5355 direct function of the polling period, and there is execution overhead for each polling  
5356 period, even if no event has arrived. If the deadline of the aperiodic activity is short  
5357 compared to the inter-arrival time, the polling frequency must be increased to guarantee  
5358 meeting the deadline. For this case, the increase in frequency can dramatically reduce the  
5359 efficiency of the system and, therefore, its capacity to meet all deadlines. Yet, polling  
5360 represents a good way to handle a large class of practical problems because it preserves  
5361 system predictability, and because the amortized overhead drops as load increases.
  - 5362 — Direct event execution consists of executing the aperiodic events at a high fixed-priority  
5363 level. Typically, the aperiodic event is processed by an interrupt service routine as soon as  
5364 it arrives. This technique provides predictable response times for aperiodic events, but  
5365 makes the response times of all lower priority activities completely unpredictable under  
5366 burst arrival conditions. Therefore, if the density of aperiodic event arrivals is  
5367 unbounded, it may be a dangerous technique for time-critical systems. Yet, for those cases  
5368 in which the physics of the system imposes a bound on the event arrival rate, it is  
5369 probably the most efficient technique.
  - 5370 — The sporadic server scheduling algorithm combines the predictability of the polling  
5371 approach with the short response times of the direct event execution. Thus, it allows  
5372 systems to meet an important class of application requirements that cannot be met by  
5373 using the traditional approaches. Multiple sporadic servers with different attributes can  
5374 be applied to the scheduling of multiple classes of aperiodic events, each with different  
5375 kinds of timing requirements, such as individual deadlines, average response times, and  
5376 so on. It also has many other interesting applications for realtime, such as scheduling  
5377 producer/consumer tasks in time-critical systems, limiting the effects of faults on the  
5378 estimation of task execution-time requirements, and so on.

- 5379           • Existing Practice
- 5380           The sporadic server has been used in different kinds of applications, including military  
5381           avionics, robot control systems, industrial automation systems, and so on. There are  
5382           examples of many systems that cannot be successfully scheduled using the classic  
5383           approaches, such as direct event execution, or polling, and are schedulable using a sporadic  
5384           server scheduler. The sporadic server algorithm itself can successfully schedule all systems  
5385           scheduled with direct event execution or polling.
- 5386           The sporadic server scheduling policy has been implemented as a commercial product in the  
5387           run-time system of the Verdex Ada compiler. There are also many applications that have  
5388           used a much less efficient application-level sporadic server. These real-time applications  
5389           would benefit from a sporadic server scheduler implemented at the scheduler level.
- 5390           • Library-Level *versus* Kernel-Level Implementation
- 5391           The sporadic server interface described in this section requires the sporadic server policy to  
5392           be implemented at the same level as the scheduler. This means that the process sporadic  
5393           server shall be implemented at the kernel level and the thread sporadic server policy shall be  
5394           implemented at the same level as the thread scheduler; that is, kernel or library level.
- 5395           In an earlier interface for the sporadic server, this mechanism was implementable at a  
5396           different level than the scheduler. This feature allowed the implementer to choose between  
5397           an efficient scheduler-level implementation, or a simpler user or library-level  
5398           implementation. However, the working group considered that this interface made the use of  
5399           sporadic servers more complex, and that library-level implementations would lack some of  
5400           the important functionality of the sporadic server, namely the limitation of the actual  
5401           execution time of aperiodic activities. The working group also felt that the interface  
5402           described in this chapter does not preclude library-level implementations of threads intended  
5403           to provide efficient low-overhead scheduling for those threads that are not scheduled under  
5404           the sporadic server policy.
- 5405           • Range of Scheduling Priorities
- 5406           Each of the scheduling policies supported in IEEE Std 1003.1-200x has an associated range of  
5407           priorities. The priority ranges for each policy might or might not overlap with the priority  
5408           ranges of other policies. For time-critical realtime applications it is usual for periodic and  
5409           aperiodic activities to be scheduled together in the same processor. Periodic activities will  
5410           usually be scheduled using the SCHED\_FIFO scheduling policy, while aperiodic activities  
5411           may be scheduled using SCHED\_SPORADIC. Since the application developer will require  
5412           complete control over the relative priorities of these activities in order to meet his timing  
5413           requirements, it would be desirable for the priority ranges of SCHED\_FIFO and  
5414           SCHED\_SPORADIC to overlap completely. Therefore, although IEEE Std 1003.1-200x does  
5415           not require any particular relationship between the different priority ranges, it is  
5416           recommended that these two ranges should coincide.
- 5417           • Dynamically Setting the Sporadic Server Policy
- 5418           Several members of the working group requested that implementations should not be  
5419           required to support dynamically setting the sporadic server scheduling policy for a thread.  
5420           The reason is that this policy may have a high overhead for library-level implementations of  
5421           threads, and if threads are allowed to dynamically set this policy, this overhead can be  
5422           experienced even if the thread does not use that policy. By disallowing the dynamic setting of  
5423           the sporadic server scheduling policy, these implementations can accomplish efficient  
5424           scheduling for threads using other policies. If a strictly conforming application needs to use  
5425           the sporadic server policy, and is therefore willing to pay the overhead, it must set this policy  
5426           at the time of thread creation.

5427 • Limitation of the Number of Pending Replenishments

5428 The number of simultaneously pending replenishment operations must be limited for each  
 5429 sporadic server for two reasons: an unlimited number of replenishment operations would  
 5430 need an unlimited number of system resources to store all the pending replenishment  
 5431 operations; on the other hand, in some implementations each replenishment operation will  
 5432 represent a source of priority inversion (just for the duration of the replenishment operation)  
 5433 and thus, the maximum amount of replenishments must be bounded to guarantee bounded  
 5434 response times. The way in which the number of replenishments is bounded is by lowering  
 5435 the priority of the sporadic server to *sched\_ss\_low\_priority* when the number of pending  
 5436 replenishments has reached its limit. In this way, no new replenishments are scheduled until  
 5437 the number of pending replenishments decreases.

5438 In the sporadic server scheduling policy defined in IEEE Std 1003.1-200x, the application can  
 5439 specify the maximum number of pending replenishment operations for a single sporadic  
 5440 server, by setting the value of the *sched\_ss\_max\_repl* scheduling parameter. This value must  
 5441 be between one and {SS\_REPL\_MAX}, which is a maximum limit imposed by the  
 5442 implementation. The limit {SS\_REPL\_MAX} must be greater than or equal to  
 5443 {\_POSIX\_SS\_REPL\_MAX}, which is defined to be four in IEEE Std 1003.1-200x. The minimum  
 5444 limit of four was chosen so that an application can at least guarantee that four different  
 5445 aperiodic events can be processed during each interval of length equal to the replenishment  
 5446 period.

5447 *B.2.8.5 Clocks and Timers*

5448 • Clocks

5449 IEEE Std 1003.1-200x and the ISO C standard both define functions for obtaining system  
 5450 time. Implicit behind these functions is a mechanism for measuring passage of time. This  
 5451 specification makes this mechanism explicit and calls it a clock. The *CLOCK\_REALTIME*  
 5452 clock required by IEEE Std 1003.1-200x is a higher resolution version of the clock that  
 5453 maintains POSIX.1 system time. This is a “system-wide” clock, in that it is visible to all  
 5454 processes and, were it possible for multiple processes to all read the clock at the same time,  
 5455 they would see the same value.

5456 An extensible interface was defined, with the ability for implementations to define additional  
 5457 clocks. This was done because of the observation that many realtime platforms support  
 5458 multiple clocks, and it was desired to fit this model within the standard interface. But  
 5459 implementation-defined clocks need not represent actual hardware devices, nor are they  
 5460 necessarily system-wide.

5461 • Timers

5462 Two timer types are required for a system to support realtime applications:

5463 1. One-shot

5464 A one-shot timer is a timer that is armed with an initial expiration time, either relative  
 5465 to the current time or at an absolute time (based on some timing base, such as time in  
 5466 seconds and nanoseconds since the Epoch). The timer expires once and then is  
 5467 disarmed. With the specified facilities, this is accomplished by setting the *it\_value*  
 5468 member of the *value* argument to the desired expiration time and the *it\_interval* member  
 5469 to zero.

5470 2. Periodic

5471 A periodic timer is a timer that is armed with an initial expiration time, again either  
 5472 relative or absolute, and a repetition interval. When the initial expiration occurs, the

5473 timer is reloaded with the repetition interval and continues counting. With the  
 5474 specified facilities, this is accomplished by setting the *it\_value* member of the *value*  
 5475 argument to the desired initial expiration time and the *it\_interval* member to the desired  
 5476 repetition interval.

5477 For both of these types of timers, the time of the initial timer expiration can be specified in  
 5478 two ways:

- 5479 1. Relative (to the current time)
- 5480 2. Absolute

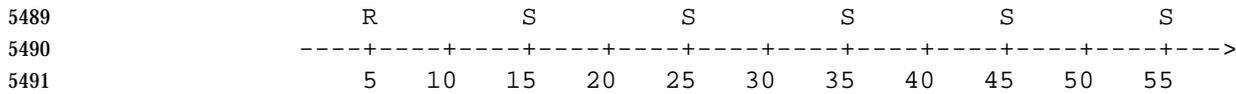
5481 • Examples of Using Realtime Timers

5482 In the diagrams below, *S* indicates a program schedule, *R* shows a schedule method request,  
 5483 and *E* suggests an internal operating system event.

5484 — Periodic Timer: Data Logging

5485 During an experiment, it might be necessary to log realtime data periodically to an  
 5486 internal buffer or to a mass storage device. With a periodic scheduling method, a logging  
 5487 module can be started automatically at fixed time intervals to log the data.

5488 Program schedule is requested every 10 seconds.



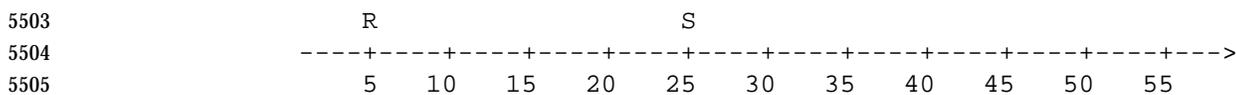
5492 [Time (in Seconds)]

5493 To achieve this type of scheduling using the specified facilities, one would allocate a per-  
 5494 process timer based on clock ID `CLOCK_REALTIME`. Then the timer would be armed via  
 5495 a call to `timer_settime()` with the `TIMER_ABSTIME` flag reset, and with an initial  
 5496 expiration value and a repetition interval of 10 seconds.

5497 — One-shot Timer (Relative Time): Device Initialization

5498 In an emission test environment, large sample bags are used to capture the exhaust from  
 5499 a vehicle. The exhaust is purged from these bags before each and every test. With a one-  
 5500 shot timer, a module could initiate the purge function and then suspend itself for a  
 5501 predetermined period of time while the sample bags are prepared.

5502 Program schedule requested 20 seconds after call is issued.



5506 [Time (in Seconds)]

5507 To achieve this type of scheduling using the specified facilities, one would allocate a per-  
 5508 process timer based on clock ID `CLOCK_REALTIME`. Then the timer would be armed via  
 5509 a call to `timer_settime()` with the `TIMER_ABSTIME` flag reset, and with an initial  
 5510 expiration value of 20 seconds and a repetition interval of zero.

5511 Note that if the program wishes merely to suspend itself for the specified interval, it  
 5512 could more easily use `nanosleep()`.

5513 — One-shot Timer (Absolute Time): Data Transmission



5557 • Relationship of Timers to Clocks

5558 The relationship between clocks and timers armed with an absolute time is straightforward:  
 5559 a timer expiration signal is requested when the associated clock reaches or exceeds the  
 5560 specified time. The relationship between clocks and timers armed with a relative time (an  
 5561 interval) is less obvious, but not unintuitive. In this case, a timer expiration signal is  
 5562 requested when the specified interval, *as measured by the associated clock*, has passed. For the  
 5563 required CLOCK\_REALTIME clock, this allows timer expiration signals to be requested at  
 5564 specified “wall clock” times (absolute), or when a specified interval of “realtime” has passed  
 5565 (relative). For an implementation-defined clock—say, a process virtual time clock—timer  
 5566 expirations could be requested when the process has used a specified total amount of virtual  
 5567 time (absolute), or when it has used a specified *additional* amount of virtual time (relative).

5568 The interfaces also allow flexibility in the implementation of the functions. For example, an  
 5569 implementation could convert all absolute times to intervals by subtracting the clock value at  
 5570 the time of the call from the requested expiration time and “counting down” at the  
 5571 supported resolution. Or it could convert all relative times to absolute expiration time by  
 5572 adding in the clock value at the time of the call and comparing the clock value to the  
 5573 expiration time at the supported resolution. Or it might even choose to maintain absolute  
 5574 times as absolute and compare them to the clock value at the supported resolution for  
 5575 absolute timers, and maintain relative times as intervals and count them down at the  
 5576 resolution supported for relative timers. The choice will be driven by efficiency  
 5577 considerations and the underlying hardware or software clock implementation.

5578 • Data Definitions for Clocks and Timers

5579 IEEE Std 1003.1-200x uses a time representation capable of supporting nanosecond resolution  
 5580 timers for the following reasons:

- 5581 — To enable IEEE Std 1003.1-200x to represent those computer systems already using  
 5582 nanosecond or submicrosecond resolution clocks.
- 5583 — To accommodate those per-process timers that might need nanoseconds to specify an  
 5584 absolute value of system-wide clocks, even though the resolution of the per-process timer  
 5585 may only be milliseconds, or *vice versa*.
- 5586 — Because the number of nanoseconds in a second can be represented in 32 bits.

5587 Time values are represented in the **timespec** structure. The *tv\_sec* member is of type **time\_t**  
 5588 so that this member is compatible with time values used by POSIX.1 functions and the ISO C  
 5589 standard. The *tv\_nsec* member is a **signed long** in order to simplify and clarify code that  
 5590 decrements or finds differences of time values. Note that because 1 billion (number of  
 5591 nanoseconds per second) is less than half of the value representable by a signed 32-bit value,  
 5592 it is always possible to add two valid fractional seconds represented as integral nanoseconds  
 5593 without overflowing the signed 32-bit value.

5594 A maximum allowable resolution for the CLOCK\_REALTIME clock of 20 ms (1/50 seconds)  
 5595 was chosen to allow line frequency clocks in European countries to be conforming. 60 Hz  
 5596 clocks in the U.S. will also be conforming, as will finer granularity clocks, although a Strictly  
 5597 Conforming Application cannot assume a granularity of less than 20 ms (1/50 seconds).

5598 The minimum allowable maximum time allowed for the CLOCK\_REALTIME clock and the  
 5599 function *nanosleep()*, and timers created with *clock\_id*=CLOCK\_REALTIME, is determined by  
 5600 the fact that the *tv\_sec* member is of type **time\_t**.

5601 IEEE Std 1003.1-200x specifies that timer expirations shall not be delivered early, nor shall  
 5602 *nanosleep()* return early due to quantization error. IEEE Std 1003.1-200x discusses the various  
 5603 implementations of *alarm()* in the rationale and states that implementations that do not

5604 allow alarm signals to occur early are the most appropriate, but refrained from mandating  
5605 this behavior. Because of the importance of predictability to realtime applications,  
5606 IEEE Std 1003.1-200x takes a stronger stance.

5607 The developers of IEEE Std 1003.1-200x considered using a time representation that differs  
5608 from POSIX.1b in the second 32 bit of the 64-bit value. Whereas POSIX.1b defines this field  
5609 as a fractional second in nanoseconds, the other methodology defines this as a binary fraction  
5610 of one second, with the radix point assumed before the most significant bit.

5611 POSIX.1b is a software, source-level standard and most of the benefits of the alternate  
5612 representation are enjoyed by hardware implementations of clocks and algorithms. It was  
5613 felt that mandating this format for POSIX.1b clocks and timers would unnecessarily burden  
5614 the application writer with writing, possibly non-portable, multiple precision arithmetic  
5615 packages to perform conversion between binary fractions and integral units such as  
5616 nanoseconds, milliseconds, and so on.

#### 5617 **Rationale for the Monotonic Clock**

5618 For those applications that use time services to achieve realtime behavior, changing the value of  
5619 the clock on which these services rely may cause erroneous timing behavior. For these  
5620 applications, it is necessary to have a monotonic clock which cannot run backwards, and which  
5621 has a maximum clock jump that is required to be documented by the implementation.  
5622 Additionally, it is desirable (but not required by IEEE Std 1003.1-200x) that the monotonic clock  
5623 increases its value uniformly. This clock should not be affected by changes to the system time;  
5624 for example, to synchronize the clock with an external source or to account for leap seconds.  
5625 Such changes would cause errors in the measurement of time intervals for those time services  
5626 that use the absolute value of the clock.

5627 One could argue that by defining the behavior of time services when the value of a clock is  
5628 changed, deterministic realtime behavior can be achieved. For example, one could specify that  
5629 relative time services should be unaffected by changes in the value of a clock. However, there  
5630 are time services that are based upon an absolute time, but that are essentially intended as  
5631 relative time services. For example, *pthread\_cond\_timedwait()* uses an absolute time to allow it to  
5632 wake up after the required interval despite spurious wakeups. Although sometimes the  
5633 *pthread\_cond\_timedwait()* timeouts are absolute in nature, there are many occasions in which  
5634 they are relative, and their absolute value is determined from the current time plus a relative  
5635 time interval. In this latter case, if the clock changes while the thread is waiting, the wait interval  
5636 will not be the expected length. If a *pthread\_cond\_timedwait()* function were created that would  
5637 take a relative time, it would not solve the problem because to retain the intended “deadline” a  
5638 thread would need to compensate for latency due to the spurious wakeup, and preemption  
5639 between wakeup and the next wait.

5640 The solution is to create a new monotonic clock, whose value does not change except for the  
5641 regular ticking of the clock, and use this clock for implementing the various relative timeouts  
5642 that appear in the different POSIX interfaces, as well as allow *pthread\_cond\_timedwait()* to choose  
5643 this new clock for its timeout. A new *clock\_nanosleep()* function is created to allow an application  
5644 to take advantage of this newly defined clock. Notice that the monotonic clock may be  
5645 implemented using the same hardware clock as the system clock.

5646 Relative timeouts for *sigtimedwait()* and *aio\_suspend()* have been redefined to use the monotonic  
5647 clock, if present. The *alarm()* function has not been redefined, because the same effect but with  
5648 better resolution can be achieved by creating a timer (for which the appropriate clock may be  
5649 chosen).

5650 The *pthread\_cond\_timedwait()* function has been treated in a different way, compared to other  
5651 functions with absolute timeouts, because it is used to wait for an event, and thus it may have a

5652 deadline, while the other timeouts are generally used as an error recovery mechanism, and for  
5653 them the use of the monotonic clock is not so important. Since the desired timeout for the  
5654 *pthread\_cond\_timedwait()* function may either be a relative interval, or an absolute time of day  
5655 deadline, a new initialization attribute has been created for condition variables, to specify the  
5656 clock that shall be used for measuring the timeout in a call to *pthread\_cond\_timedwait()*. In this  
5657 way, if a relative timeout is desired, the monotonic clock will be used; if an absolute deadline is  
5658 required instead, the `CLOCK_REALTIME` or another appropriate clock may be used. This  
5659 capability has not been added to other functions with absolute timeouts because for those  
5660 functions the expected use of the timeout is mostly to prevent errors, and not so often to meet  
5661 precise deadlines. As a consequence, the complexity of adding this capability is not justified by  
5662 its perceived application usage.

5663 The *nanosleep()* function has not been modified with the introduction of the monotonic clock.  
5664 Instead, a new *clock\_nanosleep()* function has been created, in which the desired clock may be  
5665 specified in the function call.

5666 • History of Resolution Issues

5667 Due to the shift from relative to absolute timeouts in IEEE Std 1003.1d-1999, the amendments  
5668 to the *sem\_timedwait()*, *pthread\_mutex\_timedlock()*, *mq\_timedreceive()*, and *mq\_timedsend()*  
5669 functions of that standard have been removed. Those amendments specified that  
5670 `CLOCK_MONOTONIC` would be used for the (relative) timeouts if the Monotonic Clock  
5671 option was supported.

5672 Having these functions continue to be tied solely to `CLOCK_MONOTONIC` would not  
5673 work. Since the absolute value of a time value obtained from `CLOCK_MONOTONIC` is  
5674 unspecified, under the absolute timeouts interface, applications would behave differently  
5675 depending on whether the Monotonic Clock option was supported or not (because the  
5676 absolute value of the clock would have different meanings in either case).

5677 Two options were considered:

- 5678 1. Leave the current behavior unchanged, which specifies the `CLOCK_REALTIME` clock  
5679 for these (absolute) timeouts, to allow portability of applications between  
5680 implementations supporting or not the Monotonic Clock option.
- 5681 2. Modify these functions in the way that *pthread\_cond\_timedwait()* was modified to allow  
5682 a choice of clock, so that an application could use `CLOCK_REALTIME` when it is trying  
5683 to achieve an absolute timeout and `CLOCK_MONOTONIC` when it is trying to achieve  
5684 a relative timeout.

5685 It was decided that the features of `CLOCK_MONOTONIC` are not as critical to these  
5686 functions as they are to *pthread\_cond\_timedwait()*. The *pthread\_cond\_timedwait()* function is  
5687 given a relative timeout; the timeout may represent a deadline for an event. When these  
5688 functions are given relative timeouts, the timeouts are typically for error recovery purposes  
5689 and need not be so precise.

5690 Therefore, it was decided that these functions should be tied to `CLOCK_REALTIME` and not  
5691 complicated by being given a choice of clock.

5692           **Execution Time Monitoring**

## 5693           • Introduction

5694           The main goals of the execution time monitoring facilities defined in this chapter are to  
5695           measure the execution time of processes and threads and to allow an application to establish  
5696           CPU time limits for these entities.

5697           The analysis phase of time-critical realtime systems often relies on the measurement of  
5698           execution times of individual threads or processes to determine whether the timing  
5699           requirements will be met. Also, performance analysis techniques for soft deadline realtime  
5700           systems rely heavily on the determination of these execution times. The execution time  
5701           monitoring functions provide application developers with the ability to measure these  
5702           execution times online and open the possibility of dynamic execution-time analysis and  
5703           system reconfiguration, if required.

5704           The second goal of allowing an application to establish execution time limits for individual  
5705           processes or threads and detecting when they overrun allows program robustness to be  
5706           increased by enabling online checking of the execution times.

5707           If errors are detected—possibly because of erroneous program constructs, the existence of  
5708           errors in the analysis phase, or a burst of event arrivals—online detection and recovery is  
5709           possible in a portable way. This feature can be extremely important for many time-critical  
5710           applications. Other applications require trapping CPU-time errors as a normal way to exit an  
5711           algorithm; for instance, some realtime artificial intelligence applications trigger a number of  
5712           independent inference processes of varying accuracy and speed, limit how long they can run,  
5713           and pick the best answer available when time runs out. In many periodic systems, overrun  
5714           processes are simply restarted in the next resource period, after necessary end-of-period  
5715           actions have been taken. This allows algorithms that are inherently data-dependent to be  
5716           made predictable.

5717           The interface that appears in this chapter defines a new type of clock, the CPU-time clock,  
5718           which measures execution time. Each process or thread can invoke the clock and timer  
5719           functions defined in POSIX.1 to use them. Functions are also provided to access the CPU-  
5720           time clock of other processes or threads to enable remote monitoring of these clocks.  
5721           Monitoring of threads of other processes is not supported, since these threads are not visible  
5722           from outside of their own process with the interfaces defined in POSIX.1.

## 5723           • Execution Time Monitoring Interface

5724           The clock and timer interface defined in POSIX.1 historically only defined one clock, which  
5725           measures wall-clock time. The requirements for measuring execution time of processes and  
5726           threads, and setting limits to their execution time by detecting when they overrun, can be  
5727           accomplished with that interface if a new kind of clock is defined. These new clocks measure  
5728           execution time, and one is associated with each process and with each thread. The clock  
5729           functions currently defined in POSIX.1 can be used to read and set these CPU-time clocks,  
5730           and timers can be created using these clocks as their timing base. These timers can then be  
5731           used to send a signal when some specified execution time has been exceeded. The CPU-time  
5732           clocks of each process or thread can be accessed by using the symbols  
5733           CLOCK\_PROCESS\_CPUTIME\_ID or CLOCK\_THREAD\_CPUTIME\_ID.

5734           The clock and timer interface defined in POSIX.1 and extended with the new kind of CPU-  
5735           time clock would only allow processes or threads to access their own CPU-time clocks.  
5736           However, many realtime systems require the possibility of monitoring the execution time of  
5737           processes or threads from independent monitoring entities. In order to allow applications to  
5738           construct independent monitoring entities that do not require cooperation from or  
5739           modification of the monitored entities, two functions have been added: *clock\_getcpucllockid()*,

5740 for accessing CPU-time clocks of other processes, and *pthread\_getcpuclockid()*, for accessing  
5741 CPU-time clocks of other threads. These functions return the clock identifier associated with  
5742 the process or thread specified in the call. These clock IDs can then be used in the rest of the  
5743 clock function calls.

5744 The clocks accessed through these functions could also be used as a timing base for the  
5745 creation of timers, thereby allowing independent monitoring entities to limit the CPU-time  
5746 consumed by other entities. However, this possibility would imply additional complexity  
5747 and overhead because of the need to maintain a timer queue for each process or thread, to  
5748 store the different expiration times associated with timers created by different processes or  
5749 threads. The working group decided this additional overhead was not justified by  
5750 application requirements. Therefore, creation of timers attached to the CPU-time clocks of  
5751 other processes or threads has been specified as implementation-defined.

5752 • Overhead Considerations

5753 The measurement of execution time may introduce additional overhead in the thread  
5754 scheduling, because of the need to keep track of the time consumed by each of these entities.  
5755 In library-level implementations of threads, the efficiency of scheduling could be somehow  
5756 compromised because of the need to make a kernel call, at each context switch, to read the  
5757 process CPU-time clock. Consequently, a thread creation attribute called *cpu-clock-*  
5758 *requirement* was defined, to allow threads to disconnect their respective CPU-time clocks.  
5759 However, the Ballot Group considered that this attribute itself introduced some overhead,  
5760 and that in current implementations it was not worth the effort. Therefore, the attribute was  
5761 deleted, and thus thread CPU-time clocks are required for all threads if the Thread CPU-Time  
5762 Clocks option is supported.

5763 • Accuracy of CPU-time Clocks

5764 The mechanism used to measure the execution time of processes and threads is specified in  
5765 IEEE Std 1003.1-200x as implementation-defined. The reason for this is that both the  
5766 underlying hardware and the implementation architecture have a very strong influence on  
5767 the accuracy achievable for measuring CPU time. For some implementations, the  
5768 specification of strict accuracy requirements would represent very large overheads, or even  
5769 the impossibility of being implemented.

5770 Since the mechanism for measuring execution time is implementation-defined, realtime  
5771 applications will be able to take advantage of accurate implementations using a portable  
5772 interface. Of course, strictly conforming applications cannot rely on any particular degree of  
5773 accuracy, in the same way as they cannot rely on a very accurate measurement of wall clock  
5774 time. There will always exist applications whose accuracy or efficiency requirements on the  
5775 implementation are more rigid than the values defined in IEEE Std 1003.1-200x or any other  
5776 standard.

5777 In any case, there is a minimum set of characteristics that realtime applications would expect  
5778 from most implementations. One such characteristic is that the sum of all the execution times  
5779 of all the threads in a process equals the process execution time, when no CPU-time clocks  
5780 are disabled. This need not always be the case because implementations may differ in how  
5781 they account for time during context switches. Another characteristic is that the sum of the  
5782 execution times of all processes in a system equals the number of processors, multiplied by  
5783 the elapsed time, assuming that no processor is idle during that elapsed time. However, in |  
5784 some implementations it might not be possible to relate CPU-time to elapsed time. For |  
5785 example, in a heterogeneous multi-processor system in which each processor runs at a |  
5786 different speed, an implementation may choose to define each “second” of CPU-time to be a  
5787 certain number of “cycles” that a CPU has executed.

- 5788       • Existing Practice
- 5789       Measuring and limiting the execution time of each concurrent activity are common features  
5790       of most industrial implementations of realtime systems. Almost all critical realtime systems  
5791       are currently built upon a cyclic executive. With this approach, a regular timer interrupt kicks  
5792       off the next sequence of computations. It also checks that the current sequence has  
5793       completed. If it has not, then some error recovery action can be undertaken (or at least an  
5794       overrun is avoided). Current software engineering principles and the increasing complexity  
5795       of software are driving application developers to implement these systems on multi-  
5796       threaded or multi-process operating systems. Therefore, if a POSIX operating system is to be  
5797       used for this type of application, then it must offer the same level of protection.
- 5798       Execution time clocks are also common in most UNIX implementations, although these  
5799       clocks usually have requirements different from those of realtime applications. The POSIX.1  
5800       *times()* function supports the measurement of the execution time of the calling process, and  
5801       its terminated child processes. This execution time is measured in clock ticks and is supplied  
5802       as two different values with the user and system execution times, respectively. BSD supports  
5803       the function *getrusage()*, which allows the calling process to get information about the  
5804       resources used by itself and/or all of its terminated child processes. The resource usage  
5805       includes user and system CPU time. Some UNIX systems have options to specify high  
5806       resolution (up to one microsecond) CPU time clocks using the *times()* or the *getrusage()*  
5807       functions.
- 5808       The *times()* and *getrusage()* interfaces do not meet important realtime requirements, such as  
5809       the possibility of monitoring execution time from a different process or thread, or the  
5810       possibility of detecting an execution time overrun. The latter requirement is supported in  
5811       some UNIX implementations that are able to send a signal when the execution time of a  
5812       process has exceeded some specified value. For example, BSD defines the functions  
5813       *getitimer()* and *setitimer()*, which can operate either on a realtime clock (wall-clock), or on  
5814       virtual-time or profile-time clocks which measure CPU time in two different ways. These  
5815       functions do not support access to the execution time of other processes.
- 5816       IBM's MVS operating system supports per-process and per-thread execution time clocks. It  
5817       also supports limiting the execution time of a given process.
- 5818       Given all this existing practice, the working group considered that the POSIX.1 clocks and  
5819       timers interface was appropriate to meet most of the requirements that realtime applications  
5820       have for execution time clocks. Functions were added to get the CPU time clock IDs, and to  
5821       allow/disallow the thread CPU time clocks (in order to preserve the efficiency of some  
5822       implementations of threads).
- 5823       • Clock Constants
- 5824       The definition of the manifest constants `CLOCK_PROCESS_CPUTIME_ID` and  
5825       `CLOCK_THREAD_CPUTIME_ID` allows processes or threads, respectively, to access their  
5826       own execution-time clocks. However, given a process or thread, access to its own execution-  
5827       time clock is also possible if the clock ID of this clock is obtained through a call to  
5828       *clock\_getcpuclockid()* or *pthread\_getcpuclockid()*. Therefore, these constants are not necessary  
5829       and could be deleted to make the interface simpler. Their existence saves one system call in  
5830       the first access to the CPU-time clock of each process or thread. The working group  
5831       considered this issue and decided to leave the constants in IEEE Std 1003.1-200x because they  
5832       are closer to the POSIX.1b use of clock identifiers.
- 5833       • Library Implementations of Threads
- 5834       In library implementations of threads, kernel entities and library threads can coexist. In this  
5835       case, if the CPU-time clocks are supported, most of the clock and timer functions will need to

5836 have two implementations: one in the thread library, and one in the system calls library. The  
5837 main difference between these two implementations is that the thread library  
5838 implementation will have to deal with clocks and timers that reside in the thread space,  
5839 while the kernel implementation will operate on timers and clocks that reside in kernel space.  
5840 In the library implementation, if the clock ID refers to a clock that resides in the kernel, a  
5841 kernel call will have to be made. The correct version of the function can be chosen by  
5842 specifying the appropriate order for the libraries during the link process.

5843 • History of Resolution Issues: Deletion of the *enable* Attribute

5844 In the draft corresponding to the first balloting round, CPU-time clocks had an attribute  
5845 called *enable*. This attribute was introduced by the working group to allow implementations  
5846 to avoid the overhead of measuring execution time for those processes or threads for which  
5847 this measurement was not required. However, the *enable* attribute got several ballot  
5848 objections. The main reason was that processes are already required to measure execution  
5849 time by the POSIX.1 *times()* function. Consequently, the *enable* attribute was considered  
5850 unnecessary, and was deleted from the draft.

### 5851 **Rationale Relating to Timeouts**

5852 • Requirements for Timeouts

5853 Realtime systems which must operate reliably over extended periods without human  
5854 intervention are characteristic in embedded applications such as avionics, machine control,  
5855 and space exploration, as well as more mundane applications such as cable TV, security  
5856 systems, and plant automation. A multi-tasking paradigm, in which many independent  
5857 and/or cooperating software functions relinquish the processor(s) while waiting for a  
5858 specific stimulus, resource, condition, or operation completion, is very useful in producing  
5859 well engineered programs for such systems. For such systems to be robust and fault-tolerant,  
5860 expected occurrences that are unduly delayed or that never occur must be detected so that  
5861 appropriate recovery actions may be taken. This is difficult if there is no way for a task to  
5862 regain control of a processor once it has relinquished control (blocked) awaiting an  
5863 occurrence which, perhaps because of corrupted code, hardware malfunction, or latent  
5864 software bugs, will not happen when expected. Therefore, the common practice in realtime  
5865 operating systems is to provide a capability to timeout such blocking services. Although  
5866 there are several methods to achieve this already defined by POSIX, none are as reliable or  
5867 efficient as initiating a timeout simultaneously with initiating a blocking service. This is  
5868 especially critical in hard-realtime embedded systems because the processors typically have  
5869 little time reserve, and allowed fault recovery times are measured in milliseconds rather than  
5870 seconds.

5871 The working group largely agreed that such timeouts were necessary and ought to become  
5872 part of IEEE Std 1003.1-200x, particularly vendors of realtime operating systems whose  
5873 customers had already expressed a strong need for timeouts. There was some resistance to  
5874 inclusion of timeouts in IEEE Std 1003.1-200x because the desired effect, fault tolerance,  
5875 could, in theory, be achieved using existing facilities and alternative software designs, but  
5876 there was no compelling evidence that realtime system designers would embrace such  
5877 designs at the sacrifice of performance and/or simplicity.

5878 • Which Services should be Timed Out?

5879 Originally, the working group considered the prospect of providing timeouts on all blocking  
5880 services, including those currently existing in POSIX.1, POSIX.1b, and POSIX.1c, and future  
5881 interfaces to be defined by other working groups, as sort of a general policy. This was rather  
5882 quickly rejected because of the scope of such a change, and the fact that many of those  
5883 services would not normally be used in a realtime context. More traditional timesharing

5884 solutions to timeout would suffice for most of the POSIX.1 interfaces, while others had  
 5885 asynchronous alternatives which, while more complex to utilize, would be adequate for  
 5886 some realtime and all non-realtime applications.

5887 The list of potential candidates for timeouts was narrowed to the following for further  
 5888 consideration:

5889 — POSIX.1b

5890 — *sem\_wait()*

5891 — *mq\_receive()*

5892 — *mq\_send()*

5893 — *lio\_listio()*

5894 — *aio\_suspend()*

5895 — *sigwait()* (timeout already implemented by *sigtimedwait()*)

5896 — POSIX.1c

5897 — *pthread\_mutex\_lock()*

5898 — *pthread\_join()*

5899 — *pthread\_cond\_wait()* (timeout already implemented by *pthread\_cond\_timedwait()*)

5900 — POSIX.1

5901 — *read()*

5902 — *write()*

5903 After further review by the working group, the *lio\_listio()*, *read()*, and *write()* functions (all  
 5904 forms of blocking synchronous I/O) were eliminated from the list because of the following:

5905 — Asynchronous alternatives exist

5906 — Timeouts can be implemented, albeit non-portably, in device drivers

5907 — A strong desire not to introduce modifications to POSIX.1 interfaces

5908 The working group ultimately rejected *pthread\_join()* since both that interface and a timed  
 5909 variant of that interface are non-minimal and may be implemented as a function. See below  
 5910 for a library implementation of *pthread\_join()*.

5911 Thus, there was a consensus among the working group members to add timeouts to 4 of the  
 5912 remaining 5 functions (the timeout for *aio\_suspend()* was ultimately added directly to  
 5913 POSIX.1b, while the others were added by POSIX.1d). However, *pthread\_mutex\_lock()*  
 5914 remained contentious.

5915 Many feel that *pthread\_mutex\_lock()* falls into the same class as the other functions; that is, it  
 5916 is desirable to timeout a mutex lock because a mutex may fail to be unlocked due to errant or  
 5917 corrupted code in a critical section (looping or branching outside of the unlock code), and  
 5918 therefore is equally in need of a reliable, simple, and efficient timeout. In fact, since mutexes  
 5919 are intended to guard small critical sections, most *pthread\_mutex\_lock()* calls would be  
 5920 expected to obtain the lock without blocking nor utilizing any kernel service, even in  
 5921 implementations of threads with global contention scope; the timeout alternative need only  
 5922 be considered after it is determined that the thread must block.

5923 Those opposed to timing out mutexes feel that the very simplicity of the mutex is  
 5924 compromised by adding a timeout semantic, and that to do so is senseless. They claim that if

5925 a timed mutex is really deemed useful by a particular application, then it can be constructed  
 5926 from the facilities already in POSIX.1b and POSIX.1c. The following two C-language library  
 5927 implementations of mutex locking with timeout represent the solutions offered (in both  
 5928 implementations, the timeout parameter is specified as absolute time, not relative time as in  
 5929 the proposed POSIX.1c interfaces).

5930 • Spinlock Implementation

```

5931     #include <pthread.h>
5932     #include <time.h>
5933     #include <errno.h>

5934     int pthread_mutex_timedlock(pthread_mutex_t *mutex,
5935                               const struct timespec *timeout)
5936     {
5937         struct timespec timenow;

5938         while (pthread_mutex_trylock(mutex) == EBUSY)
5939             {
5940                 clock_gettime(CLOCK_REALTIME, &timenow);
5941                 if (timespec_cmp(&timenow, timeout) >= 0)
5942                     {
5943                         return ETIMEDOUT;
5944                     }
5945                 pthread_yield();
5946             }
5947         return 0;
5948     }

```

5949 The Spinlock implementation is generally unsuitable for any application using priority-based  
 5950 thread scheduling policies such as SCHED\_FIFO or SCHED\_RR, since the mutex could  
 5951 currently be held by a thread of lower priority within the same allocation domain, but since  
 5952 the waiting thread never blocks, only threads of equal or higher priority will ever run, and  
 5953 the mutex cannot be unlocked. Setting priority inheritance or priority ceiling protocol on the  
 5954 mutex does not solve this problem, since the priority of a mutex owning thread is only  
 5955 boosted if higher priority threads are blocked waiting for the mutex; clearly not the case for  
 5956 this spinlock.

5957 • Condition Wait Implementation

```

5958     #include <pthread.h>
5959     #include <time.h>
5960     #include <errno.h>

5961     struct timed_mutex
5962     {
5963         int locked;
5964         pthread_mutex_t mutex;
5965         pthread_cond_t cond;
5966     };
5967     typedef struct timed_mutex timed_mutex_t;

5968     int timed_mutex_lock(timed_mutex_t *tm,
5969                       const struct timespec *timeout)
5970     {
5971         int timedout=FALSE;
5972         int error_status;

```

```

5973     pthread_mutex_lock(&tm->mutex);
5974     while (tm->locked && !timedout)
5975     {
5976         if ((error_status=pthread_cond_timedwait(&tm->cond,
5977         &tm->mutex,
5978         timeout))!=0)
5979         {
5980             if (error_status==ETIMEDOUT) timedout = TRUE;
5981         }
5982     }
5983     if(timedout)
5984     {
5985         pthread_mutex_unlock(&tm->mutex);
5986         return ETIMEDOUT;
5987     }
5988     else
5989     {
5990         tm->locked = TRUE;
5991         pthread_mutex_unlock(&tm->mutex);
5992         return 0;
5993     }
5994 }
5995 void timed_mutex_unlock(timed_mutex_t *tm)
5996 {
5997     pthread_mutex_lock(&tm->mutex); / for case assignment not atomic /
5998     tm->locked = FALSE;
5999     pthread_mutex_unlock(&tm->mutex);
6000     pthread_cond_signal(&tm->cond);
6001 }

```

6002 The Condition Wait implementation effectively substitutes the *pthread\_cond\_timedwait()*  
6003 function (which is currently timed out) for the desired *pthread\_mutex\_timedlock()*. Since waits  
6004 on condition variables currently do not include protocols which avoid priority inversion, this  
6005 method is generally unsuitable for realtime applications because it does not provide the same  
6006 priority inversion protection as the untimed *pthread\_mutex\_lock()*. Also, for any given  
6007 implementations of the current mutex and condition variable primitives, this library  
6008 implementation has a performance cost at least 2.5 times that of the untimed  
6009 *pthread\_mutex\_lock()* even in the case where the timed mutex is readily locked without  
6010 blocking (the interfaces required for this case are shown in bold). Even in uniprocessors or  
6011 where assignment is atomic, at least an additional *pthread\_cond\_signal()* is required.  
6012 *pthread\_mutex\_timedlock()* could be implemented at effectively no performance penalty in  
6013 this case because the timeout parameters need only be considered after it is determined that  
6014 the mutex cannot be locked immediately.

6015 Thus it has not yet been shown that the full semantics of mutex locking with timeout can be  
6016 efficiently and reliably achieved using existing interfaces. Even if the existence of an  
6017 acceptable library implementation were proven, it is difficult to justify why the interface  
6018 itself should not be made portable, especially considering approval for the other four  
6019 timeouts.

6020 • Rationale for Library Implementation of *pthread\_timedjoin()*

```

6021     Library implementation of pthread_timedjoin():
6022     /*
6023     * Construct a thread variety entirely from existing functions
6024     * with which a join can be done, allowing the join to time out.
6025     */
6026     #include <pthread.h>
6027     #include <time.h>
6028
6028     struct timed_thread {
6029         pthread_t t;
6030         pthread_mutex_t m;
6031         int exiting;
6032         pthread_cond_t exit_c;
6033         void *(*start_routine)(void *arg);
6034         void *arg;
6035         void *status;
6036     };
6037
6037     typedef struct timed_thread *timed_thread_t;
6038     static pthread_key_t timed_thread_key;
6039     static pthread_once_t timed_thread_once = PTHREAD_ONCE_INIT;
6040
6040     static void timed_thread_init()
6041     {
6042         pthread_key_create(&timed_thread_key, NULL);
6043     }
6044
6044     static void *timed_thread_start_routine(void *args)
6045
6045     /*
6046     * Routine to establish thread-specific data value and run the actual
6047     * thread start routine which was supplied to timed_thread_create().
6048     */
6049     {
6050         timed_thread_t tt = (timed_thread_t) args;
6051
6051         pthread_once(&timed_thread_once, timed_thread_init);
6052         pthread_setspecific(timed_thread_key, (void *)tt);
6053         timed_thread_exit((tt->start_routine)(tt->arg));
6054     }
6055
6055     int timed_thread_create(timed_thread_t ttp, const pthread_attr_t *attr,
6056         void *(*start_routine)(void *), void *arg)
6057
6057     /*
6058     * Allocate a thread which can be used with timed_thread_join().
6059     */
6060     {
6061         timed_thread_t tt;
6062         int result;
6063
6063         tt = (timed_thread_t) malloc(sizeof(struct timed_thread));
6064         pthread_mutex_init(&tt->m, NULL);
6065         tt->exiting = FALSE;
6066         pthread_cond_init(&tt->exit_c, NULL);
6067         tt->start_routine = start_routine;

```

```

6068         tt->arg = arg;
6069         tt->status = NULL;

6070         if ((result = pthread_create(&tt->t, attr,
6071             timed_thread_start_routine, (void *)tt)) != 0) {
6072             free(tt);
6073             return result;
6074         }

6075         pthread_detach(tt->t);
6076         ttp = tt;
6077         return 0;
6078     }

6079     int timed_thread_join(timed_thread_t tt,
6080         struct timespec *timeout,
6081         void **status)
6082     {
6083         int result;

6084         pthread_mutex_lock(&tt->m);
6085         result = 0;
6086         /*
6087          * Wait until the thread announces that it is exiting,
6088          * or until timeout.
6089          */
6090         while (result == 0 && ! tt->exiting) {
6091             result = pthread_cond_timedwait(&tt->exit_c, &tt->m, timeout);
6092         }
6093         pthread_mutex_unlock(&tt->m);
6094         if (result == 0 && tt->exiting) {
6095             *status = tt->status;
6096             free((void *)tt);
6097             return result;
6098         }
6099         return result;
6100     }

6101     void timed_thread_exit(void *status)
6102     {
6103         timed_thread_t tt;
6104         void *specific;

6105         if ((specific=pthread_getspecific(timed_thread_key)) == NULL){
6106             /*
6107              * Handle cases which won't happen with correct usage.
6108              */
6109             pthread_exit( NULL);
6110         }
6111         tt = (timed_thread_t) specific;
6112         pthread_mutex_lock(&tt->m);
6113         /*
6114          * Tell a joiner that we're exiting.
6115          */
6116         tt->status = status;

```

```

6117         tt->exiting = TRUE;
6118         pthread_cond_signal(&tt->exit_c);
6119         pthread_mutex_unlock(&tt->m);
6120         /*
6121          * Call pthread_exit() to call destructors and really
6122          * exit the thread.
6123          */
6124         pthread_exit(NULL);
6125     }

```

6126 The *pthread\_join()* C-language example shown above demonstrates that it is possible, using  
6127 existing pthread facilities, to construct a variety of thread which allows for joining such a  
6128 thread, but which allows the join operation to time out. It does this by using a  
6129 *pthread\_cond\_timedwait()* to wait for the thread to exit. A **timed\_thread\_t** descriptor structure  
6130 is used to pass parameters from the creating thread to the created thread, and from the  
6131 exiting thread to the joining thread. This implementation is roughly equivalent to what a  
6132 normal *pthread\_join()* implementation would do, with the single change being that  
6133 *pthread\_cond\_timedwait()* is used in place of a simple *pthread\_cond\_wait()*.

6134 Since it is possible to implement such a facility entirely from existing pthread interfaces, and  
6135 with roughly equal efficiency and complexity to an implementation which would be  
6136 provided directly by a pthreads implementation, it was the consensus of the working group  
6137 members that any *pthread\_timedjoin()* facility would be unnecessary, and should not be  
6138 provided.

#### 6139 • Form of the Timeout Interfaces

6140 The working group considered a number of alternative ways to add timeouts to blocking  
6141 services. At first, a system interface which would specify a one-shot or persistent timeout to  
6142 be applied to subsequent blocking services invoked by the calling process or thread was  
6143 considered because it allowed all blocking services to be timed out in a uniform manner with  
6144 a single additional interface; this was rather quickly rejected because it could easily result in  
6145 the wrong services being timed out.

6146 It was suggested that a timeout value might be specified as an attribute of the object  
6147 (semaphore, mutex, message queue, and so on), but there was no consensus on this, either on  
6148 a case-by-case basis or for all timeouts.

6149 Looking at the two existing timeouts for blocking services indicates that the working group  
6150 members favor a separate interface for the timed version of a function. However,  
6151 *pthread\_cond\_timedwait()* utilizes an absolute timeout value while *sigtimedwait()* uses a  
6152 relative timeout value. The working group members agreed that relative timeout values are  
6153 appropriate where the timeout mechanism's primary use was to deal with an unexpected or  
6154 error situation, but they are inappropriate when the timeout must expire at a particular time,  
6155 or before a specific deadline. For the timeouts being introduced in IEEE Std 1003.1-200x, the  
6156 working group considered allowing both relative and absolute timeouts as is done with  
6157 POSIX.1b timers, but ultimately favored the simpler absolute timeout form.

6158 An absolute time measure can be easily implemented on top of an interface that specifies  
6159 relative time, by reading the clock, calculating the difference between the current time and  
6160 the desired wake-up time, and issuing a relative timeout call. But there is a race condition  
6161 with this approach because the thread could be preempted after reading the clock, but before  
6162 making the timed out call; in this case, the thread would be awakened later than it should  
6163 and, thus, if the wake up time represented a deadline, it would miss it.

6164 There is also a race condition when trying to build a relative timeout on top of an interface  
6165 that specifies absolute timeouts. In this case, we would have to read the clock to calculate the  
6166 absolute wake-up time as the sum of the current time plus the relative timeout interval. In  
6167 this case, if the thread is preempted after reading the clock but before making the timed out  
6168 call, the thread would be awakened earlier than desired.

6169 But the race condition with the absolute timeouts interface is not as bad as the one that  
6170 happens with the relative timeout interface, because there are simple workarounds. For the  
6171 absolute timeouts interface, if the timing requirement is a deadline, we can still meet this  
6172 deadline because the thread woke up earlier than the deadline. If the timeout is just used as  
6173 an error recovery mechanism, the precision of timing is not really important. If the timing  
6174 requirement is that between actions A and B a minimum interval of time must elapse, we can  
6175 safely use the absolute timeout interface by reading the clock after action A has been started.  
6176 It could be argued that, since the call with the absolute timeout is atomic from the  
6177 application point of view, it is not possible to read the clock after action A, if this action is  
6178 part of the timed out call. But if we look at the nature of the calls for which we specify  
6179 timeouts (locking a mutex, waiting for a semaphore, waiting for a message, or waiting until  
6180 there is space in a message queue), the timeouts that an application would build on these  
6181 actions would not be triggered by these actions themselves, but by some other external  
6182 action. For example, if we want to wait for a message to arrive to a message queue, and wait  
6183 for at least 20 milliseconds, this time interval would start to be counted from some event that  
6184 would trigger both the action that produces the message, as well as the action that waits for  
6185 the message to arrive, and not by the wait-for-message operation itself. In this case, we could  
6186 use the workaround proposed above.

6187 For these reasons, the absolute timeout is preferred over the relative timeout interface.

## 6188 **B.2.9 Threads**

6189 Threads will normally be more expensive than subroutines (or functions, routines, and so on) if  
6190 specialized hardware support is not provided. Nevertheless, threads should be sufficiently  
6191 efficient to encourage their use as a medium to fine-grained structuring mechanism for  
6192 parallelism in an application. Structuring an application using threads then allows it to take  
6193 immediate advantage of any underlying parallelism available in the host environment. This  
6194 means implementors are encouraged to optimize for fast execution at the possible expense of  
6195 efficient utilization of storage. For example, a common thread creation technique is to cache  
6196 appropriate thread data structures. That is, rather than releasing system resources, the  
6197 implementation retains these resources and reuses them when the program next asks to create a  
6198 new thread. If this reuse of thread resources is to be possible, there has to be very little unique  
6199 state associated with each thread, because any such state has to be reset when the thread is  
6200 reused.

### 6201 **Thread Creation Attributes**

6202 Attributes objects are provided for threads, mutexes, and condition variables as a mechanism to  
6203 support probable future standardization in these areas without requiring that the interface itself  
6204 be changed. Attributes objects provide clean isolation of the configurable aspects of threads. For  
6205 example, “stack size” is an important attribute of a thread, but it cannot be expressed portably.  
6206 When porting a threaded program, stack sizes often need to be adjusted. The use of attributes  
6207 objects can help by allowing the changes to be isolated in a single place, rather than being spread  
6208 across every instance of thread creation.

6209 Attributes objects can be used to set up *classes* of threads with similar attributes; for example,  
6210 “threads with large stacks and high priority” or “threads with minimal stacks”. These classes  
6211 can be defined in a single place and then referenced wherever threads need to be created.

6212 Changes to “class” decisions become straightforward, and detailed analysis of each  
6213 *pthread\_create()* call is not required.

6214 The attributes objects are defined as opaque types as an aid to extensibility. If these objects had  
6215 been specified as structures, adding new attributes would force recompilation of all multi-  
6216 threaded programs when the attributes objects are extended; this might not be possible if  
6217 different program components were supplied by different vendors.

6218 Additionally, opaque attributes objects present opportunities for improving performance.  
6219 Argument validity can be checked once when attributes are set, rather than each time a thread is  
6220 created. Implementations will often need to cache kernel objects that are expensive to create.  
6221 Opaque attributes objects provide an efficient mechanism to detect when cached objects become  
6222 invalid due to attribute changes.

6223 Because assignment is not necessarily defined on a given opaque type, implementation-  
6224 dependent default values cannot be defined in a portable way. The solution to this problem is to  
6225 allow attribute objects to be initialized dynamically by attributes object initialization functions,  
6226 so that default values can be supplied automatically by the implementation.

6227 The following proposal was provided as a suggested alternative to the supplied attributes:

- 6228 1. Maintain the style of passing a parameter formed by the bitwise-inclusive OR of flags to  
6229 the initialization routines (*pthread\_create()*, *pthread\_mutex\_init()*, *pthread\_cond\_init()*). The  
6230 parameter containing the flags should be an opaque type for extensibility. If no flags are  
6231 set in the parameter, then the objects are created with default characteristics. An  
6232 implementation may specify implementation-defined flag values and associated behavior.
- 6233 2. If further specialization of mutexes and condition variables is necessary, implementations  
6234 may specify additional procedures that operate on the **pthread\_mutex\_t** and  
6235 **pthread\_cond\_t** objects (instead of on attributes objects).

6236 The difficulties with this solution are:

- 6237 1. A bitmask is not opaque if bits have to be set into bit-vector attributes objects using  
6238 explicitly-coded bitwise-inclusive OR operations. If the set of options exceeds an **int**,  
6239 application programmers need to know the location of each bit. If bits are set or read by  
6240 encapsulation (that is, *get\*()* or *set\*()* functions), then the bitmask is merely an  
6241 implementation of attributes objects as currently defined and should not be exposed to the  
6242 programmer.
- 6243 2. Many attributes are not Boolean or very small integral values. For example, scheduling  
6244 policy may be placed in 3 bits or 4 bits, but priority requires 5 bits or more, thereby taking  
6245 up at least 8 bits out of a possible 16 bits on machines with 16-bit integers. Because of this,  
6246 the bitmask can only reasonably control whether particular attributes are set or not, and it  
6247 cannot serve as the repository of the value itself. The value needs to be specified as a  
6248 function parameter (which is non-extensible), or by setting a structure field (which is non-  
6249 opaque), or by *get\*()* and *set\*()* functions (making the bitmask a redundant addition to the  
6250 attributes objects).

6251 Stack size is defined as an optional attribute because the very notion of a stack is inherently  
6252 machine-dependent. Some implementations may not be able to change the size of the stack, for  
6253 example, and others may not need to because stack pages may be discontinuous and can be  
6254 allocated and released on demand.

6255 The attribute mechanism has been designed in large measure for extensibility. Future extensions  
6256 to the attribute mechanism or to any attributes object defined in IEEE Std 1003.1-200x has to be  
6257 done with care so as not to affect binary-compatibility.

6258 Attribute objects, even if allocated by means of dynamic allocation functions such as *malloc()*,  
6259 may have their size fixed at compile time. This means, for example, a *pthread\_create()* in an  
6260 implementation with extensions to the **pthread\_attr\_t** cannot look beyond the area that the  
6261 binary application assumes is valid. This suggests that implementations should maintain a size  
6262 field in the attributes object, as well as possibly version information, if extensions in different  
6263 directions (possibly by different vendors) are to be accommodated.

#### 6264 **Thread Implementation Models**

6265 There are various thread implementation models. At one end of the spectrum is the “library-  
6266 thread model”. In such a model, the threads of a process are not visible to the operating system  
6267 kernel, and the threads are not kernel scheduled entities. The process is the only kernel  
6268 scheduled entity. The process is scheduled onto the processor by the kernel according to the  
6269 scheduling attributes of the process. The threads are scheduled onto the single kernel scheduled  
6270 entity (the process) by the runtime library according to the scheduling attributes of the threads.  
6271 A problem with this model is that it constrains concurrency. Since there is only one kernel  
6272 scheduled entity (namely, the process), only one thread per process can execute at a time. If the  
6273 thread that is executing blocks on I/O, then the whole process blocks.

6274 At the other end of the spectrum is the “kernel-thread model”. In this model, all threads are  
6275 visible to the operating system kernel. Thus, all threads are kernel scheduled entities, and all  
6276 threads can concurrently execute. The threads are scheduled onto processors by the kernel  
6277 according to the scheduling attributes of the threads. The drawback to this model is that the  
6278 creation and management of the threads entails operating system calls, as opposed to subroutine  
6279 calls, which makes kernel threads heavier weight than library threads.

6280 Hybrids of these two models are common. A hybrid model offers the speed of library threads  
6281 and the concurrency of kernel threads. In hybrid models, a process has some (relatively small)  
6282 number of kernel scheduled entities associated with it. It also has a potentially much larger  
6283 number of library threads associated with it. Some library threads may be bound to kernel  
6284 scheduled entities, while the other library threads are multiplexed onto the remaining kernel  
6285 scheduled entities. There are two levels of thread scheduling:

- 6286 1. The runtime library manages the scheduling of (unbound) library threads onto kernel  
6287 scheduled entities.
- 6288 2. The kernel manages the scheduling of kernel scheduled entities onto processors.

6289 For this reason, a hybrid model is referred to as a *two-level threads scheduling model*. In this model,  
6290 the process can have multiple concurrently executing threads; specifically, it can have as many  
6291 concurrently executing threads as it has kernel scheduled entities.

#### 6292 **Thread-Specific Data**

6293 Many applications require that a certain amount of context be maintained on a per-thread basis  
6294 across procedure calls. A common example is a multi-threaded library routine that allocates  
6295 resources from a common pool and maintains an active resource list for each thread. The  
6296 thread-specific data interface provided to meet these needs may be viewed as a two-dimensional  
6297 array of values with keys serving as the row index and thread IDs as the column index (although  
6298 the implementation need not work this way).

##### 6299 • Models

6300 Three possible thread-specific data models were considered:

- 6301 1. No Explicit Support

6302 A standard thread-specific data interface is not strictly necessary to support  
 6303 applications that require per-thread context. One could, for example, provide a hash  
 6304 function that converted a **pthread\_t** into an integer value that could then be used to  
 6305 index into a global array of per-thread data pointers. This hash function, in conjunction  
 6306 with *pthread\_self()*, would be all the interface required to support a mechanism of this  
 6307 sort. Unfortunately, this technique is cumbersome. It can lead to duplicated code as  
 6308 each set of cooperating modules implements their own per-thread data management  
 6309 schemes.

## 6310 2. Single (**void \***) Pointer

6311 Another technique would be to provide a single word of per-thread storage and a pair  
 6312 of functions to fetch and store the value of this word. The word could then hold a  
 6313 pointer to a block of per-thread memory. The allocation, partitioning, and general use  
 6314 of this memory would be entirely up to the application. Although this method is not as  
 6315 problematic as technique 1, it suffers from interoperability problems. For example, all  
 6316 modules using the per-thread pointer would have to agree on a common usage  
 6317 protocol.

## 6318 3. Key/Value Mechanism

6319 This method associates an opaque key (for example, stored in a variable of type  
 6320 **pthread\_key\_t**) with each per-thread datum. These keys play the role of identifiers for  
 6321 per-thread data. This technique is the most generic and avoids the problems noted  
 6322 above, albeit at the cost of some complexity.

6323 The primary advantage of the third model is its information hiding properties. Modules  
 6324 using this model are free to create and use their own key(s) independent of all other such  
 6325 usage, whereas the other models require that all modules that use thread-specific context  
 6326 explicitly cooperate with all other such modules. The data-independence provided by the  
 6327 third model is worth the additional interface.

### 6328 • Requirements

6329 It is important that it be possible to implement the thread-specific data interface without the  
 6330 use of thread private memory. To do otherwise would increase the weight of each thread,  
 6331 thereby limiting the range of applications for which the threads interfaces provided by  
 6332 IEEE Std 1003.1-200x is appropriate.

6333 The values that one binds to the key via *pthread\_setspecific()* may, in fact, be pointers to  
 6334 shared storage locations available to all threads. It is only the key/value bindings that are  
 6335 maintained on a per-thread basis, and these can be kept in any portion of the address space  
 6336 that is reserved for use by the calling thread (for example, on the stack). Thus, no per-thread  
 6337 MMU state is required to implement the interface. On the other hand, there is nothing in the  
 6338 interface specification to preclude the use of a per-thread MMU state if it is available (for  
 6339 example, the key values returned by *pthread\_key\_create()* could be thread private memory  
 6340 addresses).

### 6341 • Standardization Issues

6342 Thread-specific data is a requirement for a usable thread interface. The binding described in  
 6343 this section provides a portable thread-specific data mechanism for languages that do not  
 6344 directly support a thread-specific storage class. A binding to IEEE Std 1003.1-200x for a  
 6345 language that does include such a storage class need not provide this specific interface.

6346 If a language were to include the notion of thread-specific storage, it would be desirable (but  
 6347 *not* required) to provide an implementation of the pthreads thread-specific data interface  
 6348 based on the language feature. For example, assume that a compiler for a C-like language

6349 supports a *private* storage class that provides thread-specific storage. Something similar to  
 6350 the following macros might be used to effect a compatible implementation:

```
6351     #define pthread_key_t                private void *
6352     #define pthread_key_create(key)      /* no-op */
6353     #define pthread_setspecific(key,value) (key)=(value)
6354     #define pthread_getspecific(key)     (key)
```

6355 **Note:** For the sake of clarity, this example ignores destructor functions. A correct implementation  
 6356 would have to support them.

## 6357 Barriers

### 6358 • Background

6359 Barriers are typically used in parallel DO/FOR loops to ensure that all threads have reached  
 6360 a particular stage in a parallel computation before allowing any to proceed to the next stage.  
 6361 Highly efficient implementation is possible on machines which support a “Fetch and Add”  
 6362 operation as described in the referenced Almasi and Gottlieb (1989).

6363 The use of return value PTHREAD\_BARRIER\_SERIAL\_THREAD is shown in the following  
 6364 example:

```
6365     if ( (status=pthread_barrier_wait(&barrier)) ==
6366         PTHREAD_BARRIER_SERIAL_THREAD) {
6367         ...serial section
6368     }
6369         else if (status != 0) {
6370         ...error processing
6371     }
6372     status=pthread_barrier_wait(&barrier);
6373     ...
```

6374 This behavior allows a serial section of code to be executed by one thread as soon as all  
 6375 threads reach the first barrier. The second barrier prevents the other threads from proceeding  
 6376 until the serial section being executed by the one thread has completed.

6377 Although barriers can be implemented with mutexes and condition variables, the referenced  
 6378 Almasi and Gottlieb (1989) provides ample illustration that such implementations are  
 6379 significantly less efficient than is possible. While the relative efficiency of barriers may well  
 6380 vary by implementation, it is important that they be recognized in the IEEE Std 1003.1-200x  
 6381 to facilitate application portability while providing the necessary freedom to implementors.

### 6382 • Lack of Timeout Feature

6383 Alternate versions of most blocking routines have been provided to support watchdog  
 6384 timeouts. No alternate interface of this sort has been provided for barrier waits for the  
 6385 following reasons:

- 6386 • Multiple threads may use different timeout values, some of which may be indefinite. It is  
 6387 not clear which threads should break through the barrier with a timeout error if and when  
 6388 these timeouts expire.
- 6389 • The barrier may become unusable once a thread breaks out of a *pthread\_barrier\_wait()*  
 6390 with a timeout error. There is, in general, no way to guarantee the consistency of a  
 6391 barrier’s internal data structures once a thread has timed out of a *pthread\_barrier\_wait()*.  
 6392 Even the inclusion of a special barrier reinitialization function would not help much since  
 6393 it is not clear how this function would affect the behavior of threads that reach the barrier

6394                   between the original timeout and the call to the reinitialization function.

6395                   **Spin Locks**

6396                   • Background

6397                   Spin locks represent an extremely low-level synchronization mechanism suitable primarily  
6398                   for use on shared memory multi-processors. It is typically an atomically modified Boolean  
6399                   value that is set to one when the lock is held and to zero when the lock is freed.

6400                   When a caller requests a spin lock that is already held, it typically spins in a loop testing  
6401                   whether the lock has become available. Such spinning wastes processor cycles so the lock  
6402                   should only be held for short durations and not across sleep/block operations. Callers should  
6403                   unlock spin locks before calling sleep operations.

6404                   Spin locks are available on a variety of systems. The functions included in  
6405                   IEEE Std 1003.1-200x are an attempt to standardize that existing practice.

6406                   • Lack of Timeout Feature

6407                   Alternate versions of most blocking routines have been provided to support watchdog  
6408                   timeouts. No alternate interface of this sort has been provided for spin locks for the following  
6409                   reasons:

6410                   • It is impossible to determine appropriate timeout intervals for spin locks in a portable  
6411                   manner. The amount of time one can expect to spend spin-waiting is inversely  
6412                   proportional to the degree of parallelism provided by the system.

6413                   It can vary from a few cycles when each competing thread is running on its own  
6414                   processor, to an indefinite amount of time when all threads are multiplexed on a single  
6415                   processor (which is why spin locking is not advisable on uniprocessors).

6416                   • When used properly, the amount of time the calling thread spends waiting on a spin lock  
6417                   should be considerably less than the time required to set up a corresponding watchdog  
6418                   timer. Since the primary purpose of spin locks is to provide a low-overhead  
6419                   synchronization mechanism for multi-processors, the overhead of a timeout mechanism  
6420                   was deemed unacceptable.

6421                   It was also suggested that an additional *count* argument be provided (on the  
6422                   *pthread\_spin\_lock()* call) in *lieu* of a true timeout so that a spin lock call could fail gracefully if  
6423                   it was unable to apply the lock after *count* attempts. This idea was rejected because it is not  
6424                   existing practice. Furthermore, the same effect can be obtained with *pthread\_spin\_trylock()*,  
6425                   as illustrated below:

```

6426         int n = MAX_SPIN;
6427         while ( --n >= 0 )
6428         {
6429             if ( !pthread_spin_try_lock(...) )
6430                 break;
6431         }
6432         if ( n >= 0 )
6433         {
6434             /* Successfully acquired the lock */
6435         }
6436         else
6437         {
6438             /* Unable to acquire the lock */
6439         }

```

6440 • *process-shared* Attribute

6441 The initialization functions associated with most POSIX synchronization objects (for  
6442 example, mutexes, barriers, and read-write locks) take an attributes object with a *process-*  
6443 *shared* attribute that specifies whether or not the object is to be shared across processes. In the  
6444 draft corresponding to the first balloting round, two separate initialization functions are  
6445 provided for spin locks, however: one for spin locks that were to be shared across processes  
6446 (*spin\_init()*), and one for locks that were only used by multiple threads within a single  
6447 process (*pthread\_spin\_init()*). This was done so as to keep the overhead associated with spin  
6448 waiting to an absolute minimum. However, the balloting group requested that, since the  
6449 overhead associated to a bit check was small, spin locks should be consistent with the rest of  
6450 the synchronization primitives, and thus the *process-shared* attribute was introduced for spin  
6451 locks.

6452 • Spin Locks *versus* Mutexes

6453 It has been suggested that mutexes are an adequate synchronization mechanism and spin  
6454 locks are not necessary. Locking mechanisms typically must trade off the processor resources  
6455 consumed while setting up to block the thread and the processor resources consumed by the  
6456 thread while it is blocked. Spin locks require very little resources to set up the blocking of a  
6457 thread. Existing practice is to simply loop, repeating the atomic locking operation until the  
6458 lock is available. While the resources consumed to set up blocking of the thread are low, the  
6459 thread continues to consume processor resources while it is waiting.

6460 On the other hand, mutexes may be implemented such that the processor resources  
6461 consumed to block the thread are large relative to a spin lock. After detecting that the mutex  
6462 lock is not available, the thread must alter its scheduling state, add itself to a set of waiting  
6463 threads, and, when the lock becomes available again, undo all of this before taking over  
6464 ownership of the mutex. However, while a thread is blocked by a mutex, no processor  
6465 resources are consumed.

6466 Therefore, spin locks and mutexes may be implemented to have different characteristics.  
6467 Spin locks may have lower overall overhead for very short-term blocking, and mutexes may  
6468 have lower overall overhead when a thread will be blocked for longer periods of time. The  
6469 presence of both interfaces allows implementations with these two different characteristics,  
6470 both of which may be useful to a particular application.

6471 It has also been suggested that applications can build their own spin locks from the  
6472 *pthread\_mutex\_trylock()* function:

```
6473     while (pthread_mutex_trylock(&mutex));
```

6474 The apparent simplicity of this construct is somewhat deceiving, however. While the actual  
6475 wait is quite efficient, various guarantees on the integrity of mutex objects (for example,  
6476 priority inheritance rules) may add overhead to the successful path of the trylock operation  
6477 that is not required of spin locks. One could, of course, add an attribute to the mutex to  
6478 bypass such overhead, but the very act of finding and testing this attribute represents more  
6479 overhead than is found in the typical spin lock.

6480 The need to hold spin lock overhead to an absolute minimum also makes it impossible to  
6481 provide guarantees against starvation similar to those provided for mutexes or read-write  
6482 locks. The overhead required to implement such guarantees (for example, disabling  
6483 preemption before spinning) may well exceed the overhead of the spin wait itself by many  
6484 orders of magnitude. If a “safe” spin wait seems desirable, it can always be provided (albeit  
6485 at some performance cost) via appropriate mutex attributes.

### 6486 XSI Supported Functions

6487 On XSI-conformant systems, the following symbolic constants are always defined:

```
6488     _POSIX_READER_WRITER_LOCKS
6489     _POSIX_THREAD_ATTR_STACKADDR
6490     _POSIX_THREAD_ATTR_STACKSIZE
6491     _POSIX_THREAD_PROCESS_SHARED
6492     _POSIX_THREADS
```

6493 Therefore, the following threads functions are always supported:

6494	<i>pthread_atfork()</i>	<i>pthread_key_create()</i>
6495	<i>pthread_attr_destroy()</i>	<i>pthread_key_delete()</i>
6496	<i>pthread_attr_getdetachstate()</i>	<i>pthread_kill()</i>
6497	<i>pthread_attr_getguardsize()</i>	<i>pthread_mutex_destroy()</i>
6498	<i>pthread_attr_getschedparam()</i>	<i>pthread_mutex_init()</i>
6499	<i>pthread_attr_getstack()</i>	<i>pthread_mutex_lock()</i>
6500	<i>pthread_attr_getstackaddr()</i>	<i>pthread_mutex_trylock()</i>
6501	<i>pthread_attr_getstacksize()</i>	<i>pthread_mutex_unlock()</i>
6502	<i>pthread_attr_init()</i>	<i>pthread_mutexattr_destroy()</i>
6503	<i>pthread_attr_setdetachstate()</i>	<i>pthread_mutexattr_getpshared()</i>
6504	<i>pthread_attr_setguardsize()</i>	<i>pthread_mutexattr_gettype()</i>
6505	<i>pthread_attr_setschedparam()</i>	<i>pthread_mutexattr_init()</i>
6506	<i>pthread_attr_setstack()</i>	<i>pthread_mutexattr_setpshared()</i>
6507	<i>pthread_attr_setstackaddr()</i>	<i>pthread_mutexattr_settype()</i>
6508	<i>pthread_attr_setstacksize()</i>	<i>pthread_once()</i>
6509	<i>pthread_cancel()</i>	<i>pthread_rwlock_destroy()</i>
6510	<i>pthread_cleanup_pop()</i>	<i>pthread_rwlock_init()</i>
6511	<i>pthread_cleanup_push()</i>	<i>pthread_rwlock_rdlock()</i>
6512	<i>pthread_cond_broadcast()</i>	<i>pthread_rwlock_tryrdlock()</i>
6513	<i>pthread_cond_destroy()</i>	<i>pthread_rwlock_trywrlock()</i>
6514	<i>pthread_cond_init()</i>	<i>pthread_rwlock_unlock()</i>
6515	<i>pthread_cond_signal()</i>	<i>pthread_rwlock_wrlock()</i>
6516	<i>pthread_cond_timedwait()</i>	<i>pthread_rwlockattr_destroy()</i>
6517	<i>pthread_cond_wait()</i>	<i>pthread_rwlockattr_getpshared()</i>
6518	<i>pthread_condattr_destroy()</i>	<i>pthread_rwlockattr_init()</i>

6519	<i>pthread_condattr_getpshared()</i>	<i>pthread_rwlockattr_setpshared()</i>
6520	<i>pthread_condattr_init()</i>	<i>pthread_self()</i>
6521	<i>pthread_condattr_setpshared()</i>	<i>pthread_setcancelstate()</i>
6522	<i>pthread_create()</i>	<i>pthread_setcanceltype()</i>
6523	<i>pthread_detach()</i>	<i>pthread_setconcurrency()</i>
6524	<i>pthread_equal()</i>	<i>pthread_setspecific()</i>
6525	<i>pthread_exit()</i>	<i>pthread_sigmask()</i>
6526	<i>pthread_getconcurrency()</i>	<i>pthread_testcancel()</i>
6527	<i>pthread_getspecific()</i>	<i>sigwait()</i>
6528	<i>pthread_join()</i>	

6529 On XSI-conformant systems, the symbolic constant `_POSIX_THREAD_SAFE_FUNCTIONS` is  
6530 always defined. Therefore, the following functions are always supported:

6531	<i>asctime_r()</i>	<i>getpwuid_r()</i>
6532	<i>ctime_r()</i>	<i>gmtime_r()</i>
6533	<i>flockfile()</i>	<i>localtime_r()</i>
6534	<i>ftrylockfile()</i>	<i>putc_unlocked()</i>
6535	<i>funlockfile()</i>	<i>putchar_unlocked()</i>
6536	<i>getc_unlocked()</i>	<i>rand_r()</i>
6537	<i>getchar_unlocked()</i>	<i>readdir_r()</i>
6538	<i>getgrgid_r()</i>	<i>strerror_r()</i>
6539	<i>getgrnam_r()</i>	<i>strtok_r()</i>
6540	<i>getpwnam_r()</i>	

6541 The following threads functions are only supported on XSI-conformant systems if the Realtime  
6542 Threads Option Group is supported :

6543	<i>pthread_attr_getinheritsched()</i>	<i>pthread_mutex_getprioceiling()</i>
6544	<i>pthread_attr_getschedpolicy()</i>	<i>pthread_mutex_setprioceiling()</i>
6545	<i>pthread_attr_getscope()</i>	<i>pthread_mutexattr_getprioceiling()</i>
6546	<i>pthread_attr_setinheritsched()</i>	<i>pthread_mutexattr_getprotocol()</i>
6547	<i>pthread_attr_setschedpolicy()</i>	<i>pthread_mutexattr_setprioceiling()</i>
6548	<i>pthread_attr_setscope()</i>	<i>pthread_mutexattr_setprotocol()</i>
6549	<i>pthread_getschedparam()</i>	<i>pthread_setschedparam()</i>

### 6550 XSI Threads Extensions

6551 The following XSI extensions to POSIX.1c are now supported in IEEE Std 1003.1-200x as part of  
6552 the alignment with the Single UNIX Specification:

- 6553 • Extended mutex attribute types
- 6554 • Read-write locks and attributes (also introduced by IEEE Std 1003.1j-2000 amendment)
- 6555 • Thread concurrency level
- 6556 • Thread stack guard size
- 6557 • Parallel I/O

6558 A total of 19 new functions were added.

6559 These extensions carefully follow the threads programming model specified in POSIX.1c. As  
6560 with POSIX.1c, all the new functions return zero if successful; otherwise, an error number is

6561 returned to indicate the error.

6562 The concept of attribute objects was introduced in POSIX.1c to allow implementations to extend  
6563 IEEE Std 1003.1-200x without changing the existing interfaces. Attribute objects were defined for  
6564 threads, mutexes, and condition variables. Attributes objects are defined as implementation-  
6565 defined opaque types to aid extensibility, and functions are defined to allow attributes to be set  
6566 or retrieved. This model has been followed when adding the new type attribute of  
6567 **pthread\_mutexattr\_t** or the new read-write lock attributes object **pthread\_rwlockattr\_t**.

6568 • Extended Mutex Attributes

6569 POSIX.1c defines a mutex attributes object as an implementation-defined opaque object of  
6570 type **pthread\_mutexattr\_t**, and specifies a number of attributes which this object must have  
6571 and a number of functions which manipulate these attributes. These attributes include  
6572 *detachstate*, *inheritsched*, *schedparm*, *schedpolicy*, *contentionscope*, *stackaddr*, and *stacksize*.

6573 The System Interfaces volume of IEEE Std 1003.1-200x specifies another mutex attribute  
6574 called *type*. The *type* attribute allows applications to specify the behavior of mutex locking  
6575 operations in situations where the POSIX.1c behavior is undefined. The OSF DCE threads  
6576 implementation, based on Draft 4 of POSIX.1c, specified a similar attribute. Note that the  
6577 names of the attributes have changed somewhat from the OSF DCE threads implementation.

6578 The System Interfaces volume of IEEE Std 1003.1-200x also extends the specification of the  
6579 following POSIX.1c functions which manipulate mutexes:

6580 *pthread\_mutex\_lock()*  
6581 *pthread\_mutex\_trylock()*  
6582 *pthread\_mutex\_unlock()*

6583 to take account of the new mutex attribute type and to specify behavior which was declared  
6584 as undefined in POSIX.1c. How a calling thread acquires or releases a mutex now depends  
6585 upon the mutex *type* attribute.

6586 The *type* attribute can have the following values:

6587 PTHREAD\_MUTEX\_NORMAL  
6588 Basic mutex with no specific error checking built in. Does not report a deadlock error.

6589 PTHREAD\_MUTEX\_RECURSIVE  
6590 Allows any thread to recursively lock a mutex. The mutex must be unlocked an equal  
6591 number of times to release the mutex.

6592 PTHREAD\_MUTEX\_ERRORCHECK  
6593 Detects and reports simple usage errors; that is, an attempt to unlock a mutex that is not  
6594 locked by the calling thread or that is not locked at all, or an attempt to relock a mutex  
6595 the thread already owns.

6596 PTHREAD\_MUTEX\_DEFAULT  
6597 The default mutex type. May be mapped to any of the above mutex types or may be an  
6598 implementation-defined type.

6599 *Normal* mutexes do not detect deadlock conditions; for example, a thread will hang if it tries  
6600 to relock a normal mutex that it already owns. Attempting to unlock a mutex locked by  
6601 another thread, or unlocking an unlocked mutex, results in undefined behavior. Normal  
6602 mutexes will usually be the fastest type of mutex available on a platform but provide the  
6603 least error checking.

6604 *Recursive* mutexes are useful for converting old code where it is difficult to establish clear  
6605 boundaries of synchronization. A thread can relock a recursive mutex without first unlocking

6606 it. The relocking deadlock which can occur with normal mutexes cannot occur with this type  
6607 of mutex. However, multiple locks of a recursive mutex require the same number of unlocks  
6608 to release the mutex before another thread can acquire the mutex. Furthermore, this type of  
6609 mutex maintains the concept of an owner. Thus, a thread attempting to unlock a recursive  
6610 mutex which another thread has locked returns with an error. A thread attempting to unlock  
6611 a recursive mutex that is not locked shall return with an error. Never use a recursive mutex  
6612 with condition variables because the implicit unlock performed by `pthread_cond_wait()` or  
6613 `pthread_cond_timedwait()` will not actually release the mutex if it had been locked multiple  
6614 times.

6615 *Errorcheck* mutexes provide error checking and are useful primarily as a debugging aid. A  
6616 thread attempting to relock an errorcheck mutex without first unlocking it returns with an  
6617 error. Again, this type of mutex maintains the concept of an owner. Thus, a thread  
6618 attempting to unlock an errorcheck mutex which another thread has locked returns with an  
6619 error. A thread attempting to unlock an errorcheck mutex that is not locked also returns with  
6620 an error. It should be noted that errorcheck mutexes will almost always be much slower than  
6621 normal mutexes due to the extra state checks performed.

6622 The *default* mutex type provides implementation-defined error checking. The default mutex  
6623 may be mapped to one of the other defined types or may be something entirely different.  
6624 This enables each vendor to provide the mutex semantics which the vendor feels will be  
6625 most useful to their target users. Most vendors will probably choose to make normal  
6626 mutexes the default so as to give applications the benefit of the fastest type of mutexes  
6627 available on their platform. Check your implementation's documentation.

6628 An application developer can use any of the mutex types almost interchangeably as long as  
6629 the application does not depend upon the implementation detecting (or failing to detect) any  
6630 particular errors. Note that a recursive mutex can be used with condition variable waits as  
6631 long as the application never recursively locks the mutex.

6632 Two functions are provided for manipulating the *type* attribute of a mutex attributes object.  
6633 This attribute is set or returned in the *type* parameter of these functions. The  
6634 `pthread_mutexattr_settype()` function is used to set a specific type value while  
6635 `pthread_mutexattr_gettype()` is used to return the type of the mutex. Setting the *type* attribute  
6636 of a mutex attributes object affects only mutexes initialized using that mutex attributes  
6637 object. Changing the *type* attribute does not affect mutexes previously initialized using that  
6638 mutex attributes object.

#### 6639 • Read-Write Locks and Attributes

6640 The read-write locks introduced have been harmonized with those in IEEE Std 1003.1j-2000;  
6641 see also Section B.2.9.6 (on page 3464).

6642 Read-write locks (also known as reader-writer locks) allow a thread to exclusively lock some  
6643 shared data while updating that data, or allow any number of threads to have simultaneous  
6644 read-only access to the data.

6645 Unlike a mutex, a read-write lock distinguishes between reading data and writing data. A  
6646 mutex excludes all other threads. A read-write lock allows other threads access to the data,  
6647 providing no thread is modifying the data. Thus, a read-write lock is less primitive than  
6648 either a mutex-condition variable pair or a semaphore.

6649 Application developers should consider using a read-write lock rather than a mutex to  
6650 protect data that is frequently referenced but seldom modified. Most threads (readers) will be  
6651 able to read the data without waiting and will only have to block when some other thread (a  
6652 writer) is in the process of modifying the data. Conversely a thread that wants to change the  
6653 data is forced to wait until there are no readers. This type of lock is often used to facilitate

6654 parallel access to data on multi-processor platforms or to avoid context switches on single  
6655 processor platforms where multiple threads access the same data.

6656 If a read-write lock becomes unlocked and there are multiple threads waiting to acquire the  
6657 write lock, the implementation's scheduling policy determines which thread shall acquire the  
6658 read-write lock for writing. If there are multiple threads blocked on a read-write lock for both  
6659 read locks and write locks, it is unspecified whether the readers or a writer acquire the lock  
6660 first. However, for performance reasons, implementations often favor writers over readers to  
6661 avoid potential writer starvation.

6662 A read-write lock object is an implementation-defined opaque object of type  
6663 **pthread\_rwlock\_t** as defined in `<pthread.h>`. There are two different sorts of locks  
6664 associated with a read-write lock: a *read lock* and a *write lock*.

6665 The `pthread_rwlockattr_init()` function initializes a read-write lock attributes object with the  
6666 default value for all the attributes defined in the implementation. After a read-write lock  
6667 attributes object has been used to initialize one or more read-write locks, changes to the  
6668 read-write lock attributes object, including destruction, do not affect previously initialized  
6669 read-write locks.

6670 Implementations must provide at least the read-write lock attribute *process-shared*. This  
6671 attribute can have the following values:

6672 **PTHREAD\_PROCESS\_SHARED**  
6673 Any thread of any process that has access to the memory where the read-write lock  
6674 resides can manipulate the read-write lock.

6675 **PTHREAD\_PROCESS\_PRIVATE**  
6676 Only threads created within the same process as the thread that initialized the read-  
6677 write lock can manipulate the read-write lock. This is the default value.

6678 The `pthread_rwlockattr_setpshared()` function is used to set the *process-shared* attribute of an  
6679 initialized read-write lock attributes object while the function `pthread_rwlockattr_getpshared()`  
6680 obtains the current value of the *process-shared* attribute.

6681 A read-write lock attributes object is destroyed using the `pthread_rwlockattr_destroy()`  
6682 function. The effect of subsequent use of the read-write lock attributes object is undefined.

6683 A thread creates a read-write lock using the `pthread_rwlock_init()` function. The attributes of  
6684 the read-write lock can be specified by the application developer; otherwise, the default  
6685 implementation-defined read-write lock attributes are used if the pointer to the read-write  
6686 lock attributes object is NULL. In cases where the default attributes are appropriate, the  
6687 **PTHREAD\_RWLOCK\_INITIALIZER** macro can be used to initialize statically allocated  
6688 read-write locks.

6689 A thread which wants to apply a read lock to the read-write lock can use either  
6690 `pthread_rwlock_rdlock()` or `pthread_rwlock_tryrdlock()`. If `pthread_rwlock_rdlock()` is used, the  
6691 thread acquires a read lock if a writer does not hold the write lock and there are no writers  
6692 blocked on the write lock. If a read lock is not acquired, the calling thread blocks until it can  
6693 acquire a lock. However, if `pthread_rwlock_tryrdlock()` is used, the function returns  
6694 immediately with the error [EBUSY] if any thread holds a write lock or there are blocked  
6695 writers waiting for the write lock.

6696 A thread which wants to apply a write lock to the read-write lock can use either of two  
6697 functions: `pthread_rwlock_wrlock()` or `pthread_rwlock_trywrlock()`. If `pthread_rwlock_wrlock()`  
6698 is used, the thread acquires the write lock if no other reader or writer threads hold the read-  
6699 write lock. If the write lock is not acquired, the thread blocks until it can acquire the write  
6700 lock. However, if `pthread_rwlock_trywrlock()` is used, the function returns immediately with

6701 the error [EBUSY] if any thread is holding either a read or a write lock.

6702 The `pthread_rwlock_unlock()` function is used to unlock a read-write lock object held by the  
6703 calling thread. Results are undefined if the read-write lock is not held by the calling thread. If  
6704 there are other read locks currently held on the read-write lock object, the read-write lock  
6705 object shall remain in the read locked state but without the current thread as one of its  
6706 owners. If this function releases the last read lock for this read-write lock object, the read-  
6707 write lock object shall be put in the unlocked read state. If this function is called to release a  
6708 write lock for this read-write lock object, the read-write lock object shall be put in the  
6709 unlocked state.

6710 • Thread Concurrency Level

6711 On threads implementations that multiplex user threads onto a smaller set of kernel  
6712 execution entities, the system attempts to create a reasonable number of kernel execution  
6713 entities for the application upon application startup.

6714 On some implementations, these kernel entities are retained by user threads that block in the  
6715 kernel. Other implementations do not *timeslice* user threads so that multiple compute-bound  
6716 user threads can share a kernel thread. On such implementations, some applications may use  
6717 up all the available kernel execution entities before its user-space threads are used up. The  
6718 process may be left with user threads capable of doing work for the application but with no  
6719 way to schedule them.

6720 The `pthread_setconcurrency()` function enables an application to request more kernel entities;  
6721 that is, specify a desired concurrency level. However, this function merely provides a hint to  
6722 the implementation. The implementation is free to ignore this request or to provide some  
6723 other number of kernel entities. If an implementation does not multiplex user threads onto a  
6724 smaller number of kernel execution entities, the `pthread_setconcurrency()` function has no  
6725 effect.

6726 The `pthread_setconcurrency()` function may also have an effect on implementations where the  
6727 kernel mode and user mode schedulers cooperate to ensure that ready user threads are not  
6728 prevented from running by other threads blocked in the kernel.

6729 The `pthread_getconcurrency()` function always returns the value set by a previous call to  
6730 `pthread_setconcurrency()`. However, if `pthread_setconcurrency()` was not previously called, this  
6731 function shall return zero to indicate that the threads implementation is maintaining the  
6732 concurrency level.

6733 • Thread Stack Guard Size

6734 DCE threads introduced the concept of a *thread stack guard size*. Most thread  
6735 implementations add a region of protected memory to a thread's stack, commonly known as  
6736 a *guard region*, as a safety measure to prevent stack pointer overflow in one thread from  
6737 corrupting the contents of another thread's stack. The default size of the guard regions  
6738 attribute is {PAGESIZE} bytes and is implementation-defined.

6739 Some application developers may wish to change the stack guard size. When an application  
6740 creates a large number of threads, the extra page allocated for each stack may strain system  
6741 resources. In addition to the extra page of memory, the kernel's memory manager has to keep  
6742 track of the different protections on adjoining pages. When this is a problem, the application  
6743 developer may request a guard size of 0 bytes to conserve system resources by eliminating  
6744 stack overflow protection.

6745 Conversely an application that allocates large data structures such as arrays on the stack may  
6746 wish to increase the default guard size in order to detect stack overflow. If a thread allocates  
6747 two pages for a data array, a single guard page provides little protection against thread stack

6748 overflows since the thread can corrupt adjoining memory beyond the guard page.

6749 The System Interfaces volume of IEEE Std 1003.1-200x defines a new attribute of a thread  
6750 attributes object; that is, the *guardsize* attribute which allows applications to specify the size  
6751 of the guard region of a thread's stack.

6752 Two functions are provided for manipulating a thread's stack guard size. The  
6753 *pthread\_attr\_setguardsize()* function sets the thread *guardsize* attribute, and the  
6754 *pthread\_attr\_getguardsize()* function retrieves the current value.

6755 An implementation may round up the requested guard size to a multiple of the configurable  
6756 system variable {PAGESIZE}. In this case, *pthread\_attr\_getguardsize()* returns the guard size  
6757 specified by the previous *pthread\_attr\_setguardsize()* function call and not the rounded up  
6758 value.

6759 If an application is managing its own thread stacks using the *stackaddr* attribute, the *guardsize*  
6760 attribute is ignored and no stack overflow protection is provided. In this case, it is the  
6761 responsibility of the application to manage stack overflow along with stack allocation.

6762 • Parallel I/O

6763 Suppose two or more threads independently issue read requests on the same file. To read  
6764 specific data from a file, a thread must first call *lseek()* to seek to the proper offset in the file,  
6765 and then call *read()* to retrieve the required data. If more than one thread does this at the  
6766 same time, the first thread may complete its seek call, but before it gets a chance to issue its  
6767 read call a second thread may complete its seek call, resulting in the first thread accessing  
6768 incorrect data when it issues its read call. One workaround is to lock the file descriptor while  
6769 seeking and reading or writing, but this reduces parallelism and adds overhead.

6770 Instead, the System Interfaces volume of IEEE Std 1003.1-200x provides two functions to  
6771 make seek/read and seek/write operations atomic. The file descriptor's current offset is  
6772 unchanged, thus allowing multiple read and write operations to proceed in parallel. This  
6773 improves the I/O performance of threaded applications. The *pread()* function is used to do  
6774 an atomic read of data from a file into a buffer. Conversely, the *pwrite()* function does an  
6775 atomic write of data from a buffer to a file.

6776 **B.2.9.1 Thread-Safety**

6777 All functions required by IEEE Std 1003.1-200x need to be thread-safe. Implementations have to  
6778 provide internal synchronization when necessary in order to achieve this goal. In certain  
6779 cases—for example, most floating-point implementations—context switch code may have to  
6780 manage the writable shared state.

6781 While a read from a pipe of {PIPE\_MAX}\*2 bytes may not generate a single atomic and thread- |  
6782 safe stream of bytes, it should generate “several” (individually atomic) thread-safe streams of |  
6783 bytes. Similarly, while reading from a terminal device may not generate a single atomic and |  
6784 thread-safe stream of bytes, it should generate some finite number of (individually atomic) and |  
6785 thread-safe streams of bytes. That is, concurrent calls to read for a pipe, FIFO, or terminal device |  
6786 are not allowed to result in corrupting the stream of bytes or other internal data. However, |  
6787 *read()*, in these cases, is not required to return a single contiguous and atomic stream of bytes. |

6788 It is not required that all functions provided by IEEE Std 1003.1-200x be either async-cancel-safe |  
6789 or async-signal-safe.

6790 As it turns out, some functions are inherently not thread-safe; that is, their interface  
6791 specifications preclude reentrancy. For example, some functions (such as *asctime()*) return a  
6792 pointer to a result stored in memory space allocated by the function on a per-process basis. Such  
6793 a function is not thread-safe, because its result can be overwritten by successive invocations.

6794 Other functions, while not inherently non-thread-safe, may be implemented in ways that lead to  
6795 them not being thread-safe. For example, some functions (such as *rand()*) store state information  
6796 (such as a seed value, which survives multiple function invocations) in memory space allocated  
6797 by the function on a per-process basis. The implementation of such a function is not thread-safe  
6798 if the implementation fails to synchronize invocations of the function and thus fails to protect  
6799 the state information. The problem is that when the state information is not protected,  
6800 concurrent invocations can interfere with one another (for example, see the same seed value).

#### 6801 *Thread-Safety and Locking of Existing Functions*

6802 Originally, POSIX.1 was not designed to work in a multi-threaded environment, and some  
6803 implementations of some existing functions will not work properly when executed concurrently.  
6804 To provide routines that will work correctly in an environment with threads (“thread-safe”), two  
6805 problems need to be solved:

- 6806 1. Routines that maintain or return pointers to static areas internal to the routine (which may  
6807 now be shared) need to be modified. The routines *ttyname()* and *localtime()* are examples.
- 6808 2. Routines that access data space shared by more than one thread need to be modified. The  
6809 *malloc()* function and the *stdio* family routines are examples.

6810 There are a variety of constraints on these changes. The first is compatibility with the existing  
6811 versions of these functions—non-thread-safe functions will continue to be in use for some time,  
6812 as the original interfaces are used by existing code. Another is that the new thread-safe versions  
6813 of these functions represent as small a change as possible over the familiar interfaces provided  
6814 by the existing non-thread-safe versions. The new interfaces should be independent of any  
6815 particular threads implementation. In particular, they should be thread-safe without depending  
6816 on explicit thread-specific memory. Finally, there should be minimal performance penalty due to  
6817 the changes made to the functions.

6818 It is intended that the list of functions from POSIX.1 that cannot be made thread-safe and for  
6819 which corrected versions are provided be complete.

#### 6820 *Thread-Safety and Locking Solutions*

6821 Many of the POSIX.1 functions were thread-safe and did not change at all. However, some  
6822 functions (for example, the math functions typically found in **libm**) are not thread-safe because  
6823 of writable shared global state. For instance, in IEEE Std 754-1985 floating-point  
6824 implementations, the computation modes and flags are global and shared.

6825 Some functions are not thread-safe because a particular implementation is not reentrant,  
6826 typically because of a non-essential use of static storage. These require only a new  
6827 implementation.

6828 Thread-safe libraries are useful in a wide range of parallel (and asynchronous) programming  
6829 environments, not just within pthreads. In order to be used outside the context of pthreads,  
6830 however, such libraries still have to use some synchronization method. These could either be  
6831 independent of the pthread synchronization operations, or they could be a subset of the pthread  
6832 interfaces. Either method results in thread-safe library implementations that can be used without  
6833 the rest of pthreads.

6834 Some functions, such as the *stdio* family interface and dynamic memory allocation functions  
6835 such as *malloc()*, are interdependent routines that share resources (for example, buffers) across  
6836 related calls. These require synchronization to work correctly, but they do not require any  
6837 change to their external (user-visible) interfaces.

6838 In some cases, such as *getc()* and *putc()*, adding synchronization is likely to create an  
6839 unacceptable performance impact. In this case, slower thread-safe synchronized functions are to

6840 be provided, but the original, faster (but unsafe) functions (which may be implemented as  
6841 macros) are retained under new names. Some additional special-purpose synchronization  
6842 facilities are necessary for these macros to be usable in multi-threaded programs. This also  
6843 requires changes in `<stdio.h>`.

6844 The other common reason that functions are unsafe is that they return a pointer to static storage,  
6845 making the functions non-thread-safe. This has to be changed, and there are three natural  
6846 choices:

6847 1. Return a pointer to thread-specific storage

6848 This could incur a severe performance penalty on those architectures with a costly  
6849 implementation of the thread-specific data interface.

6850 A variation on this technique is to use `malloc()` to allocate storage for the function output  
6851 and return a pointer to this storage. This technique may also have an undesirable  
6852 performance impact, however, and a simplistic implementation requires that the user  
6853 program explicitly free the storage object when it is no longer needed. This technique is  
6854 used by some existing POSIX.1 functions. With careful implementation for infrequently  
6855 used functions, there may be little or no performance or storage penalty, and the  
6856 maintenance of already-standardized interfaces is a significant benefit.

6857 2. Return the actual value computed by the function

6858 This technique can only be used with functions that return pointers to structures—routines  
6859 that return character strings would have to wrap their output in an enclosing structure in  
6860 order to return the output on the stack. There is also a negative performance impact  
6861 inherent in this solution in that the output value has to be copied twice before it can be  
6862 used by the calling function: once from the called routine's local buffers to the top of the  
6863 stack, then from the top of the stack to the assignment target. Finally, many older  
6864 compilers cannot support this technique due to a historical tendency to use internal static  
6865 buffers to deliver the results of structure-valued functions.

6866 3. Have the caller pass the address of a buffer to contain the computed value

6867 The only disadvantage of this approach is that extra arguments have to be provided by the  
6868 calling program. It represents the most efficient solution to the problem, however, and,  
6869 unlike the `malloc()` technique, it is semantically clear.

6870 There are some routines (often groups of related routines) whose interfaces are inherently non-  
6871 thread-safe because they communicate across multiple function invocations by means of static  
6872 memory locations. The solution is to redesign the calls so that they are thread-safe, typically by  
6873 passing the needed data as extra parameters. Unfortunately, this may require major changes to  
6874 the interface as well.

6875 A floating-point implementation using IEEE Std 754-1985 is a case in point. A less problematic  
6876 example is the `rand48` family of pseudo-random number generators. The functions `getgrgid()`,  
6877 `getgrnam()`, `getpwnam()`, and `getpwuid()` are another such case.

6878 The problems with `errno` are discussed in **Alternative Solutions for Per-Thread `errno`** (on page  
6879 3382).

6880 Some functions can be thread-safe or not, depending on their arguments. These include the  
6881 `tmpnam()` and `ctermid()` functions. These functions have pointers to character strings as  
6882 arguments. If the pointers are not NULL, the functions store their results in the character string;  
6883 however, if the pointers are NULL, the functions store their results in an area that may be static  
6884 and thus subject to overwriting by successive calls. These should only be called by multi-thread  
6885 applications when their arguments are non-NULL.

6886 *Asynchronous Safety and Thread-Safety*

6887 A floating-point implementation has many modes that effect rounding and other aspects of  
6888 computation. Functions in some math library implementations may change the computation  
6889 modes for the duration of a function call. If such a function call is interrupted by a signal or  
6890 cancelation, the floating-point state is not required to be protected.

6891 There is a significant cost to make floating-point operations async-cancel-safe or async-signal-  
6892 safe; accordingly, neither form of async safety is required.

6893 *Functions Returning Pointers to Static Storage*

6894 For those functions that are not thread-safe because they return values in fixed size statically  
6895 allocated structures, alternate “\_r” forms are provided that pass a pointer to an explicit result  
6896 structure. Those that return pointers into library-allocated buffers have forms provided with  
6897 explicit buffer and length parameters.

6898 For functions that return pointers to library-allocated buffers, it makes sense to provide “\_r”  
6899 versions that allow the application control over allocation of the storage in which results are  
6900 returned. This allows the state used by these functions to be managed on an application-specific  
6901 basis, supporting per-thread, per-process, or other application-specific sharing relationships.

6902 Early proposals had provided “\_r” versions for functions that returned pointers to variable-size  
6903 buffers without providing a means for determining the required buffer size. This would have  
6904 made using such functions exceedingly clumsy, potentially requiring iteratively calling them  
6905 with increasingly larger guesses for the amount of storage required. Hence, *sysconf()* variables  
6906 have been provided for such functions that return the maximum required buffer size.

6907 Thus, the rule that has been followed by IEEE Std 1003.1-200x when adapting single-threaded  
6908 non-thread-safe functions is as follows: all functions returning pointers to library-allocated  
6909 storage should have “\_r” versions provided, allowing the application control over the storage  
6910 allocation. Those with variable-sized return values accept both a buffer address and a length  
6911 parameter. The *sysconf()* variables are provided to supply the appropriate buffer sizes when  
6912 required. Implementors are encouraged to apply the same rule when adapting their own existing  
6913 functions to a pthreads environment.

6914 *B.2.9.2 Thread IDs*

6915 Separate programs should communicate through well-defined interfaces and should not depend  
6916 on each other's implementation. For example, if a programmer decides to rewrite the *sort*  
6917 program using multiple threads, it should be easy to do this so that the interface to the *sort*  
6918 program does not change. Consider that if the user causes SIGINT to be generated while the *sort*  
6919 program is running, keeping the same interface means that the entire sort program is killed, not  
6920 just one of its threads. As another example, consider a realtime program that manages a reactor.  
6921 Such a program may wish to allow other programs to control the priority at which it watches the  
6922 control rods. One technique to accomplish this is to write the ID of the thread watching the  
6923 control rods into a file and allow other programs to change the priority of that thread as they see  
6924 fit. A simpler technique is to have the reactor process accept IPCs (Inter-Process Communication  
6925 messages) from other processes, telling it at a semantic level what priority the program should  
6926 assign to watching the control rods. This allows the programmer greater flexibility in the  
6927 implementation. For example, the programmer can change the implementation from having one  
6928 thread per rod to having one thread watching all of the rods without changing the interface.  
6929 Having threads live inside the process means that the implementation of a process is invisible to  
6930 outside processes (excepting debuggers and system management tools).

6931 Threads do not provide a protection boundary. Every thread model allows threads to share  
6932 memory with other threads and encourages this sharing to be widespread. This means that one

6933 thread can wipe out memory that is needed for the correct functioning of other threads that are  
 6934 sharing its memory. Consequently, providing each thread with its own user and/or group IDs  
 6935 would not provide a protection boundary between threads sharing memory.

### 6936 *B.2.9.3 Thread Mutexes*

6937 There is no additional rationale provided for this section.

### 6938 *B.2.9.4 Thread Scheduling*

#### 6939 • Scheduling Implementation Models

6940 The following scheduling implementation models are presented in terms of threads and  
 6941 “kernel entities”. This is to simplify exposition of the models, and it does not imply that an  
 6942 implementation actually has an identifiable “kernel entity”.

6943 A kernel entity is not defined beyond the fact that it has scheduling attributes that are used to  
 6944 resolve contention with other kernel entities for execution resources. A kernel entity may be  
 6945 thought of as an envelope that holds a thread or a separate kernel thread. It is not a  
 6946 conventional process, although it shares with the process the attribute that it has a single  
 6947 thread of control; it does not necessarily imply an address space, open files, and so on. It is  
 6948 better thought of as a primitive facility upon which conventional processes and threads may  
 6949 be constructed.

#### 6950 — System Thread Scheduling Model

6951 This model consists of one thread per kernel entity. The kernel entity is solely responsible  
 6952 for scheduling thread execution on one or more processors. This model schedules all  
 6953 threads against all other threads in the system using the scheduling attributes of the  
 6954 thread.

#### 6955 — Process Scheduling Model

6956 A generalized process scheduling model consists of two levels of scheduling. A threads  
 6957 library creates a pool of kernel entities, as required, and schedules threads to run on them  
 6958 using the scheduling attributes of the threads. Typically, the size of the pool is a function  
 6959 of the simultaneously runnable threads, not the total number of threads. The kernel then  
 6960 schedules the kernel entities onto processors according to their scheduling attributes,  
 6961 which are managed by the threads library. This set model potentially allows a wide range  
 6962 of mappings between threads and kernel entities.

#### 6963 • System and Process Scheduling Model Performance

6964 There are a number of important implications on the performance of applications using these  
 6965 scheduling models. The process scheduling model potentially provides lower overhead for  
 6966 making scheduling decisions, since there is no need to access kernel-level information or  
 6967 functions and the set of schedulable entities is smaller (only the threads within the process).

6968 On the other hand, since the kernel is also making scheduling decisions regarding the system  
 6969 resources under its control (for example, CPU(s), I/O devices, memory), decisions that do  
 6970 not take thread scheduling parameters into account can result in unspecified delays for  
 6971 realtime application threads, causing them to miss maximum response time limits. |

#### 6972 • Rate Monotonic Scheduling

6973 Rate monotonic scheduling was considered, but rejected for standardization in the context of  
 6974 pthreads. A sporadic server policy is included.

## 6975 • Scheduling Options

6976 In IEEE Std 1003.1-200x, the basic thread scheduling functions are defined under the Threads  
6977 option, so that they are required of all threads implementations. However, there are no  
6978 specific scheduling policies required by this option to allow for conforming thread  
6979 implementations that are not targeted to realtime applications.

6980 Specific standard scheduling policies are defined to be under the Thread Execution  
6981 Scheduling option, and they are specifically designed to support realtime applications by  
6982 providing predictable resource sharing sequences. The name of this option was chosen to  
6983 emphasize that this functionality is defined as appropriate for realtime applications that  
6984 require simple priority-based scheduling.

6985 It is recognized that these policies are not necessarily satisfactory for some multi-processor  
6986 implementations, and work is ongoing to address a wider range of scheduling behaviors. The  
6987 interfaces have been chosen to create abundant opportunity for future scheduling policies to  
6988 be implemented and standardized based on this interface. In order to standardize a new  
6989 scheduling policy, all that is required (from the standpoint of thread scheduling attributes) is  
6990 to define a new policy name, new members of the thread attributes object, and functions to  
6991 set these members when the scheduling policy is equal to the new value.

6992 **Scheduling Contention Scope**

6993 In order to accommodate the requirement for realtime response, each thread has a scheduling  
6994 contention scope attribute. Threads with a system scheduling contention scope have to be  
6995 scheduled with respect to all other threads in the system. These threads are usually bound to a  
6996 single kernel entity that reflects their scheduling attributes and are directly scheduled by the  
6997 kernel.

6998 Threads with a process scheduling contention scope need be scheduled only with respect to the  
6999 other threads in the process. These threads may be scheduled within the process onto a pool of  
7000 kernel entities. The implementation is also free to bind these threads directly to kernel entities  
7001 and let them be scheduled by the kernel. Process scheduling contention scope allows the  
7002 implementation the most flexibility and is the default if both contention scopes are supported  
7003 and none is specified.

7004 Thus, the choice by implementors to provide one or the other (or both) of these scheduling  
7005 models is driven by the need of their supported application domains for worst-case (that is,  
7006 realtime) response, or average-case (non-realtime) response.

7007 **Scheduling Allocation Domain**

7008 The SCHED\_FIFO and SCHED\_RR scheduling policies take on different characteristics on a  
7009 multi-processor. Other scheduling policies are also subject to changed behavior when executed  
7010 on a multi-processor. The concept of scheduling allocation domain determines the set of  
7011 processors on which the threads of an application may run. By considering the application's  
7012 processor scheduling allocation domain for its threads, scheduling policies can be defined in  
7013 terms of their behavior for varying processor scheduling allocation domain values. It is  
7014 conceivable that not all scheduling allocation domain sizes make sense for all scheduling  
7015 policies on all implementations. The concept of scheduling allocation domain, however, is a  
7016 useful tool for the description of multi-processor scheduling policies.

7017 The "process control" approach to scheduling obtains significant performance advantages from  
7018 dynamic scheduling allocation domain sizes when it is applicable.

7019 Non-Uniform Memory Access (NUMA) multi-processors may use a system scheduling structure  
7020 that involves reassignment of threads among scheduling allocation domains. In NUMA

7021 machines, a natural model of scheduling is to match scheduling allocation domains to clusters of  
7022 processors. Load balancing in such an environment requires changing the scheduling allocation  
7023 domain to which a thread is assigned.

#### 7024 **Scheduling Documentation**

7025 Implementation-provided scheduling policies need to be completely documented in order to be  
7026 useful. This documentation includes a description of the attributes required for the policy, the  
7027 scheduling interaction of threads running under this policy and all other supported policies, and  
7028 the effects of all possible values for processor scheduling allocation domain. Note that for the  
7029 implementor wishing to be minimally-compliant, it is (minimally) acceptable to define the  
7030 behavior as undefined.

#### 7031 **Scheduling Contention Scope Attribute**

7032 The scheduling contention scope defines how threads compete for resources. Within  
7033 IEEE Std 1003.1-200x, scheduling contention scope is used to describe only how threads are  
7034 scheduled in relation to one another in the system. That is, either they are scheduled against all  
7035 other threads in the system (“system scope”) or only against those threads in the process  
7036 (“process scope”). In fact, scheduling contention scope may apply to additional resources,  
7037 including virtual timers and profiling, which are not currently considered by  
7038 IEEE Std 1003.1-200x.

#### 7039 **Mixed Scopes**

7040 If only one scheduling contention scope is supported, the scheduling decision is straightforward.  
7041 To perform the processor scheduling decision in a mixed scope environment, it is necessary to  
7042 map the scheduling attributes of the thread with process-wide contention scope to the same  
7043 attribute space as the thread with system-wide contention scope.

7044 Since a conforming implementation has to support one and may support both scopes, it is useful  
7045 to discuss the effects of such choices with respect to example applications. If an implementation  
7046 supports both scopes, mixing scopes provides a means of better managing system-level (that is,  
7047 kernel-level) and library-level resources. In general, threads with system scope will require the  
7048 resources of a separate kernel entity in order to guarantee the scheduling semantics. On the  
7049 other hand, threads with process scope can share the resources of a kernel entity while  
7050 maintaining the scheduling semantics.

7051 The application is free to create threads with dedicated kernel resources, and other threads that  
7052 multiplex kernel resources. Consider the example of a window server. The server allocates two  
7053 threads per widget: one thread manages the widget user interface (including drawing), while the  
7054 other thread takes any required application action. This allows the widget to be “active” while  
7055 the application is computing. A screen image may be built from thousands of widgets. If each of  
7056 these threads had been created with system scope, then most of the kernel-level resources might  
7057 be wasted, since only a few widgets are active at any one time. In addition, mixed scope is  
7058 particularly useful in a window server where one thread with high priority and system scope  
7059 handles the mouse so that it tracks well. As another example, consider a database server. For  
7060 each of the hundreds or thousands of clients supported by a large server, an equivalent number  
7061 of threads will have to be created. If each of these threads were system, the consequences would  
7062 be the same as for the window server example above. However, the server could be constructed  
7063 so that actual retrieval of data is done by several dedicated threads. Dedicated threads that do  
7064 work for all clients frequently justify the added expense of system scope. If it were not  
7065 permissible to mix system and process threads in the same process, this type of solution would  
7066 not be possible.

**7067 Dynamic Thread Scheduling Parameters Access**

7068 In many time-constrained applications, there is no need to change the scheduling attributes  
7069 dynamically during thread or process execution, since the general use of these attributes is to  
7070 reflect directly the time constraints of the application. Since these time constraints are generally  
7071 imposed to meet higher-level system requirements, such as accuracy or availability, they  
7072 frequently should remain unchanged during application execution.

7073 However, there are important situations in which the scheduling attributes should be changed.  
7074 Generally, this will occur when external environmental conditions exist in which the time  
7075 constraints change. Consider, for example, a space vehicle major mode change, such as the  
7076 change from ascent to descent mode, or the change from the space environment to the  
7077 atmospheric environment. In such cases, the frequency with which many of the sensors or  
7078 acutators need to be read or written will change, which will necessitate a priority change. In  
7079 other cases, even the existence of a time constraint might be temporary, necessitating not just a  
7080 priority change, but also a policy change for ongoing threads or processes. For this reason, it is  
7081 critical that the interface should provide functions to change the scheduling parameters  
7082 dynamically, but, as with many of the other realtime functions, it is important that applications  
7083 use them properly to avoid the possibility of unnecessarily degrading performance.

7084 In providing functions for dynamically changing the scheduling behavior of threads, there were  
7085 two options: provide functions to get and set the individual scheduling parameters of threads, or  
7086 provide a single interface to get and set all the scheduling parameters for a given thread  
7087 simultaneously. Both approaches have merit. Access functions for individual parameters allow  
7088 simpler control of thread scheduling for simple thread scheduling parameters. However, a single  
7089 function for setting all the parameters for a given scheduling policy is required when first setting  
7090 that scheduling policy. Since the single all-encompassing functions are required, it was decided  
7091 to leave the interface as minimal as possible. Note that simpler functions (such as  
7092 *pthread\_setprio()* for threads running under the priority-based schedulers) can be easily defined  
7093 in terms of the all-encompassing functions.

7094 If the *pthread\_setschedparam()* function executes successfully, it will have set all of the scheduling  
7095 parameter values indicated in *param*; otherwise, none of the scheduling parameters will have  
7096 been modified. This is necessary to ensure that the scheduling of this and all other threads  
7097 continues to be consistent in the presence of an erroneous scheduling parameter.

7098 The [EPERM] error value is included in the list of possible *pthread\_setschedparam()* error returns  
7099 as a reflection of the fact that the ability to change scheduling parameters increases risks to the  
7100 implementation and application performance if the scheduling parameters are changed  
7101 improperly. For this reason, and based on some existing practice, it was felt that some  
7102 implementations would probably choose to define specific permissions for changing either a  
7103 thread's own or another thread's scheduling parameters. IEEE Std 1003.1-200x does not include  
7104 portable methods for setting or retrieving permissions, so any such use of permissions is  
7105 completely unspecified .

**7106 Mutex Initialization Scheduling Attributes**

7107 In a priority-driven environment, a direct use of traditional primitives like mutexes and  
7108 condition variables can lead to unbounded priority inversion, where a higher priority thread can  
7109 be blocked by a lower priority thread, or set of threads, for an unbounded duration of time. As a  
7110 result, it becomes impossible to guarantee thread deadlines. Priority inversion can be bounded  
7111 and minimized by the use of priority inheritance protocols. This allows thread deadlines to be  
7112 guaranteed even in the presence of synchronization requirements.

7113 Two useful but simple members of the family of priority inheritance protocols are the basic  
7114 priority inheritance protocol and the priority ceiling protocol emulation. Under the Basic Priority

7115 Inheritance protocol (governed by the Thread Priority Inheritance option), a thread that is  
7116 blocking higher priority threads executes at the priority of the highest priority thread that it  
7117 blocks. This simple mechanism allows priority inversion to be bounded by the duration of  
7118 critical sections and makes timing analysis possible.

7119 Under the Priority Ceiling Protocol Emulation protocol (governed by the Thread Priority  
7120 Protection option), each mutex has a priority ceiling, usually defined as the priority of the  
7121 highest priority thread that can lock the mutex. When a thread is executing inside critical  
7122 sections, its priority is unconditionally increased to the highest of the priority ceilings of all the  
7123 mutexes owned by the thread. This protocol has two very desirable properties in uni-processor  
7124 systems. First, a thread can be blocked by a lower priority thread for at most the duration of one  
7125 single critical section. Furthermore, when the protocol is correctly used in a single processor, and  
7126 if threads do not become blocked while owning mutexes, mutual deadlocks are prevented.

7127 The priority ceiling emulation can be extended to multiple processor environments, in which  
7128 case the values of the priority ceilings will be assigned depending on the kind of mutex that is  
7129 being used: local to only one processor, or global, shared by several processors. Local priority  
7130 ceilings will be assigned the usual way, equal to the priority of the highest priority thread that  
7131 may lock that mutex. Global priority ceilings will usually be assigned a priority level higher than  
7132 all the priorities assigned to any of the threads that reside in the involved processors to avoid the  
7133 effect called remote blocking.

#### 7134 **Change the Priority Ceiling of a Mutex**

7135 In order for the priority protect protocol to exhibit its desired properties of bounding priority  
7136 inversion and avoidance of deadlock, it is critical that the ceiling priority of a mutex be the same  
7137 as the priority of the highest thread that can ever hold it, or higher. Thus, if the priorities of the  
7138 threads using such mutexes never change dynamically, there is no need ever to change the  
7139 priority ceiling of a mutex.

7140 However, if a major system mode change results in an altered response time requirement for one  
7141 or more application threads, their priority has to change to reflect it. It will occasionally be the  
7142 case that the priority ceilings of mutexes held also need to change. While changing priority  
7143 ceilings should generally be avoided, it is important that IEEE Std 1003.1-200x provide these  
7144 interfaces for those cases in which it is necessary.

#### 7145 *B.2.9.5 Thread Cancellation*

7146 Many existing threads packages have facilities for canceling an operation or canceling a thread.  
7147 These facilities are used for implementing user requests (such as the CANCEL button in a  
7148 window-based application), for implementing OR parallelism (for example, telling the other  
7149 threads to stop working once one thread has found a forced mate in a parallel chess program), or  
7150 for implementing the ABORT mechanism in Ada.

7151 POSIX programs traditionally have used the signal mechanism combined with either *longjmp()*  
7152 or polling to cancel operations. Many POSIX programmers have trouble using these facilities to  
7153 solve their problems efficiently in a single-threaded process. With the introduction of threads,  
7154 these solutions become even more difficult to use.

7155 The main issues with implementing a cancellation facility are specifying the operation to be  
7156 canceled, cleanly releasing any resources allocated to that operation, controlling when the target  
7157 notices that it has been canceled, and defining the interaction between asynchronous signals and  
7158 cancellation.

**7159 Specifying the Operation to Cancel**

7160 Consider a thread that calls through five distinct levels of program abstraction and then, inside  
7161 the lowest-level abstraction, calls a function that suspends the thread. (An abstraction boundary  
7162 is a layer at which the client of the abstraction sees only the service being provided and can  
7163 remain ignorant of the implementation. Abstractions are often layered, each level of abstraction  
7164 being a client of the lower-level abstraction and implementing a higher-level abstraction.)  
7165 Depending on the semantics of each abstraction, one could imagine wanting to cancel only the  
7166 call that causes suspension, only the bottom two levels, or the operation being done by the entire  
7167 thread. Canceling operations at a finer grain than the entire thread is difficult because threads  
7168 are active and they may be run in parallel on a multi-processor. By the time one thread can make  
7169 a request to cancel an operation, the thread performing the operation may have completed that  
7170 operation and gone on to start another operation whose cancelation is not desired. Thread IDs  
7171 are not reused until the thread has exited, and either it was created with the *Attr detachstate*  
7172 attribute set to *PTHREAD\_CREATE\_DETACHED* or the *pthread\_join()* or *pthread\_detach()*  
7173 function has been called for that thread. Consequently, a thread cancelation will never be  
7174 misdirected when the thread terminates. For these reasons, the canceling of operations is done at  
7175 the granularity of the thread. Threads are designed to be inexpensive enough so that a separate  
7176 thread may be created to perform each separately cancelable operation; for example, each  
7177 possibly long running user request.

7178 For cancelation to be used in existing code, cancelation scopes and handlers will have to be  
7179 established for code that needs to release resources upon cancelation, so that it follows the  
7180 programming discipline described in the text.

**7181 A Special Signal Versus a Special Interface**

7182 Two different mechanisms were considered for providing the cancelation interfaces. The first  
7183 was to provide an interface to direct signals at a thread and then to define a special signal that  
7184 had the required semantics. The other alternative was to use a special interface that delivered the  
7185 correct semantics to the target thread.

7186 The solution using signals produced a number of problems. It required the implementation to  
7187 provide cancelation in terms of signals whereas a perfectly valid (and possibly more efficient)  
7188 implementation could have both layered on a low-level set of primitives. There were so many  
7189 exceptions to the special signal (it cannot be used with kill, no POSIX.1 interfaces can be used  
7190 with it) that it was clearly not a valid signal. Its semantics on delivery were also completely  
7191 different from any existing POSIX.1 signal. As such, a special interface that did not mandate the  
7192 implementation and did not confuse the semantics of signals and cancelation was felt to be the  
7193 better solution.

**7194 Races Between Cancelation and Resuming Execution**

7195 Due to the nature of cancelation, there is generally no synchronization between the thread  
7196 requesting the cancelation of a blocked thread and events that may cause that thread to resume  
7197 execution. For this reason, and because excess serialization hurts performance, when both an  
7198 event that a thread is waiting for has occurred and a cancelation request has been made and  
7199 cancelation is enabled, IEEE Std 1003.1-200x explicitly allows the implementation to choose  
7200 between returning from the blocking call or acting on the cancelation request.

7201 **Interaction of Cancellation with Asynchronous Signals**

7202 A typical use of cancellation is to acquire a lock on some resource and to establish a cancellation  
7203 cleanup handler for releasing the resource when and if the thread is canceled.

7204 A correct and complete implementation of cancellation in the presence of asynchronous signals  
7205 requires considerable care. An implementation has to push a cancellation cleanup handler on the  
7206 cancellation cleanup stack while maintaining the integrity of the stack data structure. If an  
7207 asynchronously generated signal is posted to the thread during a stack operation, the signal  
7208 handler cannot manipulate the cancellation cleanup stack. As a consequence, asynchronous  
7209 signal handlers may not cancel threads or otherwise manipulate the cancellation state of a thread.  
7210 Threads may, of course, be canceled by another thread that used a *sigwait()* function to wait  
7211 synchronously for an asynchronous signal.

7212 In order for cancellation to function correctly, it is required that asynchronous signal handlers not  
7213 change the cancellation state. This requires that some elements of existing practice, such as using  
7214 *longjmp()* to exit from an asynchronous signal handler implicitly, be prohibited in cases where  
7215 the integrity of the cancellation state of the interrupt thread cannot be ensured.

7216 **Thread Cancellation Overview**

## 7217 • Cancellability States

7218 The three possible cancellability states (disabled, deferred, and asynchronous) are encoded  
7219 into two separate bits ((disable, enable) and (deferred, asynchronous)) to allow them to be  
7220 changed and restored independently. For instance, short code sequences that will not block  
7221 sometimes disable cancellability on entry and restore the previous state upon exit. Likewise,  
7222 long or unbounded code sequences containing no convenient explicit cancellation points will  
7223 sometimes set the cancellability type to asynchronous on entry and restore the previous value  
7224 upon exit.

## 7225 • Cancellation Points

7226 Cancellation points are points inside of certain functions where a thread has to act on any  
7227 pending cancellation request when cancellability is enabled, if the function would block. As  
7228 with checking for signals, operations need only check for pending cancellation requests when  
7229 the operation is about to block indefinitely.

7230 The idea was considered of allowing implementations to define whether blocking calls such  
7231 as *read()* should be cancellation points. It was decided that it would adversely affect the  
7232 design of conforming applications if blocking calls were not cancellation points because  
7233 threads could be left blocked in an uncancelable state.

7234 There are several important blocking routines that are specifically not made cancellation  
7235 points:

7236 — *pthread\_mutex\_lock()*

7237 If *pthread\_mutex\_lock()* were a cancellation point, every routine that called it would also  
7238 become a cancellation point (that is, any routine that touched shared state would  
7239 automatically become a cancellation point). For example, *malloc()*, *free()*, and *rand()*  
7240 would become cancellation points under this scheme. Having too many cancellation points  
7241 makes programming very difficult, leading to either much disabling and restoring of  
7242 cancellability or much difficulty in trying to arrange for reliable cleanup at every possible  
7243 place.

7244 Since *pthread\_mutex\_lock()* is not a cancellation point, threads could result in being  
7245 blocked uninterruptibly for long periods of time if mutexes were used as a general

7246 synchronization mechanism. As this is normally not acceptable, mutexes should only be  
7247 used to protect resources that are held for small fixed lengths of time where not being  
7248 able to be canceled will not be a problem. Resources that need to be held exclusively for  
7249 long periods of time should be protected with condition variables.

7250 — *barrier\_wait()*

7251 Canceling a barrier wait will render a barrier unusable. Similar to a barrier timeout (which  
7252 the standard developers rejected), there is no way to guarantee the consistency of a  
7253 barrier's internal data structures if a barrier wait is canceled.

7254 — *pthread\_spin\_lock()*

7255 As with mutexes, spin locks should only be used to protect resources that are held for  
7256 small fixed lengths of time where not being cancelable will not be a problem.

7257 Every library routine should specify whether or not it includes any cancellation points.  
7258 Typically, only those routines that may block or compute indefinitely need to include  
7259 cancellation points.

7260 Correctly coded routines only reach cancellation points after having set up a cancellation  
7261 cleanup handler to restore invariants if the thread is canceled at that point. Being cancelable  
7262 only at specified cancellation points allows programmers to keep track of actions needed in a  
7263 cancellation cleanup handler more easily. A thread should only be made asynchronously  
7264 cancelable when it is not in the process of acquiring or releasing resources or otherwise in a  
7265 state from which it would be difficult or impossible to recover.

7266 • Thread Cancellation Cleanup Handlers

7267 The cancellation cleanup handlers provide a portable mechanism, easy to implement, for  
7268 releasing resources and restoring invariants. They are easier to use than signal handlers  
7269 because they provide a stack of cancellation cleanup handlers rather than a single handler,  
7270 and because they have an argument that can be used to pass context information to the  
7271 handler.

7272 The alternative to providing these simple cancellation cleanup handlers (whose only use is for  
7273 cleaning up when a thread is canceled) is to define a general exception package that could be  
7274 used for handling and cleaning up after hardware traps and software detected errors. This  
7275 was too far removed from the charter of providing threads to handle asynchrony. However,  
7276 it is an explicit goal of IEEE Std 1003.1-200x to be compatible with existing exception facilities  
7277 and languages having exceptions.

7278 The interaction of this facility and other procedure-based or language-level exception  
7279 facilities is unspecified in this version of IEEE Std 1003.1-200x. However, it is intended that it  
7280 be possible for an implementation to define the relationship between these cancellation  
7281 cleanup handlers and Ada, C++, or other language-level exception handling facilities.

7282 It was suggested that the cancellation cleanup handlers should also be called when the  
7283 process exits or calls the *exec* function. This was rejected partly due to the performance  
7284 problem caused by having to call the cancellation cleanup handlers of every thread before the  
7285 operation could continue. The other reason was that the only state expected to be cleaned up  
7286 by the cancellation cleanup handlers would be the intraprocess state. Any handlers that are to  
7287 clean up the interprocess state would be registered with *atexit()*. There is the orthogonal  
7288 problem that the *exec* functions do not honor the *atexit()* handlers, but resolving this is  
7289 beyond the scope of IEEE Std 1003.1-200x.

## 7290 • Async-Cancel Safety

7291 A function is said to be *async-cancel safe* if it is written in such a way that entering the function  
7292 with asynchronous cancelability enabled will not cause any invariants to be violated, even if  
7293 a cancelation request is delivered at any arbitrary instruction. Functions that are *async-*  
7294 *cancel-safe* are often written in such a way that they need to acquire no resources for their  
7295 operation and the visible variables that they may write are strictly limited.

7296 Any routine that gets a resource as a side-effect cannot be made *async-cancel-safe* (for  
7297 example, *malloc()*). If such a routine were called with asynchronous cancelability enabled, it  
7298 might acquire the resource successfully, but as it was returning to the client, it could act on a  
7299 cancelation request. In such a case, the application would have no way of knowing whether  
7300 the resource was acquired or not.

7301 Indeed, because many interesting routines cannot be made *async-cancel-safe*, most library  
7302 routines in general are not *async-cancel-safe*. Every library routine should specify whether or  
7303 not it is *async-cancel safe* so that programmers know which routines can be called from code  
7304 that is asynchronously cancelable.

7305 **B.2.9.6 Thread Read-Write Locks**7306 **Background**

7307 Read-write locks are often used to allow parallel access to data on multi-processors, to avoid  
7308 context switches on uni-processors when multiple threads access the same data, and to protect  
7309 data structures that are frequently accessed (that is, read) but rarely updated (that is, written).  
7310 The in-core representation of a file system directory is a good example of such a data structure.  
7311 One would like to achieve as much concurrency as possible when searching directories, but limit  
7312 concurrent access when adding or deleting files.

7313 Although read-write locks can be implemented with mutexes and condition variables, such  
7314 implementations are significantly less efficient than is possible. Therefore, this synchronization  
7315 primitive is included in IEEE Std 1003.1-200x for the purpose of allowing more efficient  
7316 implementations in multi-processor systems.

7317 **Queuing of Waiting Threads**

7318 The *pthread\_rwlock\_unlock()* function description states that one writer or one or more readers  
7319 shall acquire the lock if it is no longer held by any thread as a result of the call. However, the  
7320 function does not specify which thread(s) acquire the lock, unless the Thread Execution  
7321 Scheduling option is supported.

7322 The standard developers considered the issue of scheduling with respect to the queuing of  
7323 threads blocked on a read-write lock. The question turned out to be whether  
7324 IEEE Std 1003.1-200x should require priority scheduling of read-write locks for threads whose  
7325 execution scheduling policy is priority-based (for example, *SCHED\_FIFO* or *SCHED\_RR*). There  
7326 are tradeoffs between priority scheduling, the amount of concurrency achievable among readers,  
7327 and the prevention of writer and/or reader starvation.

7328 For example, suppose one or more readers hold a read-write lock and the following threads  
7329 request the lock in the listed order:

7330 pthread\_rwlock\_wrlock() - Low priority thread writer\_a  
 7331 pthread\_rwlock\_rdlock() - High priority thread reader\_a  
 7332 pthread\_rwlock\_rdlock() - High priority thread reader\_b  
 7333 pthread\_rwlock\_rdlock() - High priority thread reader\_c

7334 When the lock becomes available, should *writer\_a* block the high priority readers? Or, suppose a  
 7335 read-write lock becomes available and the following are queued:

7336 pthread\_rwlock\_rdlock() - Low priority thread reader\_a  
 7337 pthread\_rwlock\_rdlock() - Low priority thread reader\_b  
 7338 pthread\_rwlock\_rdlock() - Low priority thread reader\_c  
 7339 pthread\_rwlock\_wrlock() - Medium priority thread writer\_a  
 7340 pthread\_rwlock\_rdlock() - High priority thread reader\_d

7341 If priority scheduling is applied then *reader\_d* would acquire the lock and *writer\_a* would block  
 7342 the remaining readers. But should the remaining readers also acquire the lock to increase  
 7343 concurrency? The solution adopted takes into account that when the Thread Execution  
 7344 Scheduling option is supported, high priority threads may in fact starve low priority threads (the  
 7345 application developer is responsible in this case to design the system in such a way that this  
 7346 starvation is avoided). Therefore, IEEE Std 1003.1-200x specifies that high priority readers take  
 7347 precedence over lower priority writers. However, to prevent writer starvation from threads of  
 7348 the same or lower priority, writers take precedence over readers of the same or lower priority.

7349 Priority inheritance mechanisms are non-trivial in the context of read-write locks. When a high  
 7350 priority writer is forced to wait for multiple readers, for example, it is not clear which subset of  
 7351 the readers should inherit the writer's priority. Furthermore, the internal data structures that  
 7352 record the inheritance must be accessible to all readers, and this implies some sort of  
 7353 serialization that could negate any gain in parallelism achieved through the use of multiple  
 7354 readers in the first place. Finally, existing practice does not support the use of priority  
 7355 inheritance for read-write locks. Therefore, no specification of priority inheritance or priority  
 7356 ceiling is attempted. If reliable priority-scheduled synchronization is absolutely required, it can  
 7357 always be obtained through the use of mutexes.

### 7358 **Comparison to *fcntl()* Locks**

7359 The read-write locks and the *fcntl()* locks in IEEE Std 1003.1-200x share a common goal:  
 7360 increasing concurrency among readers, thus increasing throughput and decreasing delay.

7361 However, the read-write locks have two features not present in the *fcntl()* locks. First, under  
 7362 priority scheduling, read-write locks are granted in priority order. Second, also under priority  
 7363 scheduling, writer starvation is prevented by giving writers preference over readers of equal or  
 7364 lower priority.

7365 Also, read-write locks can be used in systems lacking a file system, such as those conforming to  
 7366 the minimal realtime system profile of IEEE Std 1003.13-1998.

### 7367 **History of Resolution Issues**

7368 Based upon some balloting objections, the draft specified the behavior of threads waiting on a  
 7369 read-write lock during the execution of a signal handler, as if the thread had not called the lock  
 7370 operation. However, this specified behavior would require implementations to establish  
 7371 internal signal handlers even though this situation would be rare, or never happen for many  
 7372 programs. This would introduce an unacceptable performance hit in comparison to the little  
 7373 additional functionality gained. Therefore, the behavior of read-write locks and signals was  
 7374 reverted back to its previous mutex-like specification.

7375 **B.2.9.7** *Thread Interactions with Regular File Operations*

7376 There is no additional rationale provided for this section.

7377 **B.2.10** **Sockets**

7378 The base document for the sockets interfaces in IEEE Std 1003.1-200x is the XNS, Issue 5.2  
7379 specification. This was primarily chosen as it aligns with IPv6. Additional material has been  
7380 added from IEEE Std 1003.1g-2000, notably socket concepts, raw sockets, the *pselect()* function,  
7381 and the `<sys/select.h>` header.

7382 **B.2.10.1** *Address Families*

7383 There is no additional rationale provided for this section.

7384 **B.2.10.2** *Addressing*

7385 There is no additional rationale provided for this section.

7386 **B.2.10.3** *Protocols*

7387 There is no additional rationale provided for this section.

7388 **B.2.10.4** *Routing*

7389 There is no additional rationale provided for this section.

7390 **B.2.10.5** *Interfaces*

7391 There is no additional rationale provided for this section.

7392 **B.2.10.6** *Socket Types*

7393 The type `socklen_t` was invented to cover the range of implementations seen in the field. The  
7394 intent of `socklen_t` is to be the type for all lengths that are naturally bounded in size; that is, that  
7395 they are the length of a buffer which cannot sensibly become of massive size: network addresses,  
7396 host names, string representations of these, ancillary data, control messages, and socket options  
7397 are examples. Truly boundless sizes are represented by `size_t` as in *read()*, *write()*, and so on.

7398 All `socklen_t` types were originally (in BSD UNIX) of type `int`. During the development of  
7399 IEEE Std 1003.1-200x, it was decided to change all buffer lengths to `size_t`, which appears at face  
7400 value to make sense. When dual mode 32/64-bit systems came along, this choice unnecessarily  
7401 complicated system interfaces because `size_t` (with `long`) was a different size under ILP32 and  
7402 LP64 models. Reverting to `int` would have happened except that some implementations had  
7403 already shipped 64-bit-only interfaces. The compromise was a type which could be defined to be  
7404 any size by the implementation: `socklen_t`.

7405 **B.2.10.7** *Socket I/O Mode*

7406 There is no additional rationale provided for this section.

7407 *B.2.10.8 Socket Owner*

7408 There is no additional rationale provided for this section.

7409 *B.2.10.9 Socket Queue Limits*

7410 There is no additional rationale provided for this section.

7411 *B.2.10.10 Pending Error*

7412 There is no additional rationale provided for this section.

7413 *B.2.10.11 Socket Receive Queue*

7414 There is no additional rationale provided for this section.

7415 *B.2.10.12 Socket Out-of-Band Data State*

7416 There is no additional rationale provided for this section.

7417 *B.2.10.13 Connection Indication Queue*

7418 There is no additional rationale provided for this section.

7419 *B.2.10.14 Signals*

7420 There is no additional rationale provided for this section.

7421 *B.2.10.15 Asynchronous Errors*

7422 There is no additional rationale provided for this section.

7423 *B.2.10.16 Use of Options*

7424 There is no additional rationale provided for this section.

7425 *B.2.10.17 Use of Sockets for Local UNIX Connections*

7426 There is no additional rationale provided for this section.

7427 *B.2.10.18 Use of Sockets over Internet Protocols*

7428 A raw socket allows privileged users direct access to a protocol; for example, raw access to the  
7429 IP and ICMP protocols is possible through raw sockets. Raw sockets are intended for  
7430 knowledgeable applications that wish to take advantage of some protocol feature not directly  
7431 accessible through the other sockets interfaces.

7432 *B.2.10.19 Use of Sockets over Internet Protocols Based on IPv4*

7433 There is no additional rationale provided for this section.

7434 *B.2.10.20 Use of Sockets over Internet Protocols Based on IPv6*

7435 There is no additional rationale provided for this section.

7436 **B.2.11 Tracing**

7437 The organization of the tracing rationale differs from the traditional rationale in that this tracing  
7438 rationale text is written against the trace interface as a whole, rather than against the individual  
7439 components of the trace interface or the normative section in which those components are  
7440 defined. Therefore the sections below do not parallel the sections of normative text in  
7441 IEEE Std 1003.1-200x.

7442 *B.2.11.1 Objectives*

7443 The intended uses of tracing are application-system debugging during system development, as a  
7444 “flight recorder” for maintenance of fielded systems, and as a performance measurement tool. In  
7445 all of these intended uses, the vendor-supplied computer system and its software are, for this  
7446 discussion, assumed error-free; the intent being to debug the user-written and/or third-party  
7447 application code, and their interactions. Clearly, problems with the vendor-supplied system and  
7448 its software will be uncovered from time to time, but this is a byproduct of the primary activity,  
7449 debugging user code.

7450 Another need for defining a trace interface in POSIX stems from the objective to provide an  
7451 efficient portable way to perform benchmarks. Existing practice shows that such interfaces are  
7452 commonly used in a variety of systems but with little commonality. As part of the benchmarking  
7453 needs, we must consider two aspects within the trace interface.

7454 The first, and perhaps more important one, is the qualitative aspect.

7455 The second is the quantitative aspect.

## 7456 • Qualitative Aspect

7457 To better understand this aspect, let us consider an example. Suppose that you want to  
7458 organize a number of actions to be performed during the day. Some of these actions are  
7459 known at the beginning of the day. Some others, which may be more or less important, will  
7460 be triggered by reading your mail. During the day you will make some phone calls and  
7461 synchronously receive some more information. Finally you will receive asynchronous phone  
7462 calls that also will trigger actions. If you, or somebody else, examines your day at work, you,  
7463 or he, can discover that you have not efficiently organized your work. For instance, relative  
7464 to the phone calls you made, would it be preferable to make some of these early in the  
7465 morning? Or to delay some others until the end of the day? Relative to the phone calls you  
7466 have received, you might find that somebody you called in the morning has called you 10  
7467 times while you were performing some important work. To examine, afterwards, your day at  
7468 work, you record in sequence all the trace events relative to your work. This should give you  
7469 a chance of organizing your next day at work.

7470 This is the qualitative aspect of the trace interface. The user of a system needs to keep a trace  
7471 of particular points the application passes through, so that he can eventually make some  
7472 changes in the application and/or system configuration, to give the application a chance of  
7473 running more efficiently.

## 7474 • Quantitative Aspect

7475 This aspect concerns primarily realtime applications, where missed deadlines can be  
7476 undesirable. Although there are, in IEEE Std 1003.1-200x, some interfaces useful for such  
7477 applications (timeouts, execution time monitoring, and so on), there are no APIs to aid in the  
7478 tuning of a realtime application's behavior (**timespec** in timeouts, length of message queues,  
7479 duration of driver interrupt service routine, and so on). The tuning of an application needs a  
7480 means of recording timestamped important trace events during execution in order to analyze  
7481 offline, and eventually, to tune some realtime features (redesign the system with less

7482 functionalities, readjust timeouts, redesign driver interrupts, and so on).

### 7483 Detailed Objectives

7484 Objectives were defined to build the trace interface and are kept for historical interest. Although  
7485 some objectives are not fully respected in this trace interface, the concept of the POSIX trace  
7486 interface assumes the following points:

- 7487 1. It shall be possible to trace both system and user trace events concurrently.
- 7488 2. It must be possible to trace per-process trace events and also to trace system trace events  
7489 which are unrelated to any particular process. A per-process trace event is either user-  
7490 initiated or system-initiated.
- 7491 3. It must be possible to control tracing on a per process basis from either inside or outside  
7492 the process.
- 7493 4. It must be possible to control tracing on a per-thread basis from inside the enclosing  
7494 process.
- 7495 5. Trace points shall be controllable by trace event type ID from inside and outside of the  
7496 process. Multiple trace points can have the same trace event type ID, and will be controlled  
7497 jointly.
- 7498 6. Recording of trace events is dependent on both trace event type ID and the  
7499 process/thread. Both must be enabled in order to record trace events. System trace events  
7500 may or may not be handled differently.
- 7501 7. The API shall not mandate the ability to control tracing for more than one process at the  
7502 same time.
- 7503 8. There is no objective for trace control on anything bigger than a process; for example,  
7504 group or session.
- 7505 9. Trace propagation and control:
  - 7506 a. Trace propagation across fork is optional; the default is to not trace a child process.
  - 7507 b. Trace control shall span *thread\_create* operations; that is, if a process is being traced,  
7508 any thread will be traced as well if this thread allows tracing. The default is to allow  
7509 tracing.
- 7510 10. Trace control shall not span *exec* or *spawn* operations.
- 7511 11. A triggering API is not required. The triggering API is the ability to command or stop  
7512 tracing based on the occurrence of specific trace event other than a `POSIX_TRACE_START`  
7513 trace event or a `POSIX_TRACE_STOP` trace event.
- 7514 12. Trace log entries shall have timestamps of implementation-defined resolution.  
7515 Implementations are exhorted to support at least microsecond resolution. When a trace log  
7516 entry is retrieved, it shall have timestamp, PC address, PID, and TID of the entity that  
7517 generated the trace event.
- 7518 13. Independently developed code should be able to use trace facilities without coordination  
7519 and without conflict.
- 7520 14. Even if the trace points in the trace calls are not unique, the trace log entries (after any  
7521 processing) shall be uniquely identified as to trace point.
- 7522 15. There shall be a standard API to read the trace stream.

- 7523 16. The format of the trace stream and the trace log is opaque and unspecified.
- 7524 17. It shall be possible to read a completed trace, if recorded on some suitable non-volatile  
7525 storage, even subsequent to a power cycle or subsequent cold boot of the system.
- 7526 18. Support of analysis of a trace log while it is being formed is implementation-defined.
- 7527 19. The API shall allow the application to write trace stream identification information into the  
7528 trace stream and to be able to retrieve it, without it being overwritten by trace entries, even  
7529 if the trace stream is full.
- 7530 20. It must be possible to specify the destination of trace data produced by trace events.
- 7531 21. It must be possible to have different trace streams, and for the tracing enabled by one trace  
7532 stream to be completely independent of the tracing of another trace stream.
- 7533 22. It must be possible to trace events from threads in different CPUs.
- 7534 23. The API shall support one or more trace streams per-system, and one or more trace  
7535 streams per-process, up to an implementation-defined set of per-system and per-process  
7536 maximums.
- 7537 24. It shall be possible to determine the order in which the trace events happened, without  
7538 necessarily depending on the clock, up to an implementation-defined time resolution.
- 7539 25. For performance reasons, the trace event point call(s) shall be implementable as a macro  
7540 (see the ISO POSIX-1: 1996 standard, 1.3.4, Statement 2).
- 7541 26. IEEE Std 1003.1-200x must not define the trace points which a conforming system must  
7542 implement, except for trace points used in the control of tracing.
- 7543 27. The APIs shall be thread-safe, and trace points should be lock-free (that is shall not require  
7544 a lock to gain exclusive access to some resource).
- 7545 28. The user-provided information associated with a trace event is variable-sized, up to some  
7546 maximum size.
- 7547 29. Bounds on record and trace stream sizes:
- 7548 a. The API must permit the application to declare the upper bounds on the length of an  
7549 application data record. The system shall return the limit it used. The limit used may  
7550 be smaller than requested.
- 7551 b. The API must permit the application to declare the upper bounds on the size of trace  
7552 streams. The system shall return the limit it used. The limit used may be different,  
7553 either larger or smaller, than requested.
- 7554 30. The API must be able to pass any fundamental data type, and a structured data type  
7555 composed only of fundamental types. The API must be able to pass data by reference,  
7556 given only as an address and a length. Fundamental types are the POSIX.1 types (see the  
7557 `<sys/types.h>` header) plus those defined in the ISO C standard.
- 7558 31. The API shall apply the POSIX notions of ownership and permission to recorded trace  
7559 data, corresponding to the sources of that data.

7560 **Comments on Objectives**

7561 **Note:** In the following comments, numbers in square brackets refer to the above objectives.

7562 It is necessary to be able to obtain a trace stream for a complete activity. This means we need to  
7563 be able to trace both application and system trace events. A per-process trace event is either  
7564 user-initiated, like the *write()* POSIX call, or system-initiated, like a timer expiration. We also  
7565 need to be able to trace an entire process's activity even when it has threads in multiple CPUs. To  
7566 avoid excess trace activity, it is necessary to be able to control tracing on a trace event type basis.  
7567 [Objectives 1,2,5,22]

7568 We need to be able to control tracing on a per-process basis, both from inside and outside the  
7569 process; that is, a process can start a trace activity on itself or any other process. We also see the  
7570 need to allow the definition of a maximum number trace streams per system.  
7571 [Objectives 3,23]

7572 From within a process, it is necessary to be able to control tracing on a per-thread basis. This  
7573 provides an additional filtering capability to keep the amount of traced data to a minimum. It  
7574 also allows for less ambiguity as to the origin of trace events. It is recognized that thread-level  
7575 control is only valid from within the process itself. It is also desirable to know the maximum  
7576 number of trace streams per process that can be started. We do not want the API to require  
7577 thread synchronization or to mandate priority inversions that would cause the thread to block.  
7578 However, the API must be thread-safe.  
7579 [Objectives 4,23,24,27]

7580 We see no objective to control tracing on anything larger than a process; for example, a group or  
7581 session. Also, the ability to start or stop a trace activity on multiple processes atomically may be  
7582 very difficult or cumbersome in some implementations.  
7583 [Objectives 6,8]

7584 It is also necessary to be able to control tracing by trace event type identifier, sometimes called a  
7585 trace hook ID. However, there is no mandated set of system trace events, since such trace points  
7586 are very system-dependent. The API must not require from the operating system facilities that  
7587 are not standard (POSIX).  
7588 [Objectives 6,26]

7589 Trace control must span *fork()* and *pthread\_create()*. If not, there will be no way to ensure that a  
7590 program's activity is entirely traced. The newly forked child would not be able to turn on its  
7591 tracing until after it obtained control after the fork, and trace control externally would be even  
7592 more problematic.  
7593 [Objective 9]

7594 Since *exec()* and *spawn()* represent a complete change in the execution of a task (a new  
7595 program), trace control need not persist over an *exec()* or *spawn()*.  
7596 [Objective 10]

7597 Where trace activities are started on multiple processes, these trace activities should not interfere  
7598 with each other.  
7599 [Objective 21]

7600 There is no need for a triggering objective, primarily for performance reasons; see also Section  
7601 B.2.11.8 (on page 3491), rationale on triggering.  
7602 [Objective 11]

7603 It must be possible to determine the origin of each traced event. We need the process and thread  
7604 identifiers for each trace event. We also saw the need for a user-specifiable origin, but felt this  
7605 would create too much overhead.  
7606 [Objectives 12,14]

7607 We must allow for trace points to come embedded in software components from several  
7608 different sources and vendors without requiring coordination.  
7609 [Objective 13]

7610 We need to be able to uniquely identify trace points that may have the same trace stream  
7611 identifier. We only need to be able to do this when a trace report is produced.  
7612 [Objectives 12,14]

7613 Tracing is a very performance-sensitive activity, and will therefore likely be implemented at a  
7614 low level within the system. Hence the interface shall not mandate any particular buffering or  
7615 storage method. Therefore, we will need a standard API to read a trace stream. Also the interface  
7616 shall not mandate the format of the trace data, and the interface shall not assume a trace storage  
7617 method. Due to the possibility of a monolithic kernel and the possible presence of multiple  
7618 processes capable of running trace activities, the two kinds of trace events may be stored in two  
7619 separate streams for performance reasons. A mandatory dump mechanism, common in some  
7620 existing practice, has been avoided to allow the implementation of this set of functions on small  
7621 realtime profiles for which the concept of a file system is not defined. The trace API calls should  
7622 be implemented as macros.  
7623 [Objectives 15,16,25,30]

7624 Since a trace facility is a valuable service tool, the output (or log) of a completed trace stream  
7625 that is written to permanent storage must be readable on other systems of the type that  
7626 produced the trace log. Note that there is no objective to be able to interpret a trace log that was  
7627 not successfully completed.  
7628 [Objectives 17,18,19]

7629 For trace streams written to permanent storage, a way to specify the destination of the trace  
7630 stream is needed.  
7631 [Objective 20]

7632 We need to be able to depend on the ordering of trace events up to some implementation- |  
7633 defined time interval. For example, we need to know the time period which, if trace events are |  
7634 closer together, their ordering is unspecified. Events that occur within an interval smaller than |  
7635 this resolution may or may not be read back in the correct order. |  
7636 [Objective 24]

7637 The application should be able to know how much data can be traced. When trace event types  
7638 can be filtered, the application should be able to specify the approximate maximum amount of  
7639 data that will be traced in a trace event so resources can be more efficiently allocated.  
7640 [Objectives 28,29]

7641 Users should not be able to trace data to which they would not normally have access to. System  
7642 trace events corresponding to a process/thread should be associated with the ownership of that  
7643 process/thread.  
7644 [Objective 31] |

7645 B.2.11.2 Trace Model

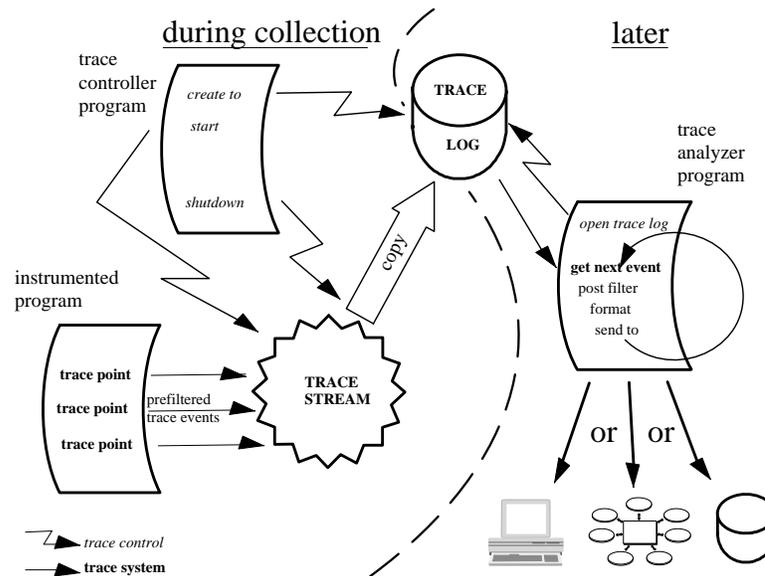
7646 **Introduction**

7647 The model is based on two base entities: the “Trace Stream” and the “Trace Log” , and a  
 7648 recorded unit called the “Trace Event”. The possibility of using Trace Streams and Trace Logs  
 7649 separately gives us two use dimensions and solves both the performance issue and the full-  
 7650 information system issue. In the case of a trace stream without log, specific information,  
 7651 although reduced in quantity, is required to be registered, in a possibly small realtime system,  
 7652 with as little overhead as possible. The Trace Log option has been added for small realtime  
 7653 systems. In the case of a trace stream with log, considerable complex application-specific  
 7654 information needs to be collected.

7655 **Trace Model Description**

7656 The trace model can be examined for three different subfunctions: Application Instrumentation,  
 7657 Trace Operation Control, and Trace Analysis.

7658



7659 **Figure B-2** Trace System Overview: for Offline Analysis

7660 Each of these subfunctions requires specific characteristics of the trace mechanism API.

7661 • Application Instrumentation

7662 When instrumenting an application, the programmer has no concern about the future  
 7663 utilization of the trace events in trace stream or trace log, the full policy of trace stream, or  
 7664 the eventual pre-filtering of trace events. But he is concerned about the correct determination  
 7665 of specific trace event type identifier, regardless of how many independent libraries are used  
 7666 in the same user application; see Figure B-2 and Figure B-3 (on page 3475).

7667 This trace API shall provide the necessary operations to accomplish this subfunction. This is  
 7668 done by providing functions to associate a programmer-defined name with an  
 7669 implementation-defined trace event type identifier; see the *posix\_trace\_eventid\_open()*  
 7670 function), and to send this trace event into a potential trace stream (see the  
 7671 *posix\_trace\_event()* function).

7672 • Trace Operation Control

7673 When controlling the recording of trace events in a trace stream, the programmer is  
 7674 concerned with the correct initialization of the trace mechanism (that is, the sizing of the  
 7675 trace stream), the correct retention of trace events in a permanent storage, the correct  
 7676 dynamic recording of trace events, and so on.

7677 This trace API shall provide the necessary material to permit this efficiently. This is done by  
 7678 providing functions to initialize a new trace stream, and optionally a trace log:

- 7679 — Trace Stream Attributes Object Initialization (see *posix\_trace\_attr\_init()*)
- 7680 — Functions to Retrieve or Set Information About a Trace Stream (see  
 7681 *posix\_trace\_attr\_getgenversion()*)
- 7682 — Functions to Retrieve or Set the Behavior of a Trace Stream (see  
 7683 *posix\_trace\_attr\_getinherited()*)
- 7684 — Functions to Retrieve or Set Trace Stream Size Attributes (see  
 7685 *posix\_trace\_attr\_getmaxuseventsizesize()*)
- 7686 — Trace Stream Initialization, Flush, and Shutdown from a Process (see *posix\_trace\_create()*)
- 7687 — Clear Trace Stream and Trace Log (see *posix\_trace\_clear()*)

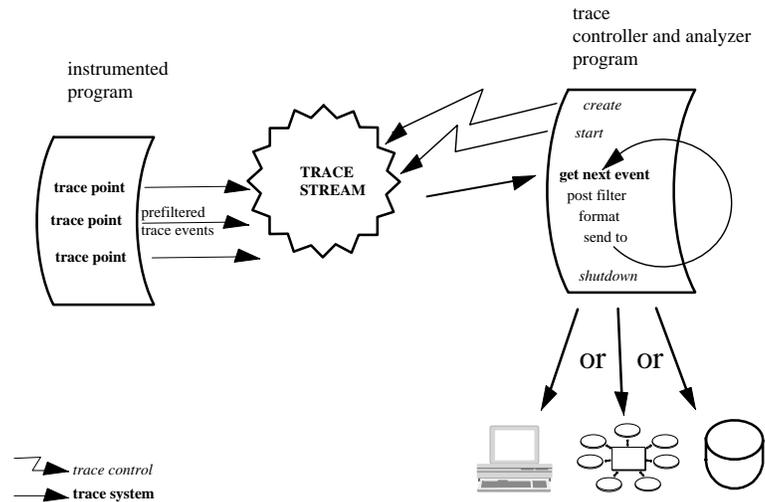
7688 To select the trace event types that are to be traced:

- 7689 — Manipulate Trace Event Type Identifier (see *posix\_trace\_trid\_eventid\_open()*)
- 7690 — Iterate over a Mapping of Trace Event Type (see *posix\_trace\_eventtypelist\_getnext\_id()*)
- 7691 — Manipulate Trace Event Type Sets (see *posix\_trace\_eventset\_empty()*)
- 7692 — Set Filter of an Initialized Trace Stream (see *posix\_trace\_set\_filter()*)

7693 To control the execution of an active trace stream:

- 7694 — Trace Start and Stop (see *posix\_trace\_start()*)
- 7695 — Functions to Retrieve the Trace Attributes or Trace Statuses (see *posix\_trace\_get\_attr()*)

7696



7697

**Figure B-3** Trace System Overview: for Online Analysis

7698

- Trace Analysis

7699

7700

7701

Once correctly recorded, on permanent storage or not, an ultimate activity consists of the analysis of the recorded information. If the recorded data is on permanent storage, a specific open operation is required to associate a trace stream to a trace log.

7702

7703

7704

7705

7706

7707

The first intent of the group was to request the presence of a system identification structure in the trace stream attribute. This was, for the application, to allow some portable way to process the recorded information. However, there is no requirement that the **utsname** structure, on which this system identification was based, be portable from one machine to another, so the contents of the attribute cannot be interpreted correctly by an application conforming to IEEE Std 1003.1-200x.

7708

7709

7710

7711

7712

Draft 6 incorporates this modification and requests that some unspecified information be recorded in the trace log in order to fail opening it if the analysis process and the controller process were running in different types of machine, but does not request that this information be accessible to the application. This modification has implied a modification in the `posix_trace_open()` function error code returns.

7713

This trace API shall provide functions to:

7714

- Extract trace stream identification attributes (see `posix_trace_attr_getgenversion()`)

7715

- Extract trace stream behavior attributes (see `posix_trace_attr_getinherited()`)

7716

7717

- Extract trace event, stream, and log size attributes (see `posix_trace_attr_getmaxusereventsize()`)

7718

- Look up trace event type names (see `posix_trace_eventid_get_name()`)

- 7719 — Iterate over trace event type identifiers (see *posix\_trace\_eventtypelist\_getnext\_id()*)
- 7720 — Open, rewind, and close a trace log (see *posix\_trace\_open()*)
- 7721 — Read trace stream attributes and status (see *posix\_trace\_get\_attr()*)
- 7722 — Read trace events (see *posix\_trace\_getnext\_event()*)

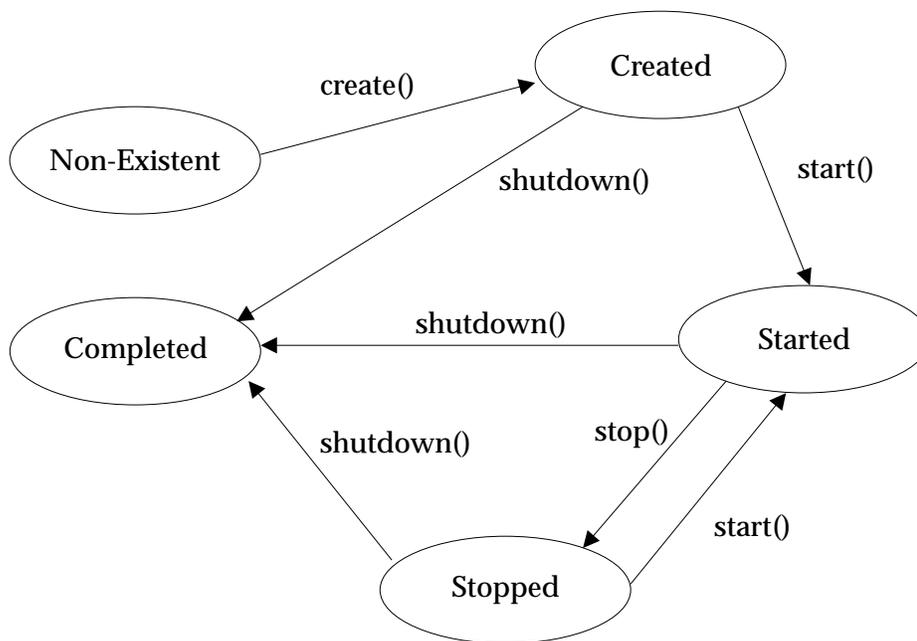
7723 Due to the following two reasons:

- 7724 1. The requirement that the trace system must not add unacceptable overhead to the traced process and so that the trace event point execution must be fast
- 7725
- 7726 2. The traced application does not care about tracing errors

7727 the trace system cannot return any internal error to the application. Internal error conditions can range from unrecoverable errors that will force the active trace stream to abort, to small errors that can affect the quality of tracing without aborting the trace stream. The group decided to define a system trace event to report to the analysis process such internal errors. It is not the intention of IEEE Std 1003.1-200x to require an implementation to report an internal error that corrupts or terminates tracing operation. The implementor is free to decide which internal documented errors, if any, the trace system is able to report.

7734 **States of a Trace Stream**

7735



7736 **Figure B-4** Trace System Overview: States of a Trace Stream

7737 Figure B-4 shows the different states an active trace stream passes through. After the  
 7738 *posix\_trace\_create()* function call, a trace stream becomes CREATED and a trace stream is  
 7739 associated for the future collection of trace events. The status of the trace stream is  
 7740 POSIX\_TRACE\_SUSPENDED. The state becomes STARTED after a call to the *posix\_trace\_start()*  
 7741 function, and the status becomes POSIX\_TRACE\_RUNNING. In this state, all trace events that  
 7742 are not filtered out shall be stored into the trace stream. After a call to *posix\_trace\_stop()*, the  
 7743 trace stream becomes STOPPED (and the status POSIX\_TRACE\_SUSPENDED). In this state, no

7744 new trace events will be recorded in the trace stream, but previously recorded trace events may  
7745 continue to be read.

7746 After a call to *posix\_trace\_shutdown()*, the trace stream is in the state COMPLETED. The trace  
7747 stream no longer exists but, if the Trace Log option is supported, all the information contained in  
7748 it has been logged. If a log object has not been associated with the trace stream at the creation, it  
7749 is the responsibility of the trace controller process to not shut the trace stream down while trace  
7750 events remain to be read in the stream.

### 7751 **Tracing All Processes**

7752 Some implementations have a tracing subsystem with the ability to trace all processes. This is  
7753 useful to debug some types of device drivers such as those for ATM or X25 adapters. These types  
7754 of adapters are used by several independent processes, that are not issued from the same  
7755 process.

7756 The POSIX trace interface does not define any constant or option to create a trace stream tracing  
7757 all processes. But the POSIX trace interface does not prevent this type of implementation and the  
7758 implementor is free to add this capability. Nevertheless, the POSIX trace interface allows to trace  
7759 all the system trace events and all the processes issued from the same process.

7760 If such a tracing system capability has to be implemented, when a trace stream is created, it is  
7761 recommended that a constant named POSIX\_TRACE\_ALLPROC be used instead of the process  
7762 identifier in the argument of the function *posix\_trace\_create()* or *posix\_trace\_create\_withlog()*. A  
7763 possible value for POSIX\_TRACE\_ALLPROC may be -1 instead of a real process identifier.

7764 The implementor has to be aware that there is some impact on the tracing behavior as defined in  
7765 the POSIX trace interface. For example:

- 7766 • If the default value for the inheritance attribute is to set to  
7767 POSIX\_TRACE\_CLOSE\_FOR\_CHILD, the implementation has to stop tracing for the child  
7768 process.
- 7769 • The trace controller which is creating this type of trace stream must have the appropriate  
7770 privilege to trace all the processes.

### 7771 **Trace Storage**

7772 The model is based on two types of trace events: system trace events and user-defined trace  
7773 events. The internal representation of trace events is implementation-defined, and so the  
7774 implementor is free to choose the more suitable, practical, and efficient way to design the  
7775 internal management of trace events. For the timestamping operation, the model does not  
7776 impose the CLOCK\_REALTIME or any other clock. The buffering allocation and operation  
7777 follow the same principle. The implementor is free to use one or more buffers to record trace  
7778 events; the interface assumes only a logical trace stream of sequentially recorded trace events.  
7779 Regarding flushing of trace events, the interface allows the definition of a trace log object which  
7780 typically can be a file. But the group was also aware of defining functions to permit the use of  
7781 this interface in small realtime systems, which may not have general file system capabilities. For  
7782 instance, the three functions *posix\_trace\_getnext\_event()* (blocking),  
7783 *posix\_trace\_timedgetnext\_event()* (blocking with timeout), and *posix\_trace\_trygetnext\_event()*  
7784 (non-blocking) are proposed to read the recorded trace events.

7785 The policy to be used when the trace stream becomes full also relies on common practice:

- 7786 • For an active trace stream, the POSIX\_TRACE\_LOOP trace stream policy permits automatic  
7787 overrun (overwrite of oldest trace events) while waiting for some user-defined condition to  
7788 cause tracing to stop. By contrast, the POSIX\_TRACE\_UNTIL\_FULL trace stream policy

7789 requires the system to stop tracing when the trace stream is full. However, if the trace stream  
 7790 that is full is at least partially emptied by a call to the *posix\_trace\_flush()* function or by calls  
 7791 to *posix\_trace\_getnext\_event()* function, the trace system will automatically resume tracing.

7792 If the Trace Log option is supported the operation of the POSIX\_TRACE\_FLUSH policy is an  
 7793 extension of the POSIX\_TRACE\_UNTIL\_FULL policy. The automatic free operation (by  
 7794 flushing to the associated trace log) is added.

7795 • If a log is associated with the trace stream and this log is a regular file, these policies also  
 7796 apply for the log. One more policy, POSIX\_TRACE\_APPEND, is defined to allow indefinite  
 7797 extension of the log. Since the log destination can be any device or pseudo-device, the  
 7798 implementation may not be able to manipulate the destination as required by  
 7799 IEEE Std 1003.1-200x. For this reason, the behavior of the log full policy may be unspecified  
 7800 depending of the trace log type.

7801 The current trace interface does not define a service to preallocate space for a trace log file,  
 7802 because this space can be preallocated by means of a call to the *posix\_fallocate()* function. This  
 7803 function could be called after the file has been opened, but before the trace stream is created.  
 7804 The *posix\_fallocate()* function ensures that any required storage for regular file data is  
 7805 allocated on the file system storage media. If *posix\_fallocate()* returns successfully,  
 7806 subsequent writes to the specified file data shall not fail due to the lack of free space on the  
 7807 file system storage media. Besides trace events, a trace stream also includes trace attributes  
 7808 and the mapping from trace event names to trace event type identifiers. The implementor is  
 7809 free to choose how to store the trace attributes and the trace event type map, but must ensure  
 7810 that this information is not lost when a trace stream overrun occurs.

### 7811 B.2.11.3 Trace Programming Examples

7812 Several programming examples are presented to show the code of the different possible  
 7813 subfunctions using a trace subsystem. All these programs need to include the `<trace.h>` header.  
 7814 In the examples shown, error checking is omitted for more simplicity.

### 7815 Trace Operation Control

7816 These examples show the creation of a trace stream for another process; one which is already  
 7817 trace instrumented. All the default trace stream attributes are used to simplify programming in  
 7818 the first example. The second example shows more possibilities.

### 7819 First Example

```
7820 /* Caution. Error checks omitted */
7821 {
7822     trace_attr_t attr;
7823     pid_t pid = traced_process_pid;
7824     int fd;
7825     trace_id_t trid;
7826     - - - - -
7827     /* Initialize trace stream attributes */
7828     posix_trace_attr_init(&attr);
7829     /* Open a trace log */
7830     fd=open("/tmp/mytracelog",...);
7831     /*
7832      * Create a new trace associated with a log
7833      * and with default attributes
7834      */
```

```

7835     posix_trace_create_withlog(pid, &attr, fd, &trid);
7836     /* Trace attribute structure can now be destroyed */
7837     posix_trace_attr_destroy(&attr);
7838     /* Start of trace event recording */
7839     posix_trace_start(trid);
7840     - - - - -
7841     - - - - -
7842     /* Duration of tracing */
7843     - - - - -
7844     - - - - -
7845     /* Stop and shutdown of trace activity */
7846     posix_trace_shutdown(trid);
7847     - - - - -
7848 }

```

### 7849 **Second Example**

7850 Between the initialization of the trace stream attributes and the creation of the trace stream,  
7851 these trace stream attributes may be modified; see **Trace Stream Attribute Manipulation** (on  
7852 page 3483) for specific programming example. Between the creation and the start of the trace  
7853 stream, the event filter may be set; after the trace stream is started, the event filter may be  
7854 changed. The setting of an event set and the change of a filter is shown in **Create a Trace Event**  
7855 **Type Set and Change the Trace Event Type Filter** (on page 3483).

```

7856 /* Caution. Error checks omitted */
7857 {
7858     trace_attr_t attr;
7859     pid_t pid = traced_process_pid;
7860     int fd;
7861     trace_id_t trid;
7862     - - - - -
7863     /* Initialize trace stream attributes */
7864     posix_trace_attr_init(&attr);
7865     /* Attr default may be changed at this place; see example */
7866     - - - - -
7867     /* Create and open a trace log with R/W user access */
7868     fd=open("/tmp/mytracelog",O_WRONLY|O_CREAT,S_IRUSR|S_IWUSR);
7869     /* Create a new trace associated with a log */
7870     posix_trace_create_withlog(pid, &attr, fd, &trid);
7871     /*
7872      * If the Trace Filter option is supported
7873      * trace event type filter default may be changed at this place;
7874      * see example about changing the trace event type filter
7875      */
7876     posix_trace_start(trid);
7877     - - - - -
7878     /*
7879      * If you have an uninteresting part of the application
7880      * you can stop temporarily.
7881      *
7882      * posix_trace_stop(trid);
7883      * - - - - -

```

```

7884     * - - - - -
7885     * posix_trace_start(trid);
7886     */
7887     - - - - -
7888     /*
7889     * If the Trace Filter option is supported
7890     * the current trace event type filter can be changed
7891     * at any time (see example about how to set
7892     * a trace event type filter
7893     */
7894     - - - - -
7895     /* Stop the recording of trace events */
7896     posix_trace_stop(trid);
7897     /* Shutdown the trace stream */
7898     posix_trace_shutdown(trid);
7899     /*
7900     * Destroy trace stream attributes; attr structure may have
7901     * been used during tracing to fetch the attributes
7902     */
7903     posix_trace_attr_destroy(&attr);
7904     - - - - -
7905 }

```

#### 7906 **Application Instrumentation**

7907 This example shows an instrumented application. The code is included in a block of instructions,  
 7908 perhaps a function from a library. Possibly in an initialization part of the instrumented  
 7909 application, two user trace event names are mapped to two trace event type identifiers  
 7910 (function *posix\_trace\_eventid\_open()*). Then two trace points are programmed.

```

7911 /* Caution. Error checks omitted */
7912 {
7913     trace_event_id_t eventid1, eventid2;
7914     - - - - -
7915     /* Initialization of two trace event type ids */
7916     posix_trace_eventid_open("my_first_event",&eventid1);
7917     posix_trace_eventid_open("my_second_event",&eventid2);
7918     - - - - -
7919     - - - - -
7920     - - - - -
7921     /* Trace point */
7922     posix_trace_event(eventid1,NULL,0);
7923     - - - - -
7924     /* Trace point */
7925     posix_trace_event(eventid2,NULL,0);
7926     - - - - -
7927 }

```

7928        **Trace Analyzer**

7929        This example shows the manipulation of a trace log resulting from the dumping of a completed  
 7930        trace stream. All the default attributes are used to simplify programming, and data associated  
 7931        with a trace event are not shown in the first example. The second example shows more  
 7932        possibilities.

7933        **First Example**

```

7934        /* Caution. Error checks omitted */
7935        {
7936            int fd;
7937            trace_id_t trid;
7938            posix_trace_event_info trace_event;
7939            char trace_event_name[TRACE_EVENT_NAME_MAX];
7940            int return_value;
7941            size_t returndatasize;
7942            int lost_event_number;
7943            - - - - -
7944            /* Open an existing trace log */
7945            fd=open("/tmp/tracelog", O_RDONLY);
7946            /* Open a trace stream on the open log */
7947            posix_trace_open(fd, &trid);
7948            /* Read a trace event */
7949            posix_trace_getnext_event(trid, &trace_event,
7950                                    NULL, 0, &returndatasize,&return_value);
7951            /* Read and print all trace event names out in a loop */
7952            while (return_value == NULL)
7953            {
7954                /*
7955                * Get the name of the trace event associated
7956                * with trid trace ID
7957                */
7958                posix_trace_eventid_get_name(trid, trace_event.event_id,
7959                                            trace_event_name);
7960                /* Print the trace event name out */
7961                printf("%s\n",trace_event_name);
7962                /* Read a trace event */
7963                posix_trace_getnext_event(trid, &trace_event,
7964                                            NULL, 0, &returndatasize,&return_value);
7965            }
7966            /* Close the trace stream */
7967            posix_trace_close(trid);
7968            /* Close the trace log */
7969            close(fd);
7970        }

```

7971 **Second Example**

7972 The complete example includes the two other examples in **Retrieve Information from a Trace**  
 7973 **Log** (on page 3484) and in **Retrieve the List of Trace Event Types Used in a Trace Log** (on page  
 7974 3485). For example, the *maxdatasize* variable is set in **Retrieve the List of Trace Event Types**  
 7975 **Used in a Trace Log** (on page 3485).

```

7976 /* Caution. Error checks omitted */
7977 {
7978     int fd;
7979     trace_id_t trid;
7980     posix_trace_event_info trace_event;
7981     char trace_event_name[TRACE_EVENT_NAME_MAX];
7982     char * data;
7983     size_t maxdatasize=1024, returndatasize;
7984     int return_value;
7985     - - - - -
7986     /* Open an existing trace log */
7987     fd=open("/tmp/tracelog", O_RDONLY);
7988     /* Open a trace stream on the open log */
7989     posix_trace_open( fd, &trid);
7990     /*
7991      * Retrieve information about the trace stream which
7992      * was dumped in this trace log (see example)
7993      */
7994     - - - - -
7995     /* Allocate a buffer for trace event data */
7996     data=(char *)malloc(maxdatasize);
7997     /*
7998      * Retrieve the list of trace event used in this
7999      * trace log (see example)
8000      */
8001     - - - - -
8002     /* Read and print all trace event names and data out in a loop */
8003     while (1)
8004     {
8005     posix_trace_getnext_event(trid, &trace_event,
8006         data, maxdatasize, &returndatasize,&return_value);
8007         if (return_value != NULL) break;
8008         /*
8009          * Get the name of the trace event type associated
8010          * with trid trace ID
8011          */
8012         posix_trace_eventid_get_name(trid, trace_event.event_id,
8013             trace_event_name);
8014         {
8015         int i;
8016
8017         /* Print the trace event name out */
8018         printf("%s: ", trace_event_name);
8019         /* Print the trace event data out */
8020         for (i=0; i<returndatasize, i++) printf("%02.2X",

```

```

8020         (unsigned char)data[i]);
8021     printf("\n");
8022     }
8023 }
8024     /* Close the trace stream */
8025     posix_trace_close(trid);
8026     /* The buffer data is deallocated */
8027     free(data);
8028     /* Now the file can be closed */
8029     close(fd);
8030 }

```

### 8031 **Several Programming Manipulations**

8032 The following examples show some typical sets of operations needed in some contexts.

#### 8033 **Trace Stream Attribute Manipulation**

8034 This example shows the manipulation of a trace stream attribute object in order to change the  
8035 default value provided by a previous *posix\_trace\_attr\_init()* call.

```

8036     /* Caution. Error checks omitted */
8037     {
8038         trace_attr_t attr;
8039         size_t logsize=100000;
8040         - - - - -
8041         /* Initialize trace stream attributes */
8042         posix_trace_attr_init(&attr);
8043         /* Set the trace name in the attributes structure */
8044         posix_trace_attr_setname(&attr, "my_trace");
8045         /* Set the trace full policy */
8046         posix_trace_attr_setstreamfullpolicy(&attr, POSIX_TRACE_LOOP);
8047         /* Set the trace log size */
8048         posix_trace_attr_setlogsize(&attr, logsize);
8049         - - - - -
8050     }

```

#### 8051 **Create a Trace Event Type Set and Change the Trace Event Type Filter**

8052 This example is valid only if the Trace Event Filter option is supported. This example shows the  
8053 manipulation of a trace event type set in order to change the trace event type filter for an existing  
8054 active trace stream, which may be just-created, running, or suspended. Some sets of trace event  
8055 types are well-known, such as the set of trace event types not associated with a process, some  
8056 trace event types are just-built trace event types for this trace stream; one trace event type is the  
8057 predefined trace event error type which is deleted from the trace event type set.

```

8058     /* Caution. Error checks omitted */
8059     {
8060         trace_id_t trid = existing_trace;
8061         trace_event_set_t set;
8062         trace_event_id_t trace_event1, trace_event2;
8063         - - - - -
8064         /* Initialize to an empty set of trace event types */

```

```

8065      /* (not strictly required because posix_trace_event_set_fill() */ |
8066      /* will ignore the prior contents of the event set.) */ |
8067      posix_trace_eventset_emptyset(&set); |
8068      /* |
8069      * Fill the set with all system trace events |
8070      * not associated with a process |
8071      */ |
8072      posix_trace_eventset_fill(&set, POSIX_TRACE_WOPID_EVENTS); |
8073      /* |
8074      * Get the trace event type identifier of the known trace event name |
8075      * my_first_event for the trid trace stream |
8076      */ |
8077      posix_trace_trid_eventid_open(trid, "my_first_event", &trace_event1); |
8078      /* Add the set with this trace event type identifier */ |
8079      posix_trace_eventset_add_event(trace_event1, &set); |
8080      /* |
8081      * Get the trace event type identifier of the known trace event name |
8082      * my_second_event for the trid trace stream |
8083      */ |
8084      posix_trace_trid_eventid_open(trid, "my_second_event", &trace_event2); |
8085      /* Add the set with this trace event type identifier */ |
8086      posix_trace_eventset_add_event(trace_event2, &set); |
8087      - - - - - |
8088      /* Delete the system trace event POSIX_TRACE_ERROR from the set */ |
8089      posix_trace_eventset_del_event(POSIX_TRACE_ERROR, &set); |
8090      - - - - - |
8091      /* Modify the trace stream filter making it equal to the new set */ |
8092      posix_trace_set_filter(trid, &set, POSIX_TRACE_SET_EVENTSET); |
8093      - - - - - |
8094      /* |
8095      * Now trace_event1, trace_event2, and all system trace event types |
8096      * not associated with a process, except for the POSIX_TRACE_ERROR |
8097      * system trace event type, are filtered out of (not recorded in) the |
8098      * existing trace stream. |
8099      */ |
8100      }

```

### 8101 **Retrieve Information from a Trace Log**

8102 This example shows how to extract information from a trace log, the dump of a trace stream.  
8103 This code:

```

8104      • Asks if the trace stream has lost trace events
8105      • Extracts the information about the version of the trace subsystem which generated this trace
8106      log
8107      • Retrieves the maximum size of trace event data; this may be used to dynamically allocate an
8108      array for extracting trace event data from the trace log without overflow
8109      /* Caution. Error checks omitted */
8110      {
8111          struct posix_trace_status_info statusinfo;

```

```

8112     trace_attr_t attr;
8113     trace_id_t trid = existing_trace;
8114     size_t maxdatasize;
8115     char genversion[TRACE_NAME_MAX];
8116     - - - - -
8117     /* Get the trace stream status */
8118     posix_trace_get_status(trid, &statusinfo);
8119     /* Detect an overrun condition */
8120     if (statusinfo.posix_stream_overrun_status == POSIX_TRACE_OVERRUN)
8121         printf("trace events have been lost\n");

8122     /* Get attributes from the trid trace stream */
8123     posix_trace_get_attr(trid, &attr);
8124     /* Get the trace generation version from the attributes */
8125     posix_trace_attr_getgenversion(&attr, genversion);
8126     /* Print the trace generation version out */
8127     printf("Information about Trace Generator:%s\n",genversion);

8128     /* Get the trace event max data size from the attributes */
8129     posix_trace_attr_getmaxdatasize(&attr, &maxdatasize);
8130     /* Print the trace event max data size out */
8131     printf("Maximum size of associated data:%d\n",maxdatasize);
8132     /* Destroy the trace stream attributes */
8133     posix_trace_attr_destroy(&attr);
8134 }

```

### 8135 **Retrieve the List of Trace Event Types Used in a Trace Log**

8136 This example shows the retrieval of a trace stream's trace event type list. This operation may be  
8137 very useful if you are interested only in tracking the type of trace events in a trace log.

```

8138 /* Caution. Error checks omitted */
8139 {
8140     trace_id_t trid = existing_trace;
8141     trace_event_id_t event_id;
8142     char event_name[TRACE_EVENT_NAME_MAX];
8143     int return_value;
8144     - - - - -

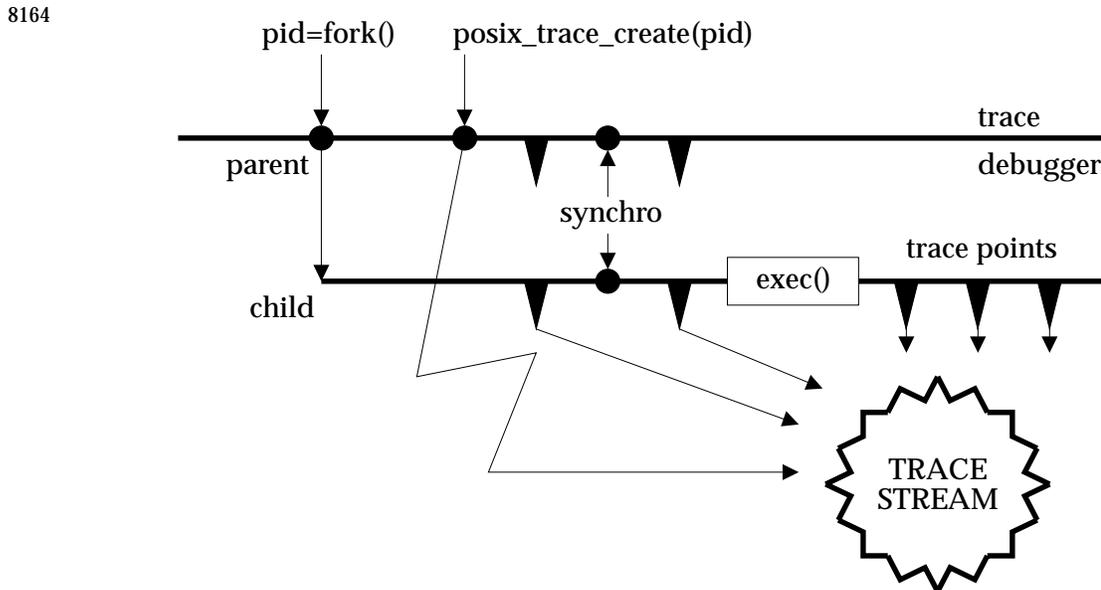
8145     /*
8146      * In a loop print all existing trace event names out
8147      * for the trid trace stream
8148      */
8149     while (1)
8150     {
8151         posix_trace_eventtypelist_getnext_id(trid, &event_id
8152             &return_value);
8153         if (return_value != NULL) break;
8154         /*
8155          * Get the name of the trace event associated
8156          * with trid trace ID
8157          */
8158         posix_trace_eventid_get_name(trid, event_id, event_name);
8159         /* Print the name out */

```

```

8160         printf("%s\n", event_name);
8161     }
8162 }
    
```

8163 *B.2.11.4 Rationale on Trace for Debugging*



8165 **Figure B-5** Trace Another Process

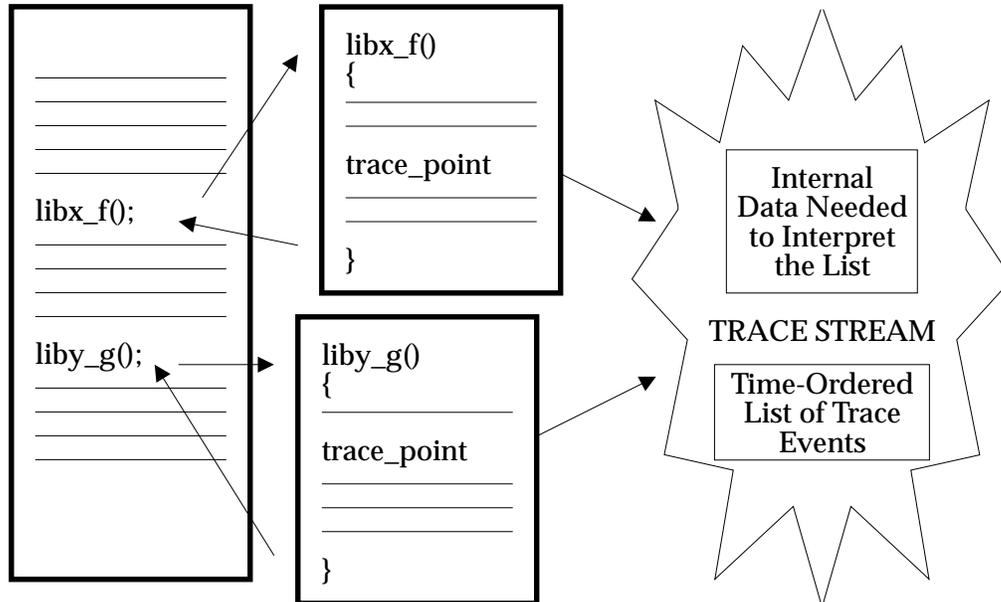
8166 Among the different possibilities offered by the trace interface defined in IEEE Std 1003.1-200x,  
 8167 the debugging of an application is the most interesting one. Typical operations in the controlling  
 8168 debugger process are to filter trace event types, to get trace events from the trace stream, to stop  
 8169 the trace stream when the debugged process is executing uninteresting code, to start the trace  
 8170 stream when some interesting point is reached, and so on. The interface defined in  
 8171 IEEE Std 1003.1-200x should define all the necessary base functions to allow this dynamic debug  
 8172 handling.

8173 Figure B-5 shows an example in which the trace stream is created after the call to the *fork()*  
 8174 function. If the user does not want to lose trace events some synchronization mechanism  
 8175 (represented in the figure) may be needed before calling the *exec()* function, to give the parent a  
 8176 chance to create the trace stream before the child begins the execution of its trace points.

8177 *B.2.11.5 Rationale on Trace Event Type Name Space*

8178 At first, the working group was in favor of the representation of a trace event type by an integer  
 8179 (*event\_name*). It seems that existing practice shows the weakness of such a representation. The  
 8180 collision of trace event types is the main problem that cannot be simply resolved using this sort  
 8181 of representation. Suppose, for example, that a third party designs an instrumented library. The  
 8182 user does not have the source of this library and wants to trace his application which uses in  
 8183 some part the third-party library. There is no means for him to know what are the trace event  
 8184 types used in the instrumented library so he has some chance of duplicating some of them and  
 8185 thus to obtain a contaminated tracing of his application.

8186



8187

**Figure B-6** Trace Name Space Overview: With Third-Party Library

8188 We have requirements to allow program images containing pieces from various vendors to be  
 8189 traced without also requiring those or any other vendors to coordinate their uses of the trace  
 8190 facility, and especially the naming of their various trace event types and trace point IDs. The  
 8191 chosen solution is to provide a very large name space, large enough so that the individual  
 8192 vendors can give their trace types and tracepoint IDs sufficiently long and descriptive names  
 8193 making the occurrence of collisions quite unlikely. The probability of collision is thus made  
 8194 sufficiently low so that the problem may, as a practical matter, be ignored. By requirement, the  
 8195 consequence of collisions will be a slight ambiguity in the trace streams; tracing will continue in  
 8196 spite of collisions and ambiguities. “The show must go on”. The *posix\_prog\_address* member of  
 8197 the **posix\_trace\_event\_info** structure is used to allow trace streams to be unambiguously  
 8198 interpreted, despite the fact that trace event types and trace event names need not be unique.

8199 The *posix\_trace\_eventid\_open()* function is required to allow the instrumented third-party library  
 8200 to get a valid trace event type identifier for its trace event names. This operation is, somehow,  
 8201 an allocation, and the group was aware of proposing some deallocation mechanism which the  
 8202 instrumented application could use to recover the resources used by a trace event type identifier.  
 8203 This would have given the instrumented application the benefit of being capable of reusing a  
 8204 possible minimum set of trace event type identifiers, but also the inconvenience to have,  
 8205 possibly in the same trace stream, one trace event type identifier identifying two different trace  
 8206 event types. After some discussions the group decided to not define such a function which  
 8207 would make this API thicker for little benefit, the user having always the possibility of adding  
 8208 identification information in the data member of the trace event structure.

8209 The set of the trace event type identifiers the controlling process wants to filter out is initialized  
 8210 in the trace mechanism using the function *posix\_trace\_set\_filter()*, setting the arguments  
 8211 according to the definitions explained in *posix\_trace\_set\_filter()*. This operation can be done  
 8212 statically (when the trace is in the STOPPED state) or dynamically (when the trace is in the  
 8213 STARTED state). The preparation of the filter is normally done using the function defined in  
 8214 *posix\_trace\_eventtypelist\_getnext\_id()* and eventually the function  
 8215 *posix\_trace\_eventtypelist\_rewind()* in order to know (before the recording) the list of the potential

8216 set of trace event types that can be recorded. In the case of an active trace stream, this list may  
8217 not be exhaustive. Actually, the target process may not have yet called the function  
8218 *posix\_trace\_eventid\_open()*. But it is a common practice, for a controlling process, to prepare the  
8219 filtering of a future trace stream before its start. Therefore the user must have a way to get the  
8220 trace event type identifier corresponding to a well-known trace event name before its future  
8221 association by the pre-cited function. This is done by calling the *posix\_trace\_trid\_eventid\_open()*  
8222 function, given the trace stream identifier and the trace name, and described hereafter. Because  
8223 this trace event type identifier is associated with a trace stream identifier, where a unique  
8224 process has initialized two or more traces, the implementation is expected to return the same  
8225 trace event type identifier for successive calls to *posix\_trace\_trid\_eventid\_open()* with different  
8226 trace stream identifiers. The *posix\_trace\_eventid\_get\_name()* function is used by the controller  
8227 process to identify, by the name, the trace event type returned by a call to the  
8228 *posix\_trace\_eventtypelist\_getnext\_id()* function.

8229 Afterwards, the set of trace event types is constructed using the functions defined in  
8230 *posix\_trace\_eventset\_empty()*, *posix\_trace\_eventset\_fill()*, *posix\_trace\_eventset\_add()*, and  
8231 *posix\_trace\_eventset\_del()*.

8232 A set of functions is provided devoted to the manipulation of the trace event type identifier and  
8233 names for an active trace stream. All these functions require the trace stream identifier argument  
8234 as the first parameter. The opacity of the trace event type identifier implies that the user cannot  
8235 associate directly its well-known trace event name with the system associated trace event type  
8236 identifier.

8237 The *posix\_trace\_trid\_eventid\_open()* function allows the application to get the system trace event  
8238 type identifier back from the system, given its well-known trace event name. This function is  
8239 useful only when a controlling process needs to specify specific events to be filtered.

8240 The *posix\_trace\_eventid\_get\_name()* function allows the application to obtain a trace event name  
8241 given its trace event type identifier. One possible use of this function is to identify the type of a  
8242 trace event retrieved from the trace stream, and print it. The easiest way to implement this  
8243 requirement, is to use a single trace event type map for all the processes whose maps are  
8244 required to be identical. A more difficult way is to attempt to keep multiple maps identical at  
8245 every call to *posix\_trace\_eventid\_open()* and *posix\_trace\_trid\_eventid\_open()*.

#### 8246 B.2.11.6 Rationale on Trace Events Type Filtering

8247 The most basic rationale for runtime and pre-registration filtering (selection/rejection) of trace  
8248 event types is to prevent choking of the trace collection facility, and/or overloading of the  
8249 computer system. Any worthwhile trace facility can bring even the largest computer to its  
8250 knees. Otherwise, we would record everything, and filter after the fact; it would be much  
8251 simpler, but impractical.

8252 To achieve debugging, measurement, or whatever the purpose of tracing, the filtering of trace  
8253 event types is an important part of trace analysis. Due to the fact that the trace events are put  
8254 into a trace stream and probably logged afterwards into a file, different levels of filtering—that  
8255 is, rejection of trace event types—are possible.

8256 **Filtering of Trace Event Types Before Tracing**

8257 This function, represented by the `posix_trace_set_filter()` function in IEEE Std 1003.1-200x (see  
 8258 `posix_trace_set_filter()`), selects, before or during tracing, the set of trace event types to be filtered  
 8259 out. It should be possible also (as OSF suggested in their ETAP trace specifications) to select the  
 8260 kernel trace event types to be traced in a system-wide fashion. These two functionalities are  
 8261 called the pre-filtering of trace event types.

8262 The restriction on the actual type used for the `trace_event_set_t` type is intended to guarantee  
 8263 that these objects can always be assigned, have their address taken, and be passed by value as  
 8264 parameters. It is not intended that this type be a structure including pointers to other data  
 8265 structures, as that could impact the portability of applications performing such operations. A  
 8266 reasonable implementation could be a structure containing an array of integer types.

8267 **Filtering of Trace Event Types at Runtime**

8268 Using this API, this functionality may be built, a privileged process or a privileged thread can  
 8269 get trace events from the trace stream of another process or thread, and thus specify the type of  
 8270 trace events to record into a file, using methods and interfaces out of the scope of  
 8271 IEEE Std 1003.1-200x. This functionality, called inline filtering of trace event types, is used for  
 8272 runtime analysis of trace streams.

8273 **Post-Mortem Filtering of Trace Event Types**

8274 The word *post-mortem* is used here to indicate that some unanticipated situation occurs during  
 8275 execution that does not permit a pre or inline filtering of trace events and that it is necessary to  
 8276 record all trace event types, to have a chance to discover the problem afterwards. When the  
 8277 program stops, all the trace events recorded previously can be analyzed in order to find the  
 8278 solution. This functionality could be named the post-filtering of trace event types.

8279 **Discussions about Trace Event Type-Filtering**

8280 After long discussions with the parties involved in the process of defining the trace interface, it  
 8281 seems that the sensitivity to the filtering problem is different, but everybody agrees that the level  
 8282 of the overhead introduced during the tracing operation depends on the filtering method  
 8283 elected. If the time that it takes the trace event to be recorded can be neglected, the overhead  
 8284 introduced by the filtering process can be classified as follows:

8285 Pre-filtering      System and process/thread-level overhead

8286 Inline-filtering    Process/thread-level overhead

8287 Post-filtering      No overhead; done offline

8288 The pre-filtering could be named *critical realtime* filtering in the sense that the filtering of trace  
 8289 event type is manageable at the user level so the user can lower to a minimum the filtering  
 8290 overhead at some user selected level of priority for the inline filtering, or delay the filtering to  
 8291 after execution for the post-filtering. The counterpart of this solution is that the size of the trace  
 8292 stream must be sufficient to record all the trace events. The advantage of the pre-filtering is that  
 8293 the utilization of the trace stream is optimized.

8294 Only pre-filtering is defined by IEEE Std 1003.1-200x. However, great care must be taken in  
 8295 specifying pre-filtering, so that it does not impose unacceptable overhead. Moreover, it is  
 8296 necessary to isolate all the functionality relative to the pre-filtering.

8297 The result of this rationale is to define a new option, the Trace Event Filter option, not  
 8298 necessarily implemented in small realtime systems, where system overhead is minimized to the  
 8299 extent possible.

8300 *B.2.11.7 Tracing, pthread API*

8301 The objective to be able to control tracing for individual threads may be in conflict with the  
 8302 efficiency expected in threads with a `contentionscope` attribute of  
 8303 `PTHREAD_SCOPE_PROCESS`. For these threads, context switches from one thread that has  
 8304 tracing enabled to another thread that has tracing disabled may require a kernel call to inform  
 8305 the kernel whether it has to trace system events executed by that thread or not. For this reason, it  
 8306 was proposed that the ability to enable or disable tracing for `PTHREAD_SCOPE_PROCESS`  
 8307 threads be made optional, through the introduction of a Trace Scope Process option. A trace  
 8308 implementation which did not implement the Trace Scope Process option would not honor the  
 8309 tracing-state attribute of a thread with `PTHREAD_SCOPE_PROCESS`; it would, however, honor  
 8310 the tracing-state attribute of a thread with `PTHREAD_SCOPE_SYSTEM`. This proposal was  
 8311 rejected as:

- 8312 1. Removing desired functionality (per-thread trace control)
- 8313 2. Introducing counter-intuitive behavior for the tracing-state attribute
- 8314 3. Mixing logically orthogonal ideas (thread scheduling and thread tracing)
- 8315 [Objective 4]

8316 Finally, to solve this complex issue, this API does not provide `pthread_gettracingstate()`,  
 8317 `pthread_settracingstate()`, `pthread_attr_gettracingstate()`, and `pthread_attr_settracingstate()`  
 8318 interfaces. These interfaces force the thread implementation to add to the weight of the thread  
 8319 and cause a revision of the threads libraries, just to support tracing. Worse yet,  
 8320 `posix_trace_userevent()` must always test this per-thread variable even in the common case where  
 8321 it is not used at all. Per-thread tracing is easy to implement using existing interfaces where  
 8322 necessary; see the following example.

8323 **Example**

```
8324 /* Caution. Error checks omitted */
8325 static pthread_key_t my_key;
8326 static trace_event_id_t my_event_id;
8327 static pthread_once_t my_once = PTHREAD_ONCE_INIT;

8328 void my_init(void)
8329 {
8330     (void) pthread_key_create(&my_key, NULL);
8331     (void) posix_trace_eventid_open("my", &my_event_id);
8332 }

8333 int get_trace_flag(void)
8334 {
8335     pthread_once(&my_once, my_init);
8336     return (pthread_getspecific(my_key) != NULL);
8337 }

8338 void set_trace_flag(int f)
8339 {
8340     pthread_once(&my_once, my_init);
8341     pthread_setspecific(my_key, f? &my_event_id: NULL);
8342 }

8343 fn()
8344 {
8345     if (get_trace_flag())
```

```
8346         posix_trace_event(my_event_id, ...)
8347     }
```

8348 The above example does not implement third-party state setting, but it is also implementable  
8349 with some more work, yet the extra functionality is rarely needed.

8350 Lastly, per-thread tracing works poorly for threads with PTHREAD\_SCOPE\_PROCESS  
8351 contention scope. These “library” threads have minimal interaction with the kernel and would  
8352 have to explicitly set the attributes whenever they are context switched to a new kernel thread in  
8353 order to trace system events. Such state was explicitly avoided in POSIX threads to keep  
8354 PTHREAD\_SCOPE\_PROCESS threads lightweight.

8355 The reason that keeping PTHREAD\_SCOPE\_PROCESS threads lightweight is important is that  
8356 such threads can be used not just for simple multi-processors but also for coroutine style  
8357 programming (such as discrete event simulation) without inventing a new threads paradigm.  
8358 Adding extra runtime cost to thread context switches will make using POSIX threads less  
8359 attractive in these situations.

#### 8360 *B.2.11.8 Rationale on Triggering*

8361 The ability to start or stop tracing based on the occurrence of specific trace event types has been  
8362 proposed as a parallel to similar functionality appearing in logic analyzers. Such triggering, in  
8363 order to be very useful, should be based not only on the trace event type, but on trace event-  
8364 specific data, including tests of user-specified fields for matching or threshold values.

8365 Such a facility is unnecessary where the buffering of the stream is not a constraint, since such  
8366 checks can be performed offline during post-mortem analysis.

8367 For example, a large system could incorporate a daemon utility to collect the trace records from  
8368 memory buffers and spool them to secondary storage for later analysis. In the instances where  
8369 resources are truly limited, such as embedded applications, the application incorporation of  
8370 application code to test the circumstances of a trace event and call the trace point only if needed  
8371 is usually straightforward.

8372 For performance reasons, the *posix\_trace\_event()* function should be implemented using a macro,  
8373 so if the trace is inactive, the trace event point calls are latent code and must cost no more than a  
8374 scalar test.

8375 The API proposed in IEEE Std 1003.1-200x does not include any triggering functionality.

#### 8376 *B.2.11.9 Rationale on Timestamp Clock*

8377 It has been suggested that the tracing mechanism should include the possibility of specifying the  
8378 clock to be used in timestamping the trace events. When application trace events must be  
8379 correlated to remote trace events, such a facility could provide a global time reference not  
8380 available from a local clock. Further, the application may be driven by timers based on a clock  
8381 different from that used for the timestamp, and the correlation of the trace to those untraced  
8382 timer activities could be an important part of the analysis of the application.

8383 However, the tracing mechanism needs to be fast and just the provision of such an option can  
8384 materially affect its performance. Leaving aside the performance costs of reading some clocks,  
8385 this notion is also ill-defined when kernel trace events are to be traced by two applications  
8386 making use of different tracing clocks. This can even happen within a single application where  
8387 different parts of the application are served by different clocks. Another complication can occur  
8388 when a clock is maintained strictly at the user level and is unavailable at the kernel level.

8389 It is felt that the benefits of a selectable trace clock do not match its costs. Applications that wish  
8390 to correlate clocks other than the default tracing clock can include trace events with sample

8391 values of those other clocks, allowing correlation of timestamps from the various independent  
8392 clocks. In any case, such a technique would be required when applications are sensitive to  
8393 multiple clocks.

#### 8394 *B.2.11.10 Rationale on Different Overrun Conditions*

8395 The analysis of the dynamic behavior of the trace mechanism shows that different overrun  
8396 conditions may occur. The API must provide a means to manage such conditions in a portable  
8397 way.

#### 8398 **Overrun in Trace Streams Initialized with POSIX\_TRACE\_LOOP Policy**

8399 In this case, the user of the trace mechanism is interested in using the trace stream with  
8400 POSIX\_TRACE\_LOOP policy to record trace events continuously, but ideally without losing any  
8401 trace events. The online analyzer process must get the trace events at a mean speed equivalent to  
8402 the recording speed. Should the trace stream become full, a trace stream overrun occurs. This  
8403 condition is detected by getting the status of the active trace stream (function *posix\_trace\_get\_status()*)  
8404 and looking at the member *posix\_stream\_overrun\_status* of the read **posix\_stream\_status**  
8405 structure. In addition, two predefined trace event types are defined:

- 8406 1. The beginning of a trace overflow, to locate the beginning of an overflow when reading a  
8407 trace stream
- 8408 2. The end of a trace overflow, to locate the end of an overflow, when reading a trace stream

8409 As a timestamp is associated with these predefined trace events, it is possible to know the  
8410 duration of the overflow.

#### 8411 **Overrun in Dumping Trace Streams into Trace Logs**

8412 The user lets the trace mechanism dump the trace stream initialized with  
8413 POSIX\_TRACE\_FLUSH policy automatically into a trace log. If the dump operation is slower  
8414 than the recording of trace events, the trace stream can overrun. This condition is detected by  
8415 getting the status of the active trace stream (function *posix\_trace\_get\_status()*) and looking at the  
8416 member *posix\_log\_overrun\_status* of the read **posix\_stream\_status** structure. This overrun  
8417 indicates that the trace mechanism is not able to operate in this mode at this speed. It is the  
8418 responsibility of the user to modify one of the trace parameters (the stream size or the trace  
8419 event type filter, for instance) to avoid such overrun conditions, if overruns are to be prevented.  
8420 The same already predefined trace event types (see **Overrun in Trace Streams Initialized with**  
8421 **POSIX\_TRACE\_LOOP Policy**) are used to detect and to know the duration of an overflow.

#### 8422 **Reading an Active Trace Stream**

8423 Although this trace API allows one to read an active trace stream with log while it is tracing, this  
8424 feature can lead to false overflow origin interpretation: the trace log or the reader of the trace  
8425 stream. Reading from an active trace stream with log is thus non-portable, and has been left  
8426 unspecified.

8427 **B.2.12 Data Types**

8428 The requirement that additional types defined in this section end in “\_t” was prompted by the  
 8429 problem of name space pollution. It is difficult to define a type (where that type is not one  
 8430 defined by IEEE Std 1003.1-200x) in one header file and use it in another without adding symbols  
 8431 to the name space of the program. To allow implementors to provide their own types, all  
 8432 conforming applications are required to avoid symbols ending in “\_t”, which permits the  
 8433 implementor to provide additional types. Because a major use of types is in the definition of  
 8434 structure members, which can (and in many cases must) be added to the structures defined in  
 8435 IEEE Std 1003.1-200x, the need for additional types is compelling.

8436 The types, such as **ushort** and **ulong**, which are in common usage, are not defined in  
 8437 IEEE Std 1003.1-200x (although **ushort\_t** would be permitted as an extension). They can be  
 8438 added to `<sys/types.h>` using a feature test macro (see Section B.2.2.1 (on page 3375)). A  
 8439 suggested symbol for these is `_SYSIII`. Similarly, the types like **u\_short** would probably be best  
 8440 controlled by `_BSD`.

8441 Some of these symbols may appear in other headers; see Section B.2.2.2 (on page 3376).

8442 **dev\_t** This type may be made large enough to accommodate host-locality considerations  
 8443 of networked systems.

8444 This type must be arithmetic. Earlier proposals allowed this to be non-arithmetic  
 8445 (such as a structure) and provided a `samefile()` function for comparison.

8446 **gid\_t** Some implementations had separated **gid\_t** from **uid\_t** before POSIX.1 was  
 8447 completed. It would be difficult for them to coalesce them when it was  
 8448 unnecessary. Additionally, it is quite possible that user IDs might be different from  
 8449 group IDs because the user ID might wish to span a heterogeneous network,  
 8450 where the group ID might not.

8451 For current implementations, the cost of having a separate **gid\_t** will be only  
 8452 lexical.

8453 **mode\_t** This type was chosen so that implementations could choose the appropriate  
 8454 integral type, and for compatibility with the ISO C standard. 4.3 BSD uses  
 8455 **unsigned short** and the SVID uses **ushort**, which is the same. Historically, only the  
 8456 low-order sixteen bits are significant.

8457 **nlink\_t** This type was introduced in place of **short** for `st_nlink` (see the `<sys/stat.h>` header)  
 8458 in response to an objection that **short** was too small.

8459 **off\_t** This type is used only in `lseek()`, `fcntl()`, and `<sys/stat.h>`. Many implementations  
 8460 would have difficulties if it were defined as anything other than **long**. Requiring  
 8461 an integral type limits the capabilities of `lseek()` to four gigabytes. The ISO C  
 8462 standard supplies routines that use larger types; see `fgetpos()` and `fsetpos()`. XSI-  
 8463 conformant systems provide the `fseeko()` and `lseeko()` functions that use larger  
 8464 types.

8465 **pid\_t** The inclusion of this symbol was controversial because it is tied to the issue of the  
 8466 representation of a process ID as a number. From the point of view of a  
 8467 conforming application, process IDs should be “magic cookies”<sup>1</sup> that are produced

8468 \_\_\_\_\_

8469 1. An historical term meaning: “An opaque object, or token, of determinate size, whose significance is known only to the entity  
 8470 which created it. An entity receiving such a token from the generating entity may only make such use of the ‘cookie’ as is defined  
 8471 and permitted by the supplying entity.”

8472 by calls such as *fork()*, used by calls such as *waitpid()* or *kill()*, and not otherwise  
8473 analyzed (except that the sign is used as a flag for certain operations).

8474 The concept of a {PID\_MAX} value interacted with this in early proposals. Treating  
8475 process IDs as an opaque type both removes the requirement for {PID\_MAX} and  
8476 allows systems to be more flexible in providing process IDs that span a large range  
8477 of values, or a small one.

8478 Since the values in **uid\_t**, **gid\_t**, and **pid\_t** will be numbers generally, and  
8479 potentially both large in magnitude and sparse, applications that are based on  
8480 arrays of objects of this type are unlikely to be fully portable in any case. Solutions  
8481 that treat them as magic cookies will be portable.

8482 {CHILD\_MAX} precludes the possibility of a “toy implementation”, where there  
8483 would only be one process.

8484 **ssize\_t** This is intended to be a signed analog of **size\_t**. The wording is such that an  
8485 implementation may either choose to use a longer type or simply to use the signed  
8486 version of the type that underlies **size\_t**. All functions that return **ssize\_t** (*read()*  
8487 and *write()*) describe as “implementation-defined” the result of an input exceeding  
8488 {SSIZE\_MAX}. It is recognized that some implementations might have **ints** that  
8489 are smaller than **size\_t**. A conforming application would be constrained not to  
8490 perform I/O in pieces larger than {SSIZE\_MAX}, but a conforming application  
8491 using extensions would be able to use the full range if the implementation  
8492 provided an extended range, while still having a single type-compatible interface.

8493 The symbols **size\_t** and **ssize\_t** are also required in **<unistd.h>** to minimize the  
8494 changes needed for calls to *read()* and *write()*. Implementors are reminded that it  
8495 must be possible to include both **<sys/types.h>** and **<unistd.h>** in the same  
8496 program (in either order) without error.

8497 **uid\_t** Before the addition of this type, the data types used to represent these values  
8498 varied throughout early proposals. The **<sys/stat.h>** header defined these values as  
8499 type **short**, the **<passwd.h>** file (now **<pwd.h>** and **<grp.h>**) used an **int**, and  
8500 *getuid()* returned an **int**. In response to a strong objection to the inconsistent  
8501 definitions, all the types were switched to **uid\_t**.

8502 In practice, those historical implementations that use varying types of this sort can  
8503 typedef **uid\_t** to **short** with no serious consequences.

8504 The problem associated with this change concerns object compatibility after  
8505 structure size changes. Since most implementations will define **uid\_t** as a short, the  
8506 only substantive change will be a reduction in the size of the **passwd** structure.  
8507 Consequently, implementations with an overriding concern for object  
8508 compatibility can pad the structure back to its current size. For that reason, this  
8509 problem was not considered critical enough to warrant the addition of a separate  
8510 type to POSIX.1.

8511 The types **uid\_t** and **gid\_t** are magic cookies. There is no {UID\_MAX} defined by  
8512 POSIX.1, and no structure imposed on **uid\_t** and **gid\_t** other than that they be  
8513 positive arithmetic types. (In fact, they could be **unsigned char**.) There is no  
8514 maximum or minimum specified for the number of distinct user or group IDs.

8515 **B.3 System Interfaces**

8516 See the RATIONALE sections on the individual reference pages.

8517 **B.3.1 Examples for Spawn**8518 The following long examples are provided in the Rationale (Informative) volume of  
8519 IEEE Std 1003.1-200x as a supplement to the reference page for *spawn()*.8520 **Example Library Implementation of Spawn**8521 The *posix\_spawn()* or *posix\_spawnnp()* functions provide the following:

- 8522 • Simply start a process executing a process image. This is the simplest application for process  
8523 creation, and it may cover most executions of POSIX *fork()*.
- 8524 • Support I/O redirection, including pipes.
- 8525 • Run the child under a user and group ID in the domain of the parent.
- 8526 • Run the child at any priority in the domain of the parent.

8527 The *posix\_spawn()* or *posix\_spawnnp()* functions do not cover every possible use of the *fork()*  
8528 function, but they do span the common applications: typical use by a shell and a login utility.8529 The price for an application is that before it calls *posix\_spawn()* or *posix\_spawnnp()*, the parent  
8530 must adjust to a state that *posix\_spawn()* or *posix\_spawnnp()* can map to the desired state for the  
8531 child. Environment changes require the parent to save some of its state and restore it afterwards.  
8532 The effective behavior of a successful invocation of *posix\_spawn()* is as if the operation were  
8533 implemented with POSIX operations as follows:

```

8534 #include <sys/types.h>
8535 #include <stdlib.h>
8536 #include <stdio.h>
8537 #include <unistd.h>
8538 #include <sched.h>
8539 #include <fcntl.h>
8540 #include <signal.h>
8541 #include <errno.h>
8542 #include <string.h>
8543 #include <signal.h>

8544 /* #include <spawn.h>*/
8545 /*****
8546 /* Things that could be defined in spawn.h */
8547 /*****
8548 typedef struct
8549     {
8550     short posix_attr_flags;
8551     #define POSIX_SPAWN_SETPGROUP          0x1
8552     #define POSIX_SPAWN_SETSIGMASK       0x2
8553     #define POSIX_SPAWN_SETSIGDEF        0x4
8554     #define POSIX_SPAWN_SETSCHEDULER     0x8
8555     #define POSIX_SPAWN_SETSCHEDPARAM    0x10
8556     #define POSIX_SPAWN_RESETIDS         0x20
8557     pid_t posix_attr_pgroup;
8558     sigset_t posix_attr_sigmask;
8559     sigset_t posix_attr_sigdefault;

```

```

8560         int posix_attr_schedpolicy;
8561         struct sched_param posix_attr_schedparam;
8562     } posix_spawnattr_t;

8563     typedef char *posix_spawn_file_actions_t;

8564     int posix_spawn_file_actions_init(
8565         posix_spawn_file_actions_t *file_actions);
8566     int posix_spawn_file_actions_destroy(
8567         posix_spawn_file_actions_t *file_actions);
8568     int posix_spawn_file_actions_addclose(
8569         posix_spawn_file_actions_t *file_actions, int fildes);
8570     int posix_spawn_file_actions_adddup2(
8571         posix_spawn_file_actions_t *file_actions, int fildes,
8572         int newfildes);
8573     int posix_spawn_file_actions_addopen(
8574         posix_spawn_file_actions_t *file_actions, int fildes,
8575         const char *path, int oflag, mode_t mode);
8576     int posix_spawnattr_init(posix_spawnattr_t *attr);
8577     int posix_spawnattr_destroy(posix_spawnattr_t *attr);
8578     int posix_spawnattr_getflags(const posix_spawnattr_t *attr, short *lags);
8579     int posix_spawnattr_setflags(posix_spawnattr_t *attr, short flags);
8580     int posix_spawnattr_getpgroup(const posix_spawnattr_t *attr,
8581         pid_t *pgroup);
8582     int posix_spawnattr_setpgroup(posix_spawnattr_t *attr, pid_t pgroup);
8583     int posix_spawnattr_getschedpolicy(const posix_spawnattr_t *attr,
8584         int *schedpolicy);
8585     int posix_spawnattr_setschedpolicy(posix_spawnattr_t *attr,
8586         int schedpolicy);
8587     int posix_spawnattr_getschedparam(const posix_spawnattr_t *attr,
8588         struct sched_param *schedparam);
8589     int posix_spawnattr_setschedparam(posix_spawnattr_t *attr,
8590         const struct sched_param *schedparam);
8591     int posix_spawnattr_getsigmask(const posix_spawnattr_t *attr,
8592         sigset_t *sigmask);
8593     int posix_spawnattr_setsigmask(posix_spawnattr_t *attr,
8594         const sigset_t *sigmask);
8595     int posix_spawnattr_getdefault(const posix_spawnattr_t *attr,
8596         sigset_t *sigdefault);
8597     int posix_spawnattr_setsigdefault(posix_spawnattr_t *attr,
8598         const sigset_t *sigdefault);
8599     int posix_spawn(pid_t *pid, const char *path,
8600         const posix_spawn_file_actions_t *file_actions,
8601         const posix_spawnattr_t *attrp, char * const argv[],
8602         char * const envp[]);
8603     int posix_spawnnp(pid_t *pid, const char *file,
8604         const posix_spawn_file_actions_t *file_actions,
8605         const posix_spawnattr_t *attrp, char * const argv[],
8606         char * const envp[]);

8607     /*****
8608     /* Example posix_spawn() library routine */
8609     /*****
8610     int posix_spawn(pid_t *pid,

```

```
8611     const char *path,
8612     const posix_spawn_file_actions_t *file_actions,
8613     const posix_spawnattr_t *attrp,
8614     char * const argv[],
8615     char * const envp[])
8616 {
8617     /* Create process */
8618     if((*pid=fork()) == (pid_t)0)
8619     {
8620         /* This is the child process */
8621         /* Worry about process group */
8622         if(attrp->posix_attr_flags & POSIX_SPAWN_SETPGROUP)
8623         {
8624             /* Override inherited process group */
8625             if(setpgid(0, attrp->posix_attr_pgroup) != 0)
8626             {
8627                 /* Failed */
8628                 exit(127);
8629             }
8630         }
8631
8632         /* Worry about process signal mask */
8633         if(attrp->posix_attr_flags & POSIX_SPAWN_SETSIGMASK)
8634         {
8635             /* Set the signal mask (can't fail) */
8636             sigprocmask(SIG_SETMASK , &attrp->posix_attr_sigmask,
8637                 NULL);
8638         }
8639
8640         /* Worry about resetting effective user and group IDs */
8641         if(attrp->posix_attr_flags & POSIX_SPAWN_RESETEUID)
8642         {
8643             /* None of these can fail for this case. */
8644             setuid(getuid());
8645             setgid(getgid());
8646         }
8647
8648         /* Worry about defaulted signals */
8649         if(attrp->posix_attr_flags & POSIX_SPAWN_SETSIGDEF)
8650         {
8651             struct sigaction deflt;
8652             sigset_t all_signals;
8653
8654             int s;
8655
8656             /* Construct default signal action */
8657             deflt.sa_handler = SIG_DFL;
8658             deflt.sa_flags = 0;
8659
8660             /* Construct the set of all signals */
8661             sigfillset(&all_signals);
8662
8663             /* Loop for all signals */
8664             for(s=0; sigismember(&all_signals,s); s++)
8665             {
8666                 /* Signal to be defaulted? */
```

```

8660         if(sigismember(&attrp->posix_attr_sigdefault,s))
8661             {
8662                 /* Yes; default this signal */
8663                 if(sigaction(s, &deflt, NULL) == -1)
8664                     {
8665                         /* Failed */
8666                         exit(127);
8667                     }
8668             }
8669     }
8670 }

8671 /* Worry about the fds if we are to map them */
8672 if(file_actions != NULL)
8673     {
8674     /* Loop for all actions in object file_actions */
8675     /*(implementation dives beneath abstraction)*/
8676     char *p = *file_actions;
8677     while(*p != ' ')
8678         {
8679             if(strncmp(p,"close(",6) == 0)
8680                 {
8681                     int fd;
8682                     if(sscanf(p+6,"%d",&fd) != 1)
8683                         {
8684                             exit(127);
8685                         }
8686                     if(close(fd) == -1) exit(127);
8687                 }
8688             else if(strncmp(p,"dup2(",5) == 0)
8689                 {
8690                     int fd,newfd;
8691                     if(sscanf(p+5,"%d,%d",&fd,&newfd) != 2)
8692                         {
8693                             exit(127);
8694                         }
8695                     if(dup2(fd, newfd) == -1) exit(127);
8696                 }
8697             else if(strncmp(p,"open(",5) == 0)
8698                 {
8699                     int fd,oflag;
8700                     mode_t mode;
8701                     int tempfd;
8702                     char path[1000]; /* Should be dynamic */
8703                     char *q;
8704                     if(sscanf(p+5,"%d",&fd) != 1)
8705                         {
8706                             exit(127);
8707                         }
8708                     p = strchr(p, ' ') + 1;
8709                     q = strchr(p, '*');
8710                     if(q == NULL) exit(127);
8711                     strncpy(path, p, q-p);

```

```

8712         path[q-p] = ' ';
8713     if(sscanf(q+1,"%o,%o",&oflag,&mode)!=2)
8714     {
8715         exit(127);
8716     }
8717     if(close(fd) == -1)
8718     {
8719         if(errno != EBADF) exit(127);
8720     }
8721     tempfd = open(path, oflag, mode);
8722     if(tempfd == -1) exit(127);
8723     if(tempfd != fd)
8724     {
8725         if(dup2(tempfd,fd) == -1)
8726         {
8727             exit(127);
8728         }
8729         if(close(tempfd) == -1)
8730         {
8731             exit(127);
8732         }
8733     }
8734     }
8735     else
8736     {
8737         exit(127);
8738     }
8739     p = strchr(p, ' ') + 1;
8740 }
8741 }
8742 /* Worry about setting new scheduling policy and parameters */
8743 if(attrp->posix_attr_flags & POSIX_SPAWN_SETSCHEDULER)
8744 {
8745     if(sched_setscheduler(0, attrp->posix_attr_schedpolicy,
8746         &attrp->posix_attr_schedparam) == -1)
8747     {
8748         exit(127);
8749     }
8750 }
8751 /* Worry about setting only new scheduling parameters */
8752 if(attrp->posix_attr_flags & POSIX_SPAWN_SETSCHEDPARAM)
8753 {
8754     if(sched_setparam(0, &attrp->posix_attr_schedparam)==-1)
8755     {
8756         exit(127);
8757     }
8758 }
8759 /* Now execute the program at path */
8760 /* Any fd that still has FD_CLOEXEC set will be closed */
8761 execve(path, argv, envp);
8762 exit(127); /* exec failed */

```

```

8763     }
8764     else
8765     {
8766         /* This is the parent (calling) process */
8767         if(*pid == (pid_t)-1) return errno;
8768         return 0;
8769     }
8770 }

8771 /*****
8772 /* Here is a crude but effective implementation of the */
8773 /* file action object operators which store actions as */
8774 /* concatenated token separated strings.          */
8775 /*****
8776 /* Create object with no actions. */
8777 int posix_spawn_file_actions_init(
8778     posix_spawn_file_actions_t *file_actions)
8779 {
8780     *file_actions = malloc(sizeof(char));
8781     if(*file_actions == NULL) return ENOMEM;
8782     strcpy(*file_actions, "");
8783     return 0;
8784 }

8785 /* Free object storage and make invalid. */
8786 int posix_spawn_file_actions_destroy(
8787     posix_spawn_file_actions_t *file_actions)
8788 {
8789     free(*file_actions);
8790     *file_actions = NULL;
8791     return 0;
8792 }

8793 /* Add a new action string to object. */
8794 static int add_to_file_actions(
8795     posix_spawn_file_actions_t *file_actions,
8796     char *new_action)
8797 {
8798     *file_actions = realloc
8799         (*file_actions, strlen(*file_actions)+strlen(new_action)+1);
8800     if(*file_actions == NULL) return ENOMEM;
8801     strcat(*file_actions, new_action);
8802     return 0;
8803 }

8804 /* Add a close action to object. */
8805 int posix_spawn_file_actions_addclose(
8806     posix_spawn_file_actions_t *file_actions, int fildes)
8807 {
8808     char temp[100];
8809     sprintf(temp, "close(%d)", fildes);
8810     return add_to_file_actions(file_actions, temp);
8811 }

```

```

8812     /* Add a dup2 action to object. */
8813     int posix_spawn_file_actions_adddup2(
8814         posix_spawn_file_actions_t *file_actions, int fildes,
8815         int newfildes)
8816     {
8817         char temp[100];
8818         sprintf(temp, "dup2(%d,%d)", fildes, newfildes);
8819         return add_to_file_actions(file_actions, temp);
8820     }

8821     /* Add an open action to object. */
8822     int posix_spawn_file_actions_addopen(
8823         posix_spawn_file_actions_t *file_actions, int fildes,
8824         const char *path, int oflag, mode_t mode)
8825     {
8826         char temp[100];
8827         sprintf(temp, "open(%d,%s*%o,%o)", fildes, path, oflag, mode);
8828         return add_to_file_actions(file_actions, temp);
8829     }

8830     /******
8831     /* Here is a crude but effective implementation of the */
8832     /* spawn attributes object functions which manipulate */
8833     /* the individual attributes. */
8834     /******
8835     /* Initialize object with default values. */
8836     int posix_spawnattr_init(
8837         posix_spawnattr_t *attr)
8838     {
8839         attr->posix_attr_flags=0;
8840         attr->posix_attr_pgroup=0;
8841         /* Default value of signal mask is the parent's signal mask; */
8842         /* other values are also allowed */
8843         sigprocmask(0,NULL,&attr->posix_attr_sigmask);
8844         sigemptyset(&attr->posix_attr_sigdefault);
8845         /* Default values of scheduling attr inherited from the parent; */
8846         /* other values are also allowed */
8847         attr->posix_attr_schedpolicy=sched_getscheduler(0);
8848         sched_getparam(0,&attr->posix_attr_schedparam);
8849         return 0;
8850     }

8851     int posix_spawnattr_destroy(posix_spawnattr_t *attr)
8852     {
8853         /* No action needed */
8854         return 0;
8855     }

8856     int posix_spawnattr_getflags(const posix_spawnattr_t *attr,
8857         short *flags)
8858     {
8859         *flags=attr->posix_attr_flags;
8860         return 0;
8861     }

```

```
8862     int posix_spawnattr_setflags(posix_spawnattr_t *attr, short flags)
8863     {
8864         attr->posix_attr_flags=flags;
8865         return 0;
8866     }

8867     int posix_spawnattr_getpgroup(const posix_spawnattr_t *attr,
8868         pid_t *pgroup)
8869     {
8870         *pgroup=attr->posix_attr_pgroup;
8871         return 0;
8872     }

8873     int posix_spawnattr_setpgroup(posix_spawnattr_t *attr, pid_t pgroup)
8874     {
8875         attr->posix_attr_pgroup=pgroup;
8876         return 0;
8877     }

8878     int posix_spawnattr_getschedpolicy(const posix_spawnattr_t *attr,
8879         int *schedpolicy)
8880     {
8881         *schedpolicy=attr->posix_attr_schedpolicy;
8882         return 0;
8883     }

8884     int posix_spawnattr_setschedpolicy(posix_spawnattr_t *attr,
8885         int schedpolicy)
8886     {
8887         attr->posix_attr_schedpolicy=schedpolicy;
8888         return 0;
8889     }

8890     int posix_spawnattr_getschedparam(const posix_spawnattr_t *attr,
8891         struct sched_param *schedparam)
8892     {
8893         *schedparam=attr->posix_attr_schedparam;
8894         return 0;
8895     }

8896     int posix_spawnattr_setschedparam(posix_spawnattr_t *attr,
8897         const struct sched_param *schedparam)
8898     {
8899         attr->posix_attr_schedparam=*schedparam;
8900         return 0;
8901     }

8902     int posix_spawnattr_getsigmask(const posix_spawnattr_t *attr,
8903         sigset_t *sigmask)
8904     {
8905         *sigmask=attr->posix_attr_sigmask;
8906         return 0;
8907     }

8908     int posix_spawnattr_setsigmask(posix_spawnattr_t *attr,
8909         const sigset_t *sigmask)
```

```

8910     {
8911     attr->posix_attr_sigmask=*sigmask;
8912     return 0;
8913     }

8914     int posix_spawnattr_getsigdefault(const posix_spawnattr_t *attr,
8915     sigset_t *sigdefault)
8916     {
8917     *sigdefault=attr->posix_attr_sigdefault;
8918     return 0;
8919     }

8920     int posix_spawnattr_setsigdefault(posix_spawnattr_t *attr,
8921     const sigset_t *sigdefault)
8922     {
8923     attr->posix_attr_sigdefault=*sigdefault;
8924     return 0;
8925     }

```

### 8926 **I/O Redirection with Spawn**

8927 I/O redirection with *posix\_spawn()* or *posix\_spawnp()* is accomplished by crafting a *file\_actions*  
8928 argument to effect the desired redirection. Such a redirection follows the general outline of the  
8929 following example:

```

8930     /* To redirect new standard output (fd 1) to a file, */
8931     /* and redirect new standard input (fd 0) from my fd socket_pair[1], */
8932     /* and close my fd socket_pair[0] in the new process. */
8933     posix_spawn_file_actions_t file_actions;
8934     posix_spawn_file_actions_init(&file_actions);
8935     posix_spawn_file_actions_addopen(&file_actions, 1, "newout", ...);
8936     posix_spawn_file_actions_dup2(&file_actions, socket_pair[1], 0);
8937     posix_spawn_file_actions_close(&file_actions, socket_pair[0]);
8938     posix_spawn_file_actions_close(&file_actions, socket_pair[1]);
8939     posix_spawn(..., &file_actions, ...);
8940     posix_spawn_file_actions_destroy(&file_actions);

```

### 8941 **Spawning a Process Under a New User ID**

8942 Spawning a process under a new user ID follows the outline shown in the following example:

```

8943     Save = getuid();
8944     setuid(newid);
8945     posix_spawn(...);
8946     setuid(Save);

```



8948 / *Rationale (Informative)*

8949 **Part C:**

8950 **Shell and Utilities**

8951 *The Open Group*

8952 *The Institute of Electrical and Electronics Engineers, Inc.*



**Rationale for Shell and Utilities**

8953

8954 **C.1 Introduction**8955 **C.1.1 Scope**

8956 Refer to Section A.1.1 (on page 3293).

8957 **C.1.2 Conformance**

8958 Refer to Section A.2 (on page 3299).

8959 **C.1.3 Normative References**

8960 There is no additional rationale provided for this section. |

8961 **C.1.4 Change History** |8962 The change history is provided as an informative section, to track changes from previous issues |  
8963 of IEEE Std 1003.1-200x. |8964 The following sections describe changes made to the Shell and Utilities volume of |  
8965 IEEE Std 1003.1-200x since Issue 5 of the base document. The CHANGE HISTORY section for |  
8966 each utility describes technical changes made to that utility since Issue 5. Changes between |  
8967 earlier issues of the base document and Issue 5 are not included. |8968 The change history between Issue 5 and Issue 6 also lists the changes since the |  
8969 ISO POSIX-2: 1993 standard. |8970 **Changes from Issue 5 to Issue 6 (IEEE Std 1003.1-200x)** |8971 The following list summarizes the major changes that were made in the Shell and Utilities |  
8972 volume of IEEE Std 1003.1-200x from Issue 5 to Issue 6: |8973 • This volume of IEEE Std 1003.1-200x is extensively revised so it can be both an IEEE POSIX |  
8974 Standard and an Open Group Technical Standard. |

8975 • The terminology has been reworked to meet the style requirements. |

8976 • Shading notation and margin codes are introduced for identification of options within the |  
8977 volume. |8978 • This volume of IEEE Std 1003.1-200x is updated to mandate support of FIPS 151-2. The |  
8979 following changes were made: |8980 — Support is mandated for the capabilities associated with the following symbolic |  
8981 constants: |8982 `_POSIX_CHOWN_RESTRICTED` |8983 `_POSIX_JOB_CONTROL` |8984 `_POSIX_SAVED_IDS` |8985 — In the environment for the login shell, the environment variables *LOGNAME* and *HOME* |  
8986 shall be defined and have the properties described in the Base Definitions volume of |

- 8987 IEEE Std 1003.1-200x, Chapter 7, Locale.
- 8988 • This volume of IEEE Std 1003.1-200x is updated to align with some features of the Single
  - 8989 UNIX Specification.
  - 8990 • A new section on Utility Limits is added.
  - 8991 • A section on the Relationships to Other Documents is added.
  - 8992 • Concepts and definitions have been moved to a separate volume.
  - 8993 • A RATIONALE section is added to each reference page.
  - 8994 • The *c99* utility is added as a replacement for *c89*, which is withdrawn in this issue.
  - 8995 • IEEE Std 1003.2d-1994 is incorporated, adding the *qalter*, *qdel*, *qhold*, *qmove*, *qmsg*, *qrerun*, *qrls*,
  - 8996 *qselect*, *qsig*, *qstat*, and *qsub* utilities.
  - 8997 • IEEE P1003.2b draft standard is incorporated, making extensive updates and adding the *iconv*
  - 8998 utility.
  - 8999 • IEEE PASC Interpretations are applied.
  - 9000 • The Open Groups corrigenda and resolutions are applied.

### 9001 **New Features in Issue 6**

9002 The following table lists the new utilities introduced since the ISO POSIX-2:1993 standard (as  
9003 modified by IEEE Std 1003.2d-1994). These are all part of the XSI extension.

9004

9005

9006

9007

9008

9009

9010

9011

New Utilities in Issue 6				
<i>admin</i>	<i>fuser</i>	<i>link</i>	<i>tsort</i>	<i>uustat</i>
<i>cal</i>	<i>genccat</i>	<i>m4</i>	<i>ulimit</i>	<i>uux</i>
<i>cflow</i>	<i>get</i>	<i>nl</i>	<i>uncompress</i>	<i>val</i>
<i>compress</i>	<i>hash</i>	<i>pr</i>	<i>unget</i>	<i>what</i>
<i>cxref</i>	<i>ipcrm</i>	<i>sact</i>	<i>unlink</i>	<i>zcat</i>
<i>delta</i>	<i>ipcs</i>	<i>sccs</i>	<i>uucp</i>	

### 9012 **C.1.5 Terminology**

9013 Refer to Section A.1.4 (on page 3295).

### 9014 **C.1.6 Definitions**

9015 Refer to Section A.3 (on page 3302).

### 9016 **C.1.7 Relationship to Other Documents**

#### 9017 *C.1.7.1 System Interfaces*

9018 It has been pointed out that the Shell and Utilities volume of IEEE Std 1003.1-200x assumes that  
9019 a great deal of functionality from the System Interfaces volume of IEEE Std 1003.1-200x is  
9020 present, but never states exactly how much (and strictly does not need to since both are  
9021 mandated on a conforming system). This section is an attempt to clarify the assumptions.

9022 **C.1.7.2** *Concepts Derived from the ISO C Standard*

9023 This section was introduced to address the issue that there was insufficient detail presented by  
 9024 such utilities as *awk* or *sh* about their procedural control statements and their methods of  
 9025 performing arithmetic functions.

9026 The ISO C standard was selected as a model because most historical implementations of the  
 9027 standard utilities were written in C. Thus, it was more likely that they would act in the desired  
 9028 manner without modification.

9029 Using the ISO C standard is primarily a notational convenience so that the many procedural  
 9030 languages in the Shell and Utilities volume of IEEE Std 1003.1-200x would not have to be  
 9031 rigorously described in every aspect. Its selection does not require that the standard utilities be  
 9032 written in Standard C; they could be written in Common Usage C, Ada, Pascal, assembler  
 9033 language, or anything else.

9034 The sizes of the various numeric values refer to C-language data types that are allowed to be  
 9035 different sizes by the ISO C standard. Thus, like a C-language application, a shell application  
 9036 cannot rely on their exact size. However, it can rely on their minimum sizes expressed in the  
 9037 ISO C standard, such as {LONG\_MAX} for a **long** type.

9038 **C.1.8** **Portability**

9039 Refer to Section A.1.5 (on page 3298).

9040 **C.1.8.1** *Codes*

9041 Refer to Section A.1.5.1 (on page 3298).

9042 **C.1.9** **Utility Limits**

9043 This section grew out of an idea that originated with the original POSIX.1, in the tables of system  
 9044 limits for the *sysconf()* and *pathconf()* functions. The idea being that a conforming application  
 9045 can be written to use the most restrictive values that a minimal system can provide, but it should  
 9046 not have to. The values provided represent compromises so that some vendors can use  
 9047 historically limited versions of UNIX system utilities. They are the highest values that a strictly  
 9048 conforming application can assume, given no other information.

9049 However, by using the *getconf* utility or the *sysconf()* function, the elegant application can be  
 9050 tailored to more liberal values on some of the specific instances of specific implementations.

9051 There is no explicitly stated requirement that an implementation provide finite limits for any of  
 9052 these numeric values; the implementation is free to provide essentially unbounded capabilities  
 9053 (where it makes sense), stopping only at reasonable points such as {ULONG\_MAX} (from the  
 9054 ISO C standard). Therefore, applications desiring to tailor themselves to the values on a  
 9055 particular implementation need to be ready for possibly huge values; it may not be a good idea  
 9056 to allocate blindly a buffer for an input line based on the value of {LINE\_MAX}, for instance.  
 9057 However, unlike the System Interfaces volume of IEEE Std 1003.1-200x, there is no set of limits  
 9058 that return a special indication meaning “unbounded”. The implementation should always  
 9059 return an actual number, even if the number is very large.

9060 The statement:

9061 “It is not guaranteed that the application ...”

9062 is an indication that many of these limits are designed to ensure that implementors design their  
 9063 utilities without arbitrary constraints related to unimaginative programming. There are certainly  
 9064 conditions under which combinations of options can cause failures that would not render an

9065 implementation non-conforming. For example, {EXPR\_NEST\_MAX} and {ARG\_MAX} could  
 9066 collide when expressions are large; combinations of {BC\_SCALE\_MAX} and {BC\_DIM\_MAX}  
 9067 could exceed virtual memory.

9068 In the Shell and Utilities volume of IEEE Std 1003.1-200x, the notion of a limit being guaranteed  
 9069 for the process lifetime, as it is in the System Interfaces volume of IEEE Std 1003.1-200x, is not as  
 9070 useful to a shell script. The *getconf* utility is probably a process itself, so the guarantee would be  
 9071 without value. Therefore, the Shell and Utilities volume of IEEE Std 1003.1-200x requires the  
 9072 guarantee to be for the session lifetime. This will mean that many vendors will either return very  
 9073 conservative values or possibly implement *getconf* as a built-in.

9074 It may seem confusing to have limits that apply only to a single utility grouped into one global  
 9075 section. However, the alternative, which would be to disperse them out into their utility  
 9076 description sections, would cause great difficulty when *sysconf()* and *getconf* were described.  
 9077 Therefore, the standard developers chose the global approach.

9078 Each language binding could provide symbol names that are slightly different from those shown |  
 9079 here. For example, the C-Language Binding option adds a leading underscore to the symbols as a |  
 9080 prefix.

9081 The following comments describe selection criteria for the symbols and their values:

9082 {ARG\_MAX}

9083 This is defined by the System Interfaces volume of IEEE Std 1003.1-200x. Unfortunately, it is  
 9084 very difficult for a conforming application to deal with this value, as it does not know how |  
 9085 much of its argument space is being consumed by the environment variables of the user.

9086 {BC\_BASE\_MAX}

9087 {BC\_DIM\_MAX}

9088 {BC\_SCALE\_MAX}

9089 These were originally one value, {BC\_SCALE\_MAX}, but it was unreasonable to link all  
 9090 three concepts into one limit.

9091 {CHILD\_MAX}

9092 This is defined by the System Interfaces volume of IEEE Std 1003.1-200x.

9093 {COLL\_WEIGHTS\_MAX}

9094 The weights assigned to **order** can be considered as “passes” through the collation  
 9095 algorithm.

9096 {EXPR\_NEST\_MAX}

9097 The value for expression nesting was borrowed from the ISO C standard.

9098 {LINE\_MAX}

9099 This is a global limit that affects all utilities, unless otherwise noted. The {MAX\_CANON}  
 9100 value from the System Interfaces volume of IEEE Std 1003.1-200x may further limit input  
 9101 lines from terminals. The {LINE\_MAX} value was the subject of much debate and is a  
 9102 compromise between those who wished to have unlimited lines and those who understood  
 9103 that many historical utilities were written with fixed buffers. Frequently, utility writers  
 9104 selected the UNIX system constant BUFSIZ to allocate these buffers; therefore, some utilities  
 9105 were limited to 512 bytes for I/O lines, while others achieved 4 096 bytes or greater.

9106 It should be noted that {LINE\_MAX} applies only to input line length; there is no  
 9107 requirement in IEEE Std 1003.1-200x that limits the length of output lines. Utilities such as  
 9108 *awk*, *sed*, and *paste* could theoretically construct lines longer than any of the input lines they  
 9109 received, depending on the options used or the instructions from the application. They are  
 9110 not required to truncate their output to {LINE\_MAX}. It is the responsibility of the  
 9111 application to deal with this. If the output of one of those utilities is to be piped into another

9112 of the standard utilities, line length restrictions will have to be considered; the *fold* utility,  
 9113 among others, could be used to ensure that only reasonable line lengths reach utilities or  
 9114 applications.

9115 {LINK\_MAX}

9116 This is defined by the System Interfaces volume of IEEE Std 1003.1-200x.

9117 {MAX\_CANON}

9118 {MAX\_INPUT}

9119 {NAME\_MAX}

9120 {NGROUPS\_MAX}

9121 {OPEN\_MAX}

9122 {PATH\_MAX}

9123 {PIPE\_BUF}

9124 These limits are defined by the System Interfaces volume of IEEE Std 1003.1-200x. Note that  
 9125 the byte lengths described by some of these values continue to represent bytes, even if the  
 9126 applicable character set uses a multi-byte encoding.

9127 {RE\_DUP\_MAX}

9128 The value selected is consistent with historical practice. Although the name implies that it  
 9129 applies to all REs, only BREs use the interval notation  $\{m,n\}$  addressed by this limit.

9130 {POSIX2\_SYMLINKS}

9131 The {POSIX2\_SYMLINKS} variable indicates that the underlying operating system supports  
 9132 the creation of symbolic links in specific directories. Many of the utilities defined in  
 9133 IEEE Std 1003.1-200x that deal with symbolic links do not depend on this value. For  
 9134 example, a utility that follows symbolic links (or does not, as the case may be) will only be  
 9135 affected by a symbolic link if it encounters one. Presumably, a file system that does not  
 9136 support symbolic links will not contain any. This variable does affect such utilities as *ln -s*  
 9137 and *pax* that attempt to create symbolic links.

9138 {POSIX2\_SYMLINKS} was developed even though there is no comparable configuration  
 9139 value for the system interfaces.

9140 There are different limits associated with command lines and input to utilities, depending on the  
 9141 method of invocation. In the case of a C program *exec*-ing a utility, {ARG\_MAX} is the  
 9142 underlying limit. In the case of the shell reading a script and *exec*-ing a utility, {LINE\_MAX}  
 9143 limits the length of lines the shell is required to process, and {ARG\_MAX} will still be a limit. If a  
 9144 user is entering a command on a terminal to the shell, requesting that it invoke the utility,  
 9145 {MAX\_INPUT} may restrict the length of the line that can be given to the shell to a value below  
 9146 {LINE\_MAX}.

9147 When an option is supported, *getconf* returns a value of 1. For example, when C development is  
 9148 supported:

```
9149     if [ "$(getconf POSIX2_C_DEV)" -eq 1 ]; then
9150         echo C supported
9151     fi
```

9152 The *sysconf()* function in the C-Language Binding option would return 1.

9153 The following comments describe selection criteria for the symbols and their values:

9154 POSIX2\_C\_BIND

9155 POSIX2\_C\_DEV

9156 POSIX2\_FORT\_DEV

9157 POSIX2\_FORT\_RUN

9158 POSIX2\_SW\_DEV

9159 POSIX2\_UPE

9160 It is possible for some (usually privileged) operations to remove utilities that support these  
 9161 options or otherwise to render these options unsupported. The header files, the *sysconf()*  
 9162 function, or the *getconf* utility will not necessarily detect such actions, in which case they  
 9163 should not be considered as rendering the implementation non-conforming. A test suite  
 9164 should not attempt tests such as:

```
9165     rm /usr/bin/c99
9166     getconf POSIX2_C_DEV
```

9167 POSIX2\_LOCALEDEF

9168 This symbol was introduced to allow implementations to restrict supported locales to only  
 9169 those supplied by the implementation.

### 9170 C.1.10 Grammar Conventions

9171 There is no additional rationale provided for this section.

### 9172 C.1.11 Utility Description Defaults

9173 This section is arranged with headings in the same order as all the utility descriptions. It is a  
 9174 collection of related and unrelated information concerning

- 9175 1. The default actions of utilities
- 9176 2. The meanings of notations used in IEEE Std 1003.1-200x that are specific to individual  
 9177 utility sections

9178 Although this material may seem out of place here, it is important that this information appear  
 9179 before any of the utilities to be described later.

#### 9180 NAME

9181 There is no additional rationale provided for this section.

#### 9182 SYNOPSIS

9183 There is no additional rationale provided for this section.

#### 9184 DESCRIPTION

9185 There is no additional rationale provided for this section.

#### 9186 OPTIONS

9187 Although it has not always been possible, the standard developers tried to avoid repeating  
 9188 information to reduce the risk that duplicate explanations could each be modified differently.

9189 The need to recognize `--` is required because conforming applications need to shield their |  
 9190 operands from any arbitrary options that the implementation may provide as an extension. For |  
 9191 example, if the standard utility *foo* is listed as taking no options, and the application needed to  
 9192 give it a pathname with a leading hyphen, it could safely do it as:

```
9193     foo -- -myfile
```

9194 and avoid any problems with `-m` used as an extension.

9195 **OPERANDS**

9196 The usage of `-` is never shown in the SYNOPSIS. Similarly, the usage of `--` is never shown.

9197 The requirement for processing operands in command-line order is to avoid a “WeirdNIX”  
 9198 utility that might choose to sort the input files alphabetically, by size, or by directory order.  
 9199 Although this might be acceptable for some utilities, in general the programmer has a right to  
 9200 know exactly what order will be chosen.

9201 Some of the standard utilities take multiple *file* operands and act as if they were processing the  
 9202 concatenation of those files. For example:

```
9203     asa file1 file2
```

9204 and:

```
9205     cat file1 file2 | asa
```

9206 have similar results when questions of file access, errors, and performance are ignored. Other  
 9207 utilities such as *grep* or *wc* have completely different results in these two cases. This latter type of  
 9208 utility is always identified in its DESCRIPTION or OPERANDS sections, whereas the former is  
 9209 not. Although it might be possible to create a general assertion about the former case, the  
 9210 following points must be addressed:

- 9211 • Access times for the files might be different in the operand case *versus* the *cat* case.
- 9212 • The utility may have error messages that are cognizant of the input filename, and this added  
 9213 value should not be suppressed. (As an example, *awk* sets a variable with the filename at  
 9214 each file boundary.)

9215 **STDIN**

9216 There is no additional rationale provided for this section.

9217 **INPUT FILES**

9218 A conforming application cannot assume the following three commands are equivalent:

```
9219 tail -n +2 file  

  9220 (sed -n 1q; cat) < file  

  9221 cat file | (sed -n 1q; cat)
```

9222 The second command is equivalent to the first only when the file is seekable. In the third  
 9223 command, if the file offset in the open file description were not unspecified, *sed* would have to be  
 9224 implemented so that it read from the pipe 1 byte at a time or it would have to employ some  
 9225 method to seek backwards on the pipe. Such functionality is not defined currently in POSIX.1  
 9226 and does not exist on all historical systems. Other utilities, such as *head*, *read*, and *sh*, have similar  
 9227 properties, so the restriction is described globally in this section.

9228 The definition of *text file* is strictly enforced for input to the standard utilities; very few of them  
 9229 list exceptions to the undefined results called for here. (Of course, “undefined” here does not  
 9230 mean that historical implementations necessarily have to change to start indicating error  
 9231 conditions. Conforming applications cannot rely on implementations succeeding or failing when  
 9232 non-text files are used.)

9233 The utilities that allow line continuation are generally those that accept input languages, rather  
 9234 than pure data. It would be unusual for an input line of this type to exceed `{LINE_MAX}` bytes  
 9235 and unreasonable to require that the implementation allow unlimited accumulation of multiple  
 9236 lines, each of which could reach `{LINE_MAX}`. Thus, for a conforming application the total of all  
 9237 the continued lines in a set cannot exceed `{LINE_MAX}`.

9238 The format description is intended to be sufficiently rigorous to allow other applications to  
9239 generate these input files. However, since <blank>s can legitimately be included in some of the  
9240 fields described by the standard utilities, particularly in locales other than the POSIX locale, this  
9241 intent is not always realized.

#### 9242 **ENVIRONMENT VARIABLES**

9243 There is no additional rationale provided for this section.

#### 9244 **ASYNCHRONOUS EVENTS**

9245 Because there is no language prohibiting it, a utility is permitted to catch a signal, perform some  
9246 additional processing (such as deleting temporary files), restore the default signal action (or  
9247 action inherited from the parent process), and resignal itself.

#### 9248 **STDOUT**

9249 The format description is intended to be sufficiently rigorous to allow post-processing of output  
9250 by other programs, particularly by an *awk* or *lex* parser.

#### 9251 **STDERR**

9252 This section does not describe error messages that refer to incorrect operation of the utility.  
9253 Consider a utility that processes program source code as its input. This section is used to  
9254 describe messages produced by a correctly operating utility that encounters an error in the  
9255 program source code on which it is processing. However, a message indicating that the utility  
9256 had insufficient memory in which to operate would not be described.

9257 Some utilities have traditionally produced warning messages without returning a non-zero exit  
9258 status; these are specifically noted in their sections. Other utilities shall not write to standard  
9259 error if they complete successfully, unless the implementation provides some sort of extension  
9260 to increase the verbosity or debugging level.

9261 The format descriptions are intended to be sufficiently rigorous to allow post-processing of  
9262 output by other programs.

#### 9263 **OUTPUT FILES**

9264 The format description is intended to be sufficiently rigorous to allow post-processing of output  
9265 by other programs, particularly by an *awk* or *lex* parser.

9266 Receipt of the SIGQUIT signal should generally cause termination (unless in some debugging  
9267 mode) that would bypass any attempted recovery actions.

#### 9268 **EXTENDED DESCRIPTION**

9269 There is no additional rationale provided for this section.

#### 9270 **EXIT STATUS**

9271 Note the additional discussion of exit values in *Exit Status for Commands* in the *sh* utility. It  
9272 describes requirements for returning exit values greater than 125.

9273 A utility may list zero as a successful return, 1 as a failure for a specific reason, and greater than  
9274 1 as “an error occurred”. In this case, unspecified conditions may cause a 2 or 3, or other value,  
9275 to be returned. A strictly conforming application should be written so that it tests for successful  
9276 exit status values (zero in this case), rather than relying upon the single specific error value listed  
9277 in IEEE Std 1003.1-200x. In that way, it will have maximum portability, even on implementations

- 9278 with extensions.
- 9279 The standard developers are aware that the general non-enumeration of errors makes it difficult  
9280 to write test suites that test the *incorrect* operation of utilities. There are some historical  
9281 implementations that have expended effort to provide detailed status messages and a helpful  
9282 environment to bypass or explain errors, such as prompting, retrying, or ignoring unimportant  
9283 syntax errors; other implementations have not. Since there is no realistic way to mandate system  
9284 behavior in cases of undefined application actions or system problems—in a manner acceptable  
9285 to all cultures and environments—attention has been limited to the correct operation of utilities  
9286 by the conforming application. Furthermore, the conforming application does not need detailed  
9287 information concerning errors that it caused through incorrect usage or that it cannot correct.
- 9288 There is no description of defaults for this section because all of the standard utilities specify  
9289 something (or explicitly state “Unspecified”) for exit status.
- 9290 **CONSEQUENCES OF ERRORS**
- 9291 Several actions are possible when a utility encounters an error condition, depending on the  
9292 severity of the error and the state of the utility. Included in the possible actions of various  
9293 utilities are: deletion of temporary or intermediate work files; deletion of incomplete files; and  
9294 validity checking of the file system or directory.
- 9295 The text about recursive traversing is meant to ensure that utilities such as *find* process as many  
9296 files in the hierarchy as they can. They should not abandon all of the hierarchy at the first error  
9297 and resume with the next command-line operand, but should attempt to keep going.
- 9298 **APPLICATION USAGE**
- 9299 This section provides additional caveats, issues, and recommendations to the developer.
- 9300 **EXAMPLES**
- 9301 This section provides sample usage.
- 9302 **RATIONALE**
- 9303 There is no additional rationale provided for this section.
- 9304 **FUTURE DIRECTIONS**
- 9305 FUTURE DIRECTIONS sections act as pointers to related work that may impact the interface in  
9306 the future, and often cautions the developer to architect the code to account for a change in this  
9307 area. Note that a future directions statement should not be taken as a commitment to adopt a  
9308 feature or interface in the future.
- 9309 **SEE ALSO**
- 9310 There is no additional rationale provided for this section.

9311 **CHANGE HISTORY**

9312 There is no additional rationale provided for this section.

9313 **C.1.12 Considerations for Utilities in Support of Files of Arbitrary Size**

9314 This section is intended to clarify the requirements for utilities in support of large files.

9315 The utilities listed in this section are utilities which are used to perform administrative tasks  
9316 such as to create, move, copy, remove, change the permissions, or measure the resources of a  
9317 file. They are useful both as end-user tools and as utilities invoked by applications during  
9318 software installation and operation.9319 The *chgrp*, *chmod*, *chown*, *ln*, and *rm* utilities probably require use of large file capable versions of  
9320 *stat()*, *lstat()*, *ftw()*, and the **stat** structure.9321 The *cat*, *cksum*, *cmp*, *cp*, *dd*, *mv*, *sum*, and *touch* utilities probably require use of large file capable  
9322 versions of *creat()*, *open()*, and *fopen()*.9323 The *cat*, *cksum*, *cmp*, *dd*, *df*, *du*, *ls*, and *sum* utilities may require writing large integer values. For  
9324 example:

- 9325 • The *cat* utility might have a `-n` option which counts `<newline>`s.
- 9326 • The *cksum* and *ls* utilities report file sizes.
- 9327 • The *cmp* utility reports the line number at which the first difference occurs, and also has a `-l`  
9328 option which reports file offsets.
- 9329 • The *dd*, *df*, *du*, *ls*, and *sum* utilities report block counts.

9330 The *dd*, *find*, and *test* utilities may need to interpret command arguments that contain 64-bit  
9331 values. For *dd*, the arguments include *skip=n*, *seek=n*, and *count=n*. For *find*, the arguments  
9332 include `-sizen`. For *test*, the arguments are those associated with algebraic comparisons.9333 The *df* utility might need to access large file systems with *statvfs()*.9334 The *ulimit* utility will need to use large file capable versions of *getrlimit()* and *setrlimit()* and be  
9335 able to read and write large integer values.9336 **C.1.13 Built-In Utilities**9337 All of these utilities can be *exec*-ed. There is no requirement that these utilities are actually built  
9338 into the shell itself, but many shells need the capability to do so because the Shell and Utilities  
9339 volume of IEEE Std 1003.1-200x, Section 2.9.1.1, Command Search and Execution requires that  
9340 they be found prior to the *PATH* search. The shell could satisfy its requirements by keeping a list  
9341 of the names and directly accessing the file-system versions regardless of *PATH*. Providing all of  
9342 the required functionality for those such as *cd* or *read* would be more difficult.9343 There were originally three justifications for allowing the omission of *exec*-able versions:

- 9344 1. It would require wasting space in the file system, at the expense of very small systems.  
9345 However, it has been pointed out that all 16 utilities in the table can be provided with 16  
9346 links to a single-line shell script:  
9347 `$0 "$@"`
- 9348 2. It is not logical to require invocation of utilities such as *cd* because they have no value  
9349 outside the shell environment or cannot be useful in a child process. However, counter-  
9350 examples always seemed to be available for even the most unusual cases:

9351 `find . -type d -exec cd {} \; -exec foo {} \;` |  
 9352 (which invokes “foo” on accessible directories) |

9353 `ps ... | sed ... | xargs kill` |

9354 `find . -exec true \; -a ...` |  
 9355 (where “true” is used for temporary debugging) |

9356 3. It is confusing to have a utility such as *kill* that can easily be in the file system in the base |  
 9357 standard, but that requires built-in status for the UPE (for the % job control job ID notation). |  
 9358 It was decided that it was more appropriate to describe the required functionality (rather |  
 9359 than the implementation) to the system implementors and let them decide how to satisfy |  
 9360 it. |

9361 On the other hand, it was realized that any distinction like this between utilities was not useful |  
 9362 to applications, and that the cost to correct it was small. These arguments were ultimately the |  
 9363 most effective. |

9364 There were varying reasons for including utilities in the table of built-ins: |

9365 *alias, fc, unalias* |  
 9366 The functionality of these utilities is performed more simply within the shell itself and that |  
 9367 is the model most historical implementations have used. |

9368 *bg, fg, jobs* |  
 9369 All of the job control-related utilities are eligible for built-in status because that is the model |  
 9370 most historical implementations have used. |

9371 *cd, getopts, newgrp, read, umask, wait* |  
 9372 The functionality of these utilities is performed more simply within the context of the |  
 9373 current process. An example can be taken from the usage of the *cd* utility. The purpose of |  
 9374 the utility is to change the working directory for subsequent operations. The actions of *cd* |  
 9375 affect the process in which *cd* is executed and all subsequent child processes of that process. |  
 9376 Based on the ISO POSIX-1 standard p1 process model, changes in the process environment |  
 9377 of a child process have no effect on the parent process. If the *cd* utility were executed from a |  
 9378 child process, the working directory change would be effective only in the child process. |  
 9379 Child processes initiated subsequent to the child process that executed the *cd* utility would |  
 9380 not have a changed working directory relative to the parent process. |

9381 *command* |  
 9382 This utility was placed in the table primarily to protect scripts that are concerned about |  
 9383 their *PATH* being manipulated. The “secure” shell script example in the *command* utility in |  
 9384 the Shell and Utilities volume of IEEE Std 1003.1-200x would not be possible if a *PATH* |  
 9385 change retrieved an alien version of *command*. (An alternative would have been to |  
 9386 implement *getconf* as a built-in, but the standard developers considered that it carried too |  
 9387 many changing configuration strings to require in the shell.) |

9388 *kill* |  
 9389 Since *kill* provides optional job control functionality using shell notation (%1, %2, and so on), |  
 9390 some implementations would find it extremely difficult to provide this outside the shell. |

9391 *true, false* |  
 9392 These are in the table as a courtesy to programmers who wish to use the “**while true**” shell |  
 9393 construct without protecting *true* from *PATH* searches. (It is acknowledged that “**while\:**” |  
 9394 also works, but the idiom with *true* is historically pervasive.) |

9395 All utilities, including those in the table, are accessible via the *system()* and *popen()* functions in |  
 9396 the System Interfaces volume of IEEE Std 1003.1-200x. There are situations where the return |

9397 functionality of *system()* and *popen()* is not desirable. Applications that require the exit status of |  
 9398 the invoked utility will not be able to use *system()* or *popen()*, since the exit status returned is |  
 9399 that of the command language interpreter rather than that of the invoked utility. The alternative |  
 9400 for such applications is the use of the *exec* family. |

## 9401 **C.2 Shell Command Language**

### 9402 **C.2.1 Shell Introduction**

9403 The System V shell was selected as the starting point for the Shell and Utilities volume of  
 9404 IEEE Std 1003.1-200x. The BSD C shell was excluded from consideration for the following  
 9405 reasons:

- 9406 • Most historically portable shell scripts assume the Version 7 Bourne shell, from which the  
 9407 System V shell is derived.
- 9408 • The majority of tutorial materials on shell programming assume the System V shell.

9409 The construct "#!" is reserved for implementations wishing to provide that extension. If it were  
 9410 not reserved, the Shell and Utilities volume of IEEE Std 1003.1-200x would disallow it by forcing  
 9411 it to be a comment. As it stands, a strictly conforming application must not use "#!" as the first  
 9412 two characters of the file.

### 9413 **C.2.2 Quoting**

9414 There is no additional rationale provided for this section.

#### 9415 *C.2.2.1 Escape Character (Backslash)*

9416 There is no additional rationale provided for this section.

#### 9417 *C.2.2.2 Single-Quotes*

9418 A backslash cannot be used to escape a single-quote in a single-quoted string. An embedded  
 9419 quote can be created by writing, for example: "'a'\ 'b'", which yields "a'b". (See the Shell  
 9420 and Utilities volume of IEEE Std 1003.1-200x, Section 2.6.5, Field Splitting for a better  
 9421 understanding of how portions of words are either split into fields or remain concatenated.) A  
 9422 single token can be made up of concatenated partial strings containing all three kinds of quoting  
 9423 or escaping, thus permitting any combination of characters.

#### 9424 *C.2.2.3 Double-Quotes*

9425 The escaped <newline> used for line continuation is removed entirely from the input and is not  
 9426 replaced by any white space. Therefore, it cannot serve as a token separator.

9427 In double-quoting, if a backslash is immediately followed by a character that would be  
 9428 interpreted as having a special meaning, the backslash is deleted and the subsequent character is  
 9429 taken literally. If a backslash does not precede a character that would have a special meaning, it  
 9430 is left in place unmodified and the character immediately following it is also left unmodified.  
 9431 Thus, for example:

9432 "\\$" → \$ |

9433 "\a" → \a |

9434 It would be desirable to include the statement “The characters from an enclosed “\${” to the  
 9435 matching ’}’ shall not be affected by the double quotes”, similar to the one for “\$( )”.  
 9436 However, historical practice in the System V shell prevents this.

9437 The requirement that double-quotes be matched inside “\${ . . . }” within double-quotes and the  
 9438 rule for finding the matching ’}’ in the Shell and Utilities volume of IEEE Std 1003.1-200x,  
 9439 Section 2.6.2, Parameter Expansion eliminate several subtle inconsistencies in expansion for  
 9440 historical shells in rare cases; for example:

```
9441     "${foo-bar}"
```

9442 yields **bar** when **foo** is not defined, and is an invalid substitution when **foo** is defined, in many  
 9443 historical shells. The differences in processing the “\${ . . . }” form have led to inconsistencies  
 9444 between historical systems. A consequence of this rule is that single-quotes cannot be used to  
 9445 quote the ’}’ within “\${ . . . }”; for example:

```
9446     unset bar
9447     foo="${bar-'}'"
```

9448 is invalid because the “\${ . . . }” substitution contains an unpaired unescaped single-quote. The  
 9449 backslash can be used to escape the ’}’ in this example to achieve the desired result:

```
9450     unset bar
9451     foo="${bar-\}]"
```

9452 The differences in processing the “\${ . . . }” form have led to inconsistencies between the  
 9453 historical System V shell, BSD, and KornShells, and the text in the Shell and Utilities volume of  
 9454 IEEE Std 1003.1-200x is an attempt to converge them without breaking too many applications.  
 9455 The only alternative to this compromise between shells would be to make the behavior  
 9456 unspecified whenever the literal characters ‘ ’, ‘{’, ‘}’, and ‘”’ appear within “\${ . . . }”.  
 9457 To write a portable script that uses these values, a user would have to assign variables; for  
 9458 example:

```
9459     squote=\' dquote=\" lbrace='{ rbrace=}'
9460     ${foo-$$squote$$rbrace$$squote}
```

9461 rather than:

```
9462     ${foo-"' }'"}
```

9463 Some implementations have allowed the end of the word to terminate the backquoted command  
 9464 substitution, such as in:

```
9465     "`echo hello"
```

9466 This usage is undefined; the matching backquote is required by the Shell and Utilities volume of  
 9467 IEEE Std 1003.1-200x. The other undefined usage can be illustrated by the example:

```
9468     sh -c '` echo "foo`'
```

9469 The description of the recursive actions involving command substitution can be illustrated with  
 9470 an example. Upon recognizing the introduction of command substitution, the shell parses input  
 9471 (in a new context), gathering the source for the command substitution until an unbalanced ’)’  
 9472 or ‘`’ is located. For example, in the following:

```
9473     echo "$(date; echo "  

  9474         one" )"
```

9475 the double-quote following the *echo* does not terminate the first double-quote; it is part of the  
 9476 command substitution script. Similarly, in:

9477           echo "\$ (echo \* )" |  
 9478           the asterisk is not quoted since it is inside command substitution; however:  
 9479           echo "\$ (echo "\*" )" |  
 9480           is quoted (and represents the asterisk character itself).

### 9481 C.2.3 Token Recognition

9482           The "(" and ")" symbols are control operators in the KornShell, used for an alternative |  
 9483           syntax of an arithmetic expression command. A conforming application cannot use "(" as a |  
 9484           single token (with the exception of the "\$ (" form for shell arithmetic).

9485           On some implementations, the symbol "(" is a control operator; its use produces unspecified |  
 9486           results. Applications that wish to have nested subshells, such as:

```
9487           ((echo Hello);(echo World)) |
```

9488           shall separate the "(" characters into two tokens by including white space between them. |  
 9489           Some systems may treat these as invalid arithmetic expressions instead of subshells.

9490           Certain combinations of characters are invalid in portable scripts, as shown in the grammar. |  
 9491           Implementations may use these combinations (such as "|&") as valid control operators. Portable |  
 9492           scripts cannot rely on receiving errors in all cases where this volume of IEEE Std 1003.1-200x |  
 9493           indicates that a syntax is invalid.

9494           The (3) rule about combining characters to form operators is not meant to preclude systems from |  
 9495           extending the shell language when characters are combined in otherwise invalid ways. |  
 9496           Conforming applications cannot use invalid combinations, and test suites should not penalize |  
 9497           systems that take advantage of this fact. For example, the unquoted combination "|&" is not |  
 9498           valid in a POSIX script, but has a specific KornShell meaning.

9499           The (10) rule about '#' as the current character is the first in the sequence in which a new token |  
 9500           is being assembled. The '#' starts a comment only when it is at the beginning of a token. This |  
 9501           rule is also written to indicate that the search for the end-of-comment does not consider escaped |  
 9502           <newline> specially, so that a comment cannot be continued to the next line.

#### 9503 C.2.3.1 Alias Substitution

9504           The alias capability was added in the UPE because it is widely used in historical |  
 9505           implementations by interactive users.

9506           The definition of *alias name* precludes an alias name containing a slash character. Since the text |  
 9507           applies to the command words of simple commands, reserved words (in their proper places) |  
 9508           cannot be confused with aliases.

9509           The placement of alias substitution in token recognition makes it clear that it precedes all of the |  
 9510           word expansion steps.

9511           An example concerning trailing <blank>s and reserved words follows. If the user types:

```
9512           $ alias foo="/bin/ls " |  

  9513           $ alias while="/ " |
```

9514           The effect of executing:

```

9515     $ while true
9516     > do
9517     > echo "Hello, World"
9518     > done

```

9519 is a never-ending sequence of "Hello, World" strings to the screen. However, if the user  
 9520 types:

```

9521     $ foo while

```

9522 the result is an *ls* listing of /. Since the alias substitution for **foo** ends in a <space>, the next word  
 9523 is checked for alias substitution. The next word, **while**, has also been aliased, so it is substituted  
 9524 as well. Since it is not in the proper position as a command word, it is not recognized as a  
 9525 reserved word.

9526 If the user types:

```

9527     $ foo; while

```

9528 **while** retains its normal reserved-word properties.

#### 9529 C.2.4 Reserved Words

9530 All reserved words are recognized syntactically as such in the contexts described. However, note  
 9531 that **in** is the only meaningful reserved word after a **case** or **for**; similarly, **in** is not meaningful as  
 9532 the first word of a simple command.

9533 Reserved words are recognized only when they are delimited (that is, meet the definition of the  
 9534 Base Definitions volume of IEEE Std 1003.1-200x, Section 3.435, Word), whereas operators are  
 9535 themselves delimiters. For instance, '( ' and ' ) ' are control operators, so that no <space> is  
 9536 needed in (*list*). However, '{ ' and ' } ' are reserved words in { *list*; }, so that in this case the  
 9537 leading <space> and semicolon are required.

9538 The list of unspecified reserved words is from the KornShell, so conforming applications cannot  
 9539 use them in places a reserved word would be recognized. This list contained **time** in early  
 9540 proposals, but it was removed when the *time* utility was selected for the Shell and Utilities  
 9541 volume of IEEE Std 1003.1-200x.

9542 There was a strong argument for promoting braces to operators (instead of reserved words), so  
 9543 they would be syntactically equivalent to subshell operators. Concerns about compatibility  
 9544 outweighed the advantages of this approach. Nevertheless, conforming applications should  
 9545 consider quoting '{ ' and ' } ' when they represent themselves.

9546 The restriction on ending a name with a colon is to allow future implementations that support  
 9547 named labels for flow control; see the RATIONALE for the *break* built-in utility .

9548 It is possible that a future version of the Shell and Utilities volume of IEEE Std 1003.1-200x may  
 9549 require that '{ ' and ' } ' be treated individually as control operators, although the token "{ }"  
 9550 will probably be a special-case exemption from this because of the often-used *find*{ } construct.

9551 **C.2.5 Parameters and Variables**9552 *C.2.5.1 Positional Parameters*

9553 There is no additional rationale provided for this section.

9554 *C.2.5.2 Special Parameters*

9555 Most historical implementations implement subshells by forking; thus, the special parameter  
 9556 ' \$ ' does not necessarily represent the process ID of the shell process executing the commands  
 9557 since the subshell execution environment preserves the value of ' \$ '.

9558 If a subshell were to execute a background command, the value of "\$!" for the parent would  
 9559 not change. For example:

```
9560 (
9561     date &
9562     echo $!
9563 )
9564     echo $!
```

9565 would echo two different values for "\$!".

9566 The "\$-" special parameter can be used to save and restore *set* options:

```
9567     Save=$(echo $- | sed 's/[ics]//g')
9568     ...
9569     set +aCefnuvx
9570     if [ -n "$Save" ]; then
9571         set -$Save
9572     fi
```

9573 The three options are removed using *sed* in the example because they may appear in the value of  
 9574 "\$-" (from the *sh* command line), but are not valid options to *set*.

9575 The descriptions of parameters '\*' and '@' assume the reader is familiar with the field splitting  
 9576 discussion in the Shell and Utilities volume of IEEE Std 1003.1-200x, Section 2.6.5, Field Splitting  
 9577 and understands that portions of the word remain concatenated unless there is some reason to  
 9578 split them into separate fields.

9579 Some examples of the '\*' and '@' properties, including the concatenation aspects:

```
9580     set "abc" "def ghi" "jkl"
9581     echo $*      => "abc" "def" "ghi" "jkl"
9582     echo "$*"   => "abc def ghi jkl"
9583     echo $@     => "abc" "def" "ghi" "jkl"
```

9584 **but:**

```
9585     echo "$@"      => "abc" "def ghi" "jkl"
9586     echo "xx$@yy" => "xxabc" "def ghi" "jkl"yy"
9587     echo "$@$@"   => "abc" "def ghi" "jklabc" "def ghi" "jkl"
```

9588 In the preceding examples, the double-quote characters that appear after the "=>" do not appear  
 9589 in the output and are used only to illustrate word boundaries.

9590 The following example illustrates the effect of setting *IFS* to a null string:

```

9591      $ IFS=' '
9592      $ set foo bar bam
9593      $ echo "$@"
9594      foo bar bam
9595      $ echo "$*"
9596      foobarbam
9597      $ unset IFS
9598      $ echo "$*"
9599      foo bar bam

```

### 9600 C.2.5.3 Shell Variables

9601 See the discussion of *IFS* in Section C.2.6.5 (on page 3529).

9602 The prohibition on *LC\_CTYPE* changes affecting lexical processing protects the shell  
 9603 implementor (and the shell programmer) from the ill effects of changing the definition of  
 9604 <blank> or the set of alphabetic characters in the current environment. It would probably not be  
 9605 feasible to write a compiled version of a shell script without this rule. The rule applies only to  
 9606 the current invocation of the shell and its subshells—invoking a shell script or performing *exec sh*  
 9607 would subject the new shell to the changes in *LC\_CTYPE*.

9608 Other common environment variables used by historical shells are not specified by the Shell and  
 9609 Utilities volume of IEEE Std 1003.1-200x, but they should be reserved for the historical uses.

9610 Tilde expansion for components of the *PATH* in an assignment such as:

```
9611     PATH=~hlj/bin:~dwc/bin:$PATH
```

9612 is a feature of some historical shells and is allowed by the wording of the Shell and Utilities  
 9613 volume of IEEE Std 1003.1-200x, Section 2.6.1, Tilde Expansion. Note that the tildes are expanded  
 9614 during the assignment to *PATH*, not when *PATH* is accessed during command search.

9615 The following entries represent additional information about variables included in the Shell and  
 9616 Utilities volume of IEEE Std 1003.1-200x, or rationale for common variables in use by shells that  
 9617 have been excluded:

9618 — (Underscore.) While underscore is historical practice, its overloaded usage in  
 9619 the KornShell is confusing, and it has been omitted from the Shell and Utilities  
 9620 volume of IEEE Std 1003.1-200x.

9621 *ENV* This variable can be used to set aliases and other items local to the invocation  
 9622 of a shell. The file referred to by *ENV* differs from *\$HOME/.profile* in that  
 9623 *.profile* is typically executed at session start-up, whereas the *ENV* file is  
 9624 executed at the beginning of each shell invocation. The *ENV* value is  
 9625 interpreted in a manner similar to a dot script, in that the commands are  
 9626 executed in the current environment and the file needs to be readable, but not  
 9627 executable. However, unlike dot scripts, no *PATH* searching is performed.  
 9628 This is used as a guard against Trojan Horse security breaches.

9629 *ERRNO* This variable was omitted from the Shell and Utilities volume of  
 9630 IEEE Std 1003.1-200x because the values of error numbers are not defined in  
 9631 IEEE Std 1003.1-200x in a portable manner.

9632 *FCEDIT* Since this variable affects only the *fc* utility, it has been omitted from this more  
 9633 global place. The value of *FCEDIT* does not affect the command line editing  
 9634 mode in the shell; see the description of *set -o vi* in the *set* built-in utility.

9635	<i>PS1</i>	This variable is used for interactive prompts. Historically, the “superuser”
9636		has had a prompt of ‘#’. Since privileges are not required to be monolithic, it
9637		is difficult to define which privileges should cause the alternate prompt.
9638		However, a sufficiently powerful user should be reminded of that power by
9639		having an alternate prompt.
9640	<i>PS3</i>	This variable is used by the KornShell for the <i>select</i> command. Since the POSIX
9641		shell does not include <i>select</i> , <i>PS3</i> was omitted.
9642	<i>PS4</i>	This variable is used for shell debugging. For example, the following script:
9643		<pre>PS4='[ \${LINENO} ]+ '</pre>
9644		<pre>set -x</pre>
9645		<pre>echo Hello</pre>
9646		writes the following to standard error:
9647		<pre>[3]+ echo Hello</pre>
9648	<i>RANDOM</i>	This pseudo-random number generator was not seen as being useful to
9649		interactive users.
9650	<i>SECONDS</i>	Although this variable is sometimes used with <i>PS1</i> to allow the display of the
9651		current time in the prompt of the user, it is not one that would be manipulated
9652		frequently enough by an interactive user to include in the Shell and Utilities
9653		volume of IEEE Std 1003.1-200x.

## 9654 C.2.6 Word Expansions

9655 Step (2) refers to the “portions of fields generated by step (1)”. For example, if the word being  
 9656 expanded were “\$x+\$y” and *IFS*=+, the word would be split only if “\$x” or “\$y” contained  
 9657 ‘+’; the ‘+’ in the original word was not generated by step (1).

9658 *IFS* is used for performing field splitting on the results of parameter and command substitution;  
 9659 it is not used for splitting all fields. Previous versions of the shell used it for splitting all fields  
 9660 during field splitting, but this has severe problems because the shell can no longer parse its own  
 9661 script. There are also important security implications caused by this behavior. All useful  
 9662 applications of *IFS* use it for parsing input of the *read* utility and for splitting the results of  
 9663 parameter and command substitution.

9664 The rule concerning expansion to a single field requires that if **foo=abc** and **bar=def**, that:

```
9665 "$foo" "$bar"
```

9666 expands to the single field:

```
9667 abcdef
```

9668 The rule concerning empty fields can be illustrated by:

```

9669      $   unset foo
9670      $   set $foo bar ' ' xyz "$foo" abc
9671      $   for i
9672      >   do
9673      >     echo "-$i-"
9674      >   done
9675      -bar-
9676      --
9677      -xyz-
9678      --
9679      -abc-

```

9680 Step (1) indicates that parameter expansion, command substitution, and arithmetic expansion  
 9681 are all processed simultaneously as they are scanned. For example, the following is valid  
 9682 arithmetic:

```

9683      x=1
9684      echo $(( $(echo 3)+$x ))

```

9685 An early proposal stated that tilde expansion preceded the other steps, but this is not the case in  
 9686 known historical implementations; if it were, and if a referenced home directory contained a '\$'  
 9687 character, expansions would result within the directory name.

#### 9688 C.2.6.1 Tilde Expansion

9689 Tilde expansion generally occurs only at the beginning of words, but an exception based on  
 9690 historical practice has been included:

```

9691      PATH=/posix/bin:~djk/bin

```

9692 This is eligible for tilde expansion because tilde follows a colon and none of the relevant  
 9693 characters is quoted. Consideration was given to prohibiting this behavior because any of the  
 9694 following are reasonable substitutes:

```

9695      PATH=$(printf %s ~karels/bin : ~bostic/bin)
9696      for Dir in ~maat/bin ~srb/bin ...
9697      do
9698          PATH=${PATH:+$PATH:}$Dir
9699      done

```

9700 In the first command, explicit colons are used for each directory. In all cases, the shell performs  
 9701 tilde expansion on each directory because all are separate words to the shell.

9702 Note that expressions in operands such as:

```

9703      make -k mumble LIBDIR=~chet/lib

```

9704 do not qualify as shell variable assignments, and tilde expansion is not performed (unless the  
 9705 command does so itself, which *make* does not).

9706 Because of the requirement that the word is not quoted, the following are not equivalent; only  
 9707 the last causes tilde expansion:

```

9708      \~hlj/  ~h\lj/  ~"hlj"/  ~hlj\  ~hlj/

```

9709 In an early proposal, tilde expansion occurred following any unquoted equals sign or colon, but  
 9710 this was removed because of its complexity and to avoid breaking commands such as:

9711 rcp hostname:~marc/.profile .

9712 A suggestion was made that the special sequence "\$~" should be allowed to force tilde  
9713 expansion anywhere. Since this is not historical practice, it has been left for future  
9714 implementations to evaluate. (The description in the Shell and Utilities volume of  
9715 IEEE Std 1003.1-200x, Section 2.2, Quoting requires that a dollar sign be quoted to represent  
9716 itself, so the "\$~" combination is already unspecified.)

9717 The results of giving tilde with an unknown login name are undefined because the KornShell  
9718 "~+" and "~-" constructs make use of this condition, but in general it is an error to give an  
9719 incorrect login name with tilde. The results of having *HOME* unset are unspecified because some  
9720 historical shells treat this as an error.

### 9721 C.2.6.2 Parameter Expansion

9722 The rule for finding the closing '}' in "\${...}" is the one used in the KornShell and is  
9723 upwardly-compatible with the Bourne shell, which does not determine the closing '}' until the  
9724 word is expanded. The advantage of this is that incomplete expansions, such as:

9725 \${foo

9726 can be determined during tokenization, rather than during expansion.

9727 The string length and substring capabilities were included because of the demonstrated need for  
9728 them, based on their usage in other shells, such as C shell and KornShell.

9729 Historical versions of the KornShell have not performed tilde expansion on the word part of  
9730 parameter expansion; however, it is more consistent to do so.

### 9731 C.2.6.3 Command Substitution

9732 The "\$()" form of command substitution solves a problem of inconsistent behavior when using  
9733 backquotes. For example:

Command	Output
echo '\\$x'	\\$x
echo `echo '\\$x'`	\$x
echo \$(echo '\\$x')	\\$x

9738 Additionally, the backquoted syntax has historical restrictions on the contents of the embedded  
9739 command. While the newer "\$()" form can process any kind of valid embedded script, the  
9740 backquoted form cannot handle some valid scripts that include backquotes. For example, these  
9741 otherwise valid embedded scripts do not work in the left column, but do work on the right:

9742 echo `	echo \$(
9743 cat <<\eof	cat <<\eof
9744 a here-doc with `	a here-doc with )
9745 eof	eof
9746 `	)
9747 echo `	echo \$(
9748 echo abc # a comment with `	echo abc # a comment with )
9749 `	)
9750 echo `	echo \$(
9751 echo ` ` `	echo ` ` `
9752 `	)

9753 Because of these inconsistent behaviors, the backquoted variety of command substitution is not  
 9754 recommended for new applications that nest command substitutions or attempt to embed  
 9755 complex scripts.

9756 The KornShell feature:

9757 If *command* is of the form `<word`, *word* is expanded to generate a pathname, and the value of |  
 9758 the command substitution is the contents of this file with any trailing `<newline>`s deleted. |

9759 was omitted from the Shell and Utilities volume of IEEE Std 1003.1-200x because `$(cat word)` is  
 9760 an appropriate substitute. However, to prevent breaking numerous scripts relying on this  
 9761 feature, it is unspecified to have a script within `"$( )"` that has only redirections.

9762 The requirement to separate `"$( "` and `'( '` when a single subshell is command-substituted is to  
 9763 avoid any ambiguities with arithmetic expansion.

#### 9764 C.2.6.4 Arithmetic Expansion

9765 The `"( )"` form of KornShell arithmetic in early proposals was omitted. The standard  
 9766 developers concluded that there was a strong desire for some kind of arithmetic evaluator to  
 9767 replace *expr*, and that relating it to `'$'` makes it work well with the standard shell language, and  
 9768 it provides access to arithmetic evaluation in places where accessing a utility would be  
 9769 inconvenient.

9770 The syntax and semantics for arithmetic were changed for the ISO/IEC 9945-2:1993 standard.  
 9771 The language is essentially a pure arithmetic evaluator of constants and operators (excluding  
 9772 assignment) and represents a simple subset of the previous arithmetic language (which was  
 9773 derived from the KornShell `"( )"` construct). The syntax was changed from that of a  
 9774 command denoted by `((expression))` to an expansion denoted by `$(expression)`. The new form is  
 9775 a dollar expansion `'$'` that evaluates the expression and substitutes the resulting value.  
 9776 Objections to the previous style of arithmetic included that it was too complicated, did not fit in  
 9777 well with the use of variables in the shell, and its syntax conflicted with subshells. The  
 9778 justification for the new syntax is that the shell is traditionally a macro language, and if a new  
 9779 feature is to be added, it should be accomplished by extending the capabilities presented by the  
 9780 current model of the shell, rather than by inventing a new one outside the model; adding a new  
 9781 dollar expansion was perceived to be the most intuitive and least destructive way to add such a  
 9782 new capability.

9783 In early proposals, a form `$(expression)` was used. It was functionally equivalent to the `"$( )"`  
 9784 of the current text, but objections were lodged that the 1988 KornShell had already implemented  
 9785 `"$( )"` and there was no compelling reason to invent yet another syntax. Furthermore, the  
 9786 `"$[ ]"` syntax had a minor incompatibility involving the patterns in `case` statements.

9787 The portion of the ISO C standard arithmetic operations selected corresponds to the operations  
 9788 historically supported in the KornShell.

9789 It was concluded that the `test` command (`D`) was sufficient for the majority of relational arithmetic  
 9790 tests, and that tests involving complicated relational expressions within the shell are rare, yet  
 9791 could still be accommodated by testing the value of `"$( )"` itself. For example:

```
9792     # a complicated relational expression
9793     while [ $( ( (($x + $y)/($a * $b)) < ($foo*$bar) ) ) -ne 0 ]
```

9794 or better yet, the rare script that has many complex relational expressions could define a  
 9795 function like this:

```

9796     val() {
9797         return $((!$1))
9798     }

```

9799 and complicated tests would be less intimidating:

```

9800     while val $(( (($x + $y)/($a * $b)) < ($foo*$bar) ))
9801     do
9802         # some calculations
9803     done

```

9804 A suggestion that was not adopted was to modify *true* and *false* to take an optional argument,  
 9805 and *true* would exit true only if the argument was non-zero, and *false* would exit false only if the  
 9806 argument was non-zero:

```

9807     while true $(( $x > 5 && $y <= 25 ))

```

9808 There is a minor portability concern with the new syntax. The example `$((2+2))` could have been  
 9809 intended to mean a command substitution of a utility named `2+2` in a subshell. The standard  
 9810 developers considered this to be obscure and isolated to some KornShell scripts (because `"$()"`  
 9811 command substitution existed previously only in the KornShell). The text on command  
 9812 substitution requires that the `"$("` and `'('` be separate tokens if this usage is needed.

9813 An example such as:

```

9814     echo $((echo hi);(echo there))

```

9815 should not be misinterpreted by the shell as arithmetic because attempts to balance the  
 9816 parentheses pairs would indicate that they are subshells. However, as indicated by the Base  
 9817 Definitions volume of IEEE Std 1003.1-200x, Section 3.112, Control Operator, a conforming  
 9818 application must separate two adjacent parentheses with white space to indicate nested  
 9819 subshells.

9820 Although the ISO/IEC 9899:1999 standard now requires support for **long long** and allows  
 9821 extended integer types with higher ranks, IEEE Std 1003.1-200x only requires arithmetic  
 9822 expansions to support **signed long** integer arithmetic. Implementations are encouraged to  
 9823 support signed integer values at least as large as the size of the largest file allowed on the  
 9824 implementation.

9825 Implementations are also allowed to perform floating-point evaluations as long as an  
 9826 application won't see different results for expressions that would not overflow **signed long**  
 9827 integer expression evaluation. (This includes appropriate truncation of results to integer values.)

9828 Changes made in response to IEEE PASC Interpretation 1003.2 #208 removed the requirement  
 9829 that the integer constant suffixes `l` and `L` had to be recognized. The ISO POSIX-2: 1993 standard  
 9830 didn't require the `u`, `uL`, `uL`, `U`, `UL`, `UL`, `lu`, `lU`, `Lu`, and `LU` suffixes since only signed integer  
 9831 arithmetic was required. Since all arithmetic expressions were treated as handling **signed long**  
 9832 integer types anyway, the `l` and `L` suffixes were redundant. No known scripts used them and  
 9833 some historic shells didn't support them. When the ISO/IEC 9899: 1999 standard was used as the  
 9834 basis for the description of arithmetic processing, the `ll` and `LL` suffixes and combinations were  
 9835 also not required. Implementations are still free to accept any or all of these suffices, but are not  
 9836 required to do so.

9837 There was also some confusion as to whether the shell was required to recognize character  
 9838 constants. Syntactically, character constants were required to be recognized, but the  
 9839 requirements for the handling of backslash ("`\"`") and quote ("`'\''`") characters (needed to  
 9840 specify character constants) within an arithmetic expansion were ambiguous. Furthermore, no  
 9841 known shells supported them. Changes made in response to IEEE PASC Interpretation 1003.2

9842 #208 removed the requirement to support them (if they were indeed required before). |  
 9843 IEEE Std 1003.1-200x clearly does not require support for character constants. |

#### 9844 C.2.6.5 Field Splitting

9845 The operation of field splitting using *IFS*, as described in early proposals, was based on the way  
 9846 the KornShell splits words, but it is incompatible with other common versions of the shell.  
 9847 However, each has merit, and so a decision was made to allow both. If the *IFS* variable is unset  
 9848 or is <space><tab><newline>, the operation is equivalent to the way the System V shell splits  
 9849 words. Using characters outside the <space><tab><newline> set yields the KornShell behavior,  
 9850 where each of the non-<space><tab><newline>s is significant. This behavior, which affords the  
 9851 most flexibility, was taken from the way the original *awk* handled field splitting.

9852 Rule (3) can be summarized as a pseudo-ERE:

```
9853 (s*ns*|s+)
```

9854 where *s* is an *IFS* white space character and *n* is a character in the *IFS* that is not white space.  
 9855 Any string matching that ERE delimits a field, except that the *s+* form does not delimit fields at  
 9856 the beginning or the end of a line. For example, if *IFS* is <space>/<comma>/<tab>, the string:

```
9857 <space><space>red<space><space>, <space>white<space>blue
```

9858 yields the three colors as the delimited fields.

#### 9859 C.2.6.6 Pathname Expansion

9860 There is no additional rationale provided for this section.

#### 9861 C.2.6.7 Quote Removal

9862 There is no additional rationale provided for this section.

### 9863 C.2.7 Redirection

9864 In the System Interfaces volume of IEEE Std 1003.1-200x, file descriptors are integers in the range  
 9865 0–({OPEN\_MAX}–1). The file descriptors discussed in the Shell and Utilities volume of  
 9866 IEEE Std 1003.1-200x, Section 2.7, Redirection are that same set of small integers.

9867 Having multi-digit file descriptor numbers for I/O redirection can cause some obscure  
 9868 compatibility problems. Specifically, scripts that depend on an example command:

```
9869 echo 22>/dev/null
```

9870 echoing 2 to standard error or 22 to standard output are no longer portable. However, the file  
 9871 descriptor number still must be delimited from the preceding text. For example:

```
9872 cat file2>foo
```

9873 writes the contents of **file2**, not the contents of **file**.

9874 The ">|" format of output redirection was adopted from the KornShell. Along with the  
 9875 *noclobber* option, *set -C*, it provides a safety feature to prevent inadvertent overwriting of  
 9876 existing files. (See the RATIONALE for the *pathchk* utility for why this step was taken.) The  
 9877 restriction on regular files is historical practice.

9878 The System V shell and the KornShell have differed historically on pathname expansion of *word*;  
 9879 the former never performed it, the latter only when the result was a single field (file). As a  
 9880 compromise, it was decided that the KornShell functionality was useful, but only as a shorthand  
 9881 device for interactive users. No reasonable shell script would be written with a command such

9882 as:

```
9883     cat foo > a*
```

9884 Thus, shell scripts are prohibited from doing it, while interactive users can select the shell with  
9885 which they are most comfortable.

9886 The construct `2>&1` is often used to redirect standard error to the same file as standard output.  
9887 Since the redirections take place beginning to end, the order of redirections is significant. For  
9888 example:

```
9889     ls > foo 2>&1
```

9890 directs both standard output and standard error to file **foo**. However:

```
9891     ls 2>&1 > foo
```

9892 only directs standard output to file **foo** because standard error was duplicated as standard  
9893 output before standard output was directed to file **foo**.

9894 The "<>" operator could be useful in writing an application that worked with several terminals,  
9895 and occasionally wanted to start up a shell. That shell would in turn be unable to run  
9896 applications that run from an ordinary controlling terminal unless it could make use of "<>"  
9897 redirection. The specific example is a historical version of the pager *more*, which reads from  
9898 standard error to get its commands, so standard input and standard output are both available  
9899 for their usual usage. There is no way of saying the following in the shell without "<>":

```
9900     cat food | more - >/dev/tty03 2<>/dev/tty03
```

9901 Another example of "<>" is one that opens `/dev/tty` on file descriptor 3 for reading and writing:

```
9902     exec 3<> /dev/tty
```

9903 An example of creating a lock file for a critical code region:

```
9904     set -C
9905     until    2> /dev/null > lockfile
9906     do      sleep 30
9907     done
9908     set +C
9909     perform critical function
9910     rm lockfile
```

9911 Since `/dev/null` is not a regular file, no error is generated by redirecting to it in *noclobber* mode.

9912 Tilde expansion is not performed on a here-document because the data is treated as if it were  
9913 enclosed in double quotes.

#### 9914 C.2.7.1 *Redirecting Input*

9915 There is no additional rationale provided for this section.

#### 9916 C.2.7.2 *Redirecting Output*

9917 There is no additional rationale provided for this section.

9918 *C.2.7.3 Appending Redirected Output*

9919 Note that when a file is opened (even with the `O_APPEND` flag set), the initial file offset for that  
9920 file is set to the beginning of the file. Some historic shells set the file offset to the current end-of-  
9921 file when append mode shell redirection was used, but this is not allowed by  
9922 IEEE Std 1003.1-200x.

9923 *C.2.7.4 Here-Document*

9924 There is no additional rationale provided for this section.

9925 *C.2.7.5 Duplicating an Input File Descriptor*

9926 There is no additional rationale provided for this section.

9927 *C.2.7.6 Duplicating an Output File Descriptor*

9928 There is no additional rationale provided for this section.

9929 *C.2.7.7 Open File Descriptors for Reading and Writing*

9930 There is no additional rationale provided for this section.

9931 **C.2.8 Exit Status and Errors**9932 *C.2.8.1 Consequences of Shell Errors*

9933 There is no additional rationale provided for this section.

9934 *C.2.8.2 Exit Status for Commands*

9935 There is a historical difference in *sh* and *ksh* non-interactive error behavior. When a command  
9936 named in a script is not found, some implementations of *sh* exit immediately, but *ksh* continues  
9937 with the next command. Thus, the Shell and Utilities volume of IEEE Std 1003.1-200x says that  
9938 the shell “may” exit in this case. This puts a small burden on the programmer, who has to test  
9939 for successful completion following a command if it is important that the next command not be  
9940 executed if the previous command was not found. If it is important for the command to have  
9941 been found, it was probably also important for it to complete successfully. The test for successful  
9942 completion would not need to change.

9943 Historically, shells have returned an exit status of  $128+n$ , where  $n$  represents the signal number.  
9944 Since signal numbers are not standardized, there is no portable way to determine which signal  
9945 caused the termination. Also, it is possible for a command to exit with a status in the same range  
9946 of numbers that the shell would use to report that the command was terminated by a signal.  
9947 Implementations are encouraged to choose exit values greater than 256 to indicate programs  
9948 that terminate by a signal so that the exit status cannot be confused with an exit status generated  
9949 by a normal termination.

9950 Historical shells make the distinction between “utility not found” and “utility found but cannot  
9951 execute” in their error messages. By specifying two seldomly used exit status values for these  
9952 cases, 127 and 126 respectively, this gives an application the opportunity to make use of this  
9953 distinction without having to parse an error message that would probably change from locale to  
9954 locale. The *command*, *env*, *nohup*, and *xargs* utilities in the Shell and Utilities volume of  
9955 IEEE Std 1003.1-200x have also been specified to use this convention.

9956 When a command fails during word expansion or redirection, most historical implementations  
9957 exit with a status of 1. However, there was some sentiment that this value should probably be

9958 much higher so that an application could distinguish this case from the more normal exit status  
 9959 values. Thus, the language “greater than zero” was selected to allow either method to be  
 9960 implemented.

## 9961 C.2.9 Shell Commands

9962 A description of an “empty command” was removed from an early proposal because it is only  
 9963 relevant in the cases of *sh -c " "*, *system(" ")*, or an empty shell-script file (such as the  
 9964 implementation of *true* on some historical systems). Since it is no longer mentioned in the Shell  
 9965 and Utilities volume of IEEE Std 1003.1-200x, it falls into the silently unspecified category of  
 9966 behavior where implementations can continue to operate as they have historically, but  
 9967 conforming applications do not construct empty commands. (However, note that *sh* does  
 9968 explicitly state an exit status for an empty string or file.) In an interactive session or a script with  
 9969 other commands, extra <newline>s or semicolons, such as;

```
9970     $ false
9971     $
9972     $ echo $?
9973     1
```

9974 would not qualify as the empty command described here because they would be consumed by  
 9975 other parts of the grammar.

### 9976 C.2.9.1 Simple Commands

9977 The enumerated list is used only when the command is actually going to be executed. For  
 9978 example, in:

```
9979     true || $foo *
```

9980 no expansions are performed.

9981 The following example illustrates both how a variable assignment without a command name  
 9982 affects the current execution environment, and how an assignment with a command name only  
 9983 affects the execution environment of the command:

```
9984     $ x=red
9985     $ echo $x
9986     red
9987     $ export x
9988     $ sh -c 'echo $x'
9989     red
9990     $ x=blue sh -c 'echo $x'
9991     blue
9992     $ echo $x
9993     red
```

9994 This next example illustrates that redirections without a command name are still performed:

```
9995     $ ls foo
9996     ls: foo: no such file or directory
9997     $ > foo
9998     $ ls foo
9999     foo
```

10000 A command without a command name, but one that includes a command substitution, has an  
 10001 exit status of the last command substitution that the shell performed. For example:

```

10002     if      x=$(command)
10003     then    ...
10004     fi

```

10005 An example of redirections without a command name being performed in a subshell shows that  
 10006 the here-document does not disrupt the standard input of the **while** loop:

```

10007     IFS=:
10008     while  read a b
10009     do     echo $a
10010           <<-eof
10011           Hello
10012           eof
10013     done </etc/passwd

```

10014 Some examples of commands without command names in AND-OR lists:

```

10015     > foo || {
10016         echo "error: foo cannot be created" >&2
10017         exit 1
10018     }
10019     # set saved if /vmunix.save exists
10020     test -f /vmunix.save && saved=1

```

10021 Command substitution and redirections without command names both occur in subshells, but  
 10022 they are not necessarily the same ones. For example, in:

```

10023     exec 3> file
10024     var=$(echo foo >&3) 3>&1

```

10025 it is unspecified whether **foo** is echoed to the file or to standard output.

### 10026 **Command Search and Execution**

10027 This description requires that the shell can execute shell scripts directly, even if the underlying  
 10028 system does not support the common "#!" interpreter convention. That is, if file **foo** contains  
 10029 shell commands and is executable, the following executes **foo**:

```

10030     ./foo

```

10031 The command search shown here does not match all historical implementations. A more typical  
 10032 sequence has been:

- 10033 • Any built-in (special or regular)
- 10034 • Functions
- 10035 • Path search for executable files

10036 But there are problems with this sequence. Since the programmer has no idea in advance which  
 10037 utilities might have been built into the shell, a function cannot be used to override portably a  
 10038 utility of the same name. (For example, a function named *cd* cannot be written for many  
 10039 historical systems.) Furthermore, the *PATH* variable is partially ineffective in this case, and only  
 10040 a pathname with a slash can be used to ensure a specific executable file is invoked.

10041 After the *execve()* failure described, the shell normally executes the file as a shell script. Some  
 10042 implementations, however, attempt to detect whether the file is actually a script and not an  
 10043 executable from some other architecture. The method used by the KornShell is allowed by the  
 10044 text that indicates non-text files may be bypassed.

10045 The sequence selected for the Shell and Utilities volume of IEEE Std 1003.1-200x acknowledges  
 10046 that special built-ins cannot be overridden, but gives the programmer full control over which  
 10047 versions of other utilities are executed. It provides a means of suppressing function lookup (via  
 10048 the *command* utility) for the user's own functions and ensures that any regular built-ins or  
 10049 functions provided by the implementation are under the control of the path search. The  
 10050 mechanisms for associating built-ins or functions with executable files in the path are not  
 10051 specified by the Shell and Utilities volume of IEEE Std 1003.1-200x, but the wording requires that  
 10052 if either is implemented, the application is not able to distinguish a function or built-in from an  
 10053 executable (other than in terms of performance, presumably). The implementation ensures that  
 10054 all effects specified by the Shell and Utilities volume of IEEE Std 1003.1-200x resulting from the  
 10055 invocation of the regular built-in or function (interaction with the environment, variables, traps,  
 10056 and so on) are identical to those resulting from the invocation of an executable file.

### 10057 Examples

10058 Consider three versions of the *ls* utility:

- 10059 1. The application includes a shell function named *ls*.
- 10060 2. The user writes a utility named *ls* and puts it in **/fred/bin**.
- 10061 3. The example implementation provides *ls* as a regular shell built-in that is invoked (either  
 10062 by the shell or directly by *exec*) when the path search reaches the directory **/posix/bin**.

10063 If *PATH*=**/posix/bin**, various invocations yield different versions of *ls*:

10064  
 10065  
 10066  
 10067  
 10068  
 10069  
 10070

Invocation	Version of <i>ls</i>
<i>ls</i> (from within application script)	(1) function
<i>command ls</i> (from within application script)	(3) built-in
<i>ls</i> (from within makefile called by application)	(3) built-in
<i>system("ls")</i>	(3) built-in
<i>PATH="/fred/bin:\$PATH" ls</i>	(2) user's version

### 10071 C.2.9.2 Pipelines

10072 Because pipeline assignment of standard input or standard output or both takes place before  
 10073 redirection, it can be modified by redirection. For example:

```
10074 $ command1 2>&1 | command2
```

10075 sends both the standard output and standard error of *command1* to the standard input of  
 10076 *command2*.

10077 The reserved word **!** allows more flexible testing using AND and OR lists.

10078 It was suggested that it would be better to return a non-zero value if any command in the  
 10079 pipeline terminates with non-zero status (perhaps the bitwise-inclusive OR of all return values).  
 10080 However, the choice of the last-specified command semantics are historical practice and would  
 10081 cause applications to break if changed. An example of historical behavior:

```
10082 $ sleep 5 | (exit 4)
10083 $ echo $?
10084 4
10085 $ (exit 4) | sleep 5
10086 $ echo $?
10087 0
```

## 10088 C.2.9.3 Lists

10089 The equal precedence of "&&" and "||" is historical practice. The standard developers  
 10090 evaluated the model used more frequently in high-level programming languages, such as C, to  
 10091 allow the shell logical operators to be used for complex expressions in an unambiguous way, but  
 10092 they could not allow historical scripts to break in the subtle way unequal precedence might  
 10093 cause. Some arguments were posed concerning the "{" or "(" groupings that are required  
 10094 historically. There are some disadvantages to these groupings:

- 10095 • The "(" can be expensive, as they spawn other processes on some implementations. This  
 10096 performance concern is primarily an implementation issue.
- 10097 • The "{" braces are not operators (they are reserved words) and require a trailing space  
 10098 after each '{', and a semicolon before each '}'. Most programmers (and certainly  
 10099 interactive users) have avoided braces as grouping constructs because of the problematic  
 10100 syntax required. Braces were not changed to operators because that would generate  
 10101 compatibility issues even greater than the precedence question; braces appear outside the  
 10102 context of a keyword in many shell scripts.

10103 IEEE PASC Interpretation 1003.2 #204 is applied, clarifying that the operators "&&" and "||"  
 10104 are evaluated with left associativity.

10105 **Asynchronous Lists**

10106 The grammar treats a construct such as:

```
10107     foo & bar & bam &
```

10108 as one "asynchronous list", but since the status of each element is tracked by the shell, the term  
 10109 "element of an asynchronous list" was introduced to identify just one of the **foo**, **bar**, or **bam**  
 10110 portions of the overall list.

10111 Unless the implementation has an internal limit, such as {CHILD\_MAX}, on the retained process  
 10112 IDs, it would require unbounded memory for the following example:

```
10113     while true
10114     do       foo & echo $!
10115     done
```

10116 The treatment of the signals SIGINT and SIGQUIT with asynchronous lists is described in the  
 10117 Shell and Utilities volume of IEEE Std 1003.1-200x, Section 2.11, Signals and Error Handling.

10118 Since the connection of the input to the equivalent of /dev/null is considered to occur before  
 10119 redirections, the following script would produce no output:

```
10120     exec < /etc/passwd
10121     cat <&0 &
10122     wait
```

10123 **Sequential Lists**

10124 There is no additional rationale provided for this section.

- 10125           **AND Lists**
- 10126           There is no additional rationale provided for this section.
- 10127           **OR Lists**
- 10128           There is no additional rationale provided for this section.
- 10129 *C.2.9.4 Compound Commands*
- 10130           **Grouping Commands**
- 10131           The semicolon shown in *{compound-list;}* is an example of a control operator delimiting the } reserved word. Other delimiters are possible, as shown in the Shell and Utilities volume of IEEE Std 1003.1-200x, Section 2.10, Shell Grammar; <newline> is frequently used.
- 10132
- 10133
- 10134           A proposal was made to use the <do-done> construct in all cases where command grouping in the current process environment is performed, identifying it as a construct for the grouping commands, as well as for shell functions. This was not included because the shell already has a grouping construct for this purpose ("{}"), and changing it would have been counter-productive.
- 10135
- 10136
- 10137
- 10138
- 10139           **For Loop**
- 10140           The format is shown with generous usage of <newline>s. See the grammar in the Shell and Utilities volume of IEEE Std 1003.1-200x, Section 2.10, Shell Grammar for a precise description of where <newline>s and semicolons can be interchanged.
- 10141
- 10142
- 10143           Some historical implementations support ' { ' and ' } ' as substitutes for **do** and **done**. The standard developers chose to omit them, even as an obsolescent feature. (Note that these substitutes were only for the **for** command; the **while** and **until** commands could not use them historically because they are followed by compound-lists that may contain "{ . . . }" grouping commands themselves.)
- 10144
- 10145
- 10146
- 10147
- 10148           The reserved word pair **do** . . . **done** was selected rather than **do** . . . **od** (which would have matched the spirit of **if** . . . **fi** and **case** . . . **esac**) because *od* is already the name of a standard utility.
- 10149
- 10150
- 10151           PASC Interpretation 1003.2 #169 has been applied changing the grammar.
- 10152           **Case Conditional Construct**
- 10153           An optional left parenthesis before *pattern* was added to allow numerous historical KornShell scripts to conform. At one time, using the leading parenthesis was required if the **case** statement was to be embedded within a "\$ ( )" command substitution; this is no longer the case with the POSIX shell. Nevertheless, many historical scripts use the left parenthesis, if only because it makes matching-parenthesis searching easier in *vi* and other editors. This is a relatively simple implementation change that is upward-compatible for all scripts.
- 10154
- 10155
- 10156
- 10157
- 10158
- 10159           Consideration was given to requiring *break* inside the *compound-list* to prevent falling through to the next pattern action list. This was rejected as being nonexistent practice. An interesting undocumented feature of the KornShell is that using ";&" instead of ";" as a terminator causes the exact opposite behavior—the flow of control continues with the next *compound-list*.
- 10160
- 10161
- 10162
- 10163           The pattern ' \* ', given as the last pattern in a **case** construct, is equivalent to the default case in a C-language **switch** statement.
- 10164

10165 The grammar shows that reserved words can be used as patterns, even if one is the first word on  
10166 a line. Obviously, the reserved word **esac** cannot be used in this manner.

### 10167 **If Conditional Construct**

10168 The precise format for the command syntax is described in the Shell and Utilities volume of  
10169 IEEE Std 1003.1-200x, Section 2.10, Shell Grammar.

### 10170 **While Loop**

10171 The precise format for the command syntax is described in the Shell and Utilities volume of  
10172 IEEE Std 1003.1-200x, Section 2.10, Shell Grammar.

### 10173 **Until Loop**

10174 The precise format for the command syntax is described in the Shell and Utilities volume of  
10175 IEEE Std 1003.1-200x, Section 2.10, Shell Grammar.

#### 10176 *C.2.9.5 Function Definition Command*

10177 The description of functions in an early proposal was based on the notion that functions should  
10178 behave like miniature shell scripts; that is, except for sharing variables, most elements of an  
10179 execution environment should behave as if they were a new execution environment, and  
10180 changes to these should be local to the function. For example, traps and options should be reset  
10181 on entry to the function, and any changes to them do not affect the traps or options of the caller.  
10182 There were numerous objections to this basic idea, and the opponents asserted that functions  
10183 were intended to be a convenient mechanism for grouping common commands that were to be  
10184 executed in the current execution environment, similar to the execution of the *dot* special built-  
10185 in.

10186 It was also pointed out that the functions described in that early proposal did not provide a local  
10187 scope for everything a new shell script would, such as the current working directory, or *umask*,  
10188 but instead provided a local scope for only a few select properties. The basic argument was that  
10189 if a local scope is needed for the execution environment, the mechanism already existed: the  
10190 application can put the commands in a new shell script and call that script. All historical shells  
10191 that implemented functions, other than the KornShell, have implemented functions that operate  
10192 in the current execution environment. Because of this, traps and options have a global scope  
10193 within a shell script. Local variables within a function were considered and included in another  
10194 early proposal (controlled by the special built-in *local*), but were removed because they do not fit  
10195 the simple model developed for functions and because there was some opposition to adding yet  
10196 another new special built-in that was not part of historical practice. Implementations should  
10197 reserve the identifier *local* (as well as *typeset*, as used in the KornShell) in case this local variable  
10198 mechanism is adopted in a future version of IEEE Std 1003.1-200x.

10199 A separate issue from the execution environment of a function is the availability of that function  
10200 to child shells. A few objectors maintained that just as a variable can be shared with child shells  
10201 by exporting it, so should a function. In early proposals, the *export* command therefore had a *-f*  
10202 flag for exporting functions. Functions that were exported were to be put into the environment  
10203 as *name()=value* pairs, and upon invocation, the shell would scan the environment for these and  
10204 automatically define these functions. This facility was strongly opposed and was omitted. Some  
10205 of the arguments against exportable functions were as follows:

- 10206 • There was little historical practice. The Ninth Edition shell provided them, but there was  
10207 controversy over how well it worked.

10208       • There are numerous security problems associated with functions appearing in the  
 10209       environment of a user and overriding standard utilities or the utilities owned by the  
 10210       application.

10211       • There was controversy over requiring *make* to import functions, where it has historically used  
 10212       an *exec* function for many of its command line executions.

10213       • Functions can be big and the environment is of a limited size. (The counter-argument was |  
 10214       that functions are no different from variables in terms of size: there can be big ones, and there |  
 10215       can be small ones—and just as one does not export huge variables, one does not export huge |  
 10216       functions. However, this might not apply to the average shell-function writer, who typically |  
 10217       writes much larger functions than variables.)

10218       As far as can be determined, the functions in the Shell and Utilities volume of  
 10219       IEEE Std 1003.1-200x match those in System V. Earlier versions of the KornShell had two  
 10220       methods of defining functions:

```
10221       function fname { compound-list } |
```

10222       and:

```
10223       fname() { compound-list } |
```

10224       The latter used the same definition as the Shell and Utilities volume of IEEE Std 1003.1-200x, but  
 10225       differed in semantics, as described previously. The current edition of the KornShell aligns the  
 10226       latter syntax with the Shell and Utilities volume of IEEE Std 1003.1-200x and keeps the former as  
 10227       is.

10228       The name space for functions is limited to that of a *name* because of historical practice.  
 10229       Complications in defining the syntactic rules for the function definition command and in dealing  
 10230       with known extensions such as the "@()" usage in the KornShell prevented the name space  
 10231       from being widened to a *word*. Using functions to support synonyms such as the "!!" and '%'  
 10232       usage in the C shell is thus disallowed to conforming applications, but acceptable as an |  
 10233       extension. For interactive users, the aliasing facilities in the Shell and Utilities volume of |  
 10234       IEEE Std 1003.1-200x should be adequate for this purpose. It is recognized that the name space |  
 10235       for utilities in the file system is wider than that currently supported for functions, if the portable |  
 10236       filename character set guidelines are ignored, but it did not seem useful to mandate extensions |  
 10237       in systems for so little benefit to conforming applications.

10238       The "()" in the function definition command consists of two operators. Therefore, intermixing  
 10239       <blank>s with the *fname*, '( ', and ' )' is allowed, but unnecessary.

10240       An example of how a function definition can be used wherever a simple command is allowed:

```
10241       # If variable i is equal to "yes", |
10242       # define function foo to be ls -l |
10243       # |
10244       [ "$i" = yes ] && foo() { |
10245           ls -l |
10246       }
```

10247 **C.2.10 Shell Grammar**

10248 There are several subtle aspects of this grammar where conventional usage implies rules about  
10249 the grammar that in fact are not true.

10250 For *compound\_list*, only the forms that end in a *separator* allow a reserved word to be recognized,  
10251 so usually only a *separator* can be used where a compound list precedes a reserved word (such as  
10252 **Then, Else, Do** and **Rbrace**). Explicitly requiring a separator would disallow such valid (if rare)  
10253 statements as:

```
10254     if (false) then (echo x) else (echo y) fi
```

10255 See the Note under special grammar rule 1.

10256 Concerning the third sentence of rule (1) (“Also, if the parser ...”):

10257 • This sentence applies rather narrowly: when a compound list is terminated by some clear  
10258 delimiter (such as the closing **fi** of an inner **if\_clause**) then it would apply; where the  
10259 compound list might continue (as in after a ‘;’), rule (7a) (and consequently the first  
10260 sentence of rule (1)) would apply. In many instances the two conditions are identical, but this  
10261 part of rule (1) does not give license to treating a **WORD** as a reserved word unless it is in a  
10262 place where a reserved word has to appear.

10263 • The statement is equivalent to requiring that when the LR(1) lookahead set contains exactly  
10264 one reserved word, it must be recognized if it is present. (Here “LR(1)” refers to the  
10265 theoretical concepts, not to any real parser generator.)

10266 For example, in the construct below, and when the parser is at the point marked with ‘^’,  
10267 the only next legal token is **then** (this follows directly from the grammar rules):

```
10268     if if...fi then ... fi
10269         ^
```

10270 At that point, the **then** must be recognized as a reserved word.

10271 (Depending on the parser generator actually used, “extra” reserved words may be in some  
10272 lookahead sets. It does not really matter if they are recognized, or even if any possible  
10273 reserved word is recognized in that state, because if it is recognized and is not in the  
10274 (theoretical) LR(1) lookahead set, an error is ultimately detected. In the example above, if  
10275 some other reserved word (for example, **while**) is also recognized, an error occurs later.

10276 This is approximately equivalent to saying that reserved words are recognized after other  
10277 reserved words (because it is after a reserved word that this condition occurs), but avoids the  
10278 “except for ...” list that would be required for **case**, **for**, and so on. (Reserved words are of  
10279 course recognized anywhere a *simple\_command* can appear, as well. Other rules take care of  
10280 the special cases of non-recognition, such as rule (4) for **case** statements.)

10281 Note that the body of here-documents are handled by token recognition (see the Shell and  
10282 Utilities volume of IEEE Std 1003.1-200x, Section 2.3, Token Recognition) and do not appear in  
10283 the grammar directly. (However, the here-document I/O redirection operator is handled as part  
10284 of the grammar.)

10285 The start symbol of the grammar (**complete\_command**) represents either input from the  
10286 command line or a shell script. It is repeatedly applied by the interpreter to its input and  
10287 represents a single “chunk” of that input as seen by the interpreter.

10288 *C.2.10.1 Shell Grammar Lexical Conventions*

10289 There is no additional rationale provided for this section.

10290 *C.2.10.2 Shell Grammar Rules*

10291 There is no additional rationale provided for this section.

10292 **C.2.11 Signals and Error Handling**

10293 There is no additional rationale provided for this section.

10294 **C.2.12 Shell Execution Environment**10295 Some implementations have implemented the last stage of a pipeline in the current environment |  
10296 so that commands such as: |10297 `command | read foo` |10298 set variable **foo** in the current environment. This extension is allowed, but not required;  
10299 therefore, a shell programmer should consider a pipeline to be in a subshell environment, but  
10300 not depend on it.10301 In early proposals, the description of execution environment failed to mention that each  
10302 command in a multiple command pipeline could be in a subshell execution environment. For  
10303 compatibility with some historical shells, the wording was phrased to allow an implementation  
10304 to place any or all commands of a pipeline in the current environment. However, this means that  
10305 a POSIX application must assume each command is in a subshell environment, but not depend  
10306 on it.10307 The wording about shell scripts is meant to convey the fact that describing “trap actions” can  
10308 only be understood in the context of the shell command language. Outside of this context, such  
10309 as in a C-language program, signals are the operative condition, not traps.10310 **C.2.13 Pattern Matching Notation**10311 Pattern matching is a simpler concept and has a simpler syntax than REs, as the former is  
10312 generally used for the manipulation of filenames, which are relatively simple collections of  
10313 characters, while the latter is generally used to manipulate arbitrary text strings of potentially  
10314 greater complexity. However, some of the basic concepts are the same, so this section points  
10315 liberally to the detailed descriptions in the Base Definitions volume of IEEE Std 1003.1-200x,  
10316 Chapter 9, Regular Expressions.10317 *C.2.13.1 Patterns Matching a Single Character*10318 Both quoting and escaping are described here because pattern matching must work in three  
10319 separate circumstances:

- 10320 1. Calling directly upon the shell, such as in pathname expansion or in a
- case**
- statement. All
- 
- 10321 of the following match the string or file
- abc**
- :

10322 `abc "abc" a"b" c a\bc a[b]c a["b"]c a[\b]c a["\b"]c a?c a*c` |

10323 The following do not:

10324 `"a?c" a\*c a\[b]c` |

- 10325 2. Calling a utility or function without going through a shell, as described for
- find*
- and the
- 
- 10326
- fnmatch()*
- function defined in the System Interfaces volume of IEEE Std 1003.1-200x.

10327 3. Calling utilities such as *find*, *cpio*, *tar*, or *pax* through the shell command line. In this case,  
10328 shell quote removal is performed before the utility sees the argument. For example, in:

```
10329 find /bin -name "e\c[\h]o" -print
```

10330 after quote removal, the backslashes are presented to *find* and it treats them as escape  
10331 characters. Both precede ordinary characters, so the *c* and *h* represent themselves and *echo*  
10332 would be found on many historical systems (that have it in **/bin**). To find a filename that  
10333 contained shell special characters or pattern characters, both quoting and escaping are  
10334 required, such as:

```
10335 pax -r ... "*a\(\?"
```

10336 to extract a filename ending with "a(?".

10337 Conforming applications are required to quote or escape the shell special characters (sometimes  
10338 called metacharacters). If used without this protection, syntax errors can result or  
10339 implementation extensions can be triggered. For example, the KornShell supports a series of  
10340 extensions based on parentheses in patterns.

10341 The restriction on a circumflex in a bracket expression is to allow implementations that support  
10342 pattern matching using the circumflex as the negation character in addition to the exclamation  
10343 mark. A conforming application must use something like "[\^!]" to match either character.

#### 10344 C.2.13.2 Patterns Matching Multiple Characters

10345 Since each asterisk matches zero or more occurrences, the patterns "a\*b" and "a\*\*b" have  
10346 identical functionality.

#### 10347 Examples

10348 a[bc] Matches the strings "ab" and "ac".

10349 a\*d Matches the strings "ad", "abd", and "abcd", but not the string "abc".

10350 a\*d\* Matches the strings "ad", "abcd", "abcdef", "aaaad", and "adddd".

10351 \*a\*d Matches the strings "ad", "abcd", "efabcd", "aaaad", and "adddd".

#### 10352 C.2.13.3 Patterns Used for Filename Expansion

10353 The caveat about a slash within a bracket expression is derived from historical practice. The  
10354 pattern "a[b/c]d" does not match such pathnames as **abd** or **a/d**. On some implementations  
10355 (including those conforming to the Single UNIX Specification), it matched a pathname of  
10356 literally "a[b/c]d". On other systems, it produced an undefined condition (an unescaped '['  
10357 used outside a bracket expression). In this version, the XSI behavior is now required.

10358 Filenames beginning with a period historically have been specially protected from view on  
10359 UNIX systems. A proposal to allow an explicit period in a bracket expression to match a leading  
10360 period was considered; it is allowed as an implementation extension, but a conforming  
10361 application cannot make use of it. If this extension becomes popular in the future, it will be  
10362 considered for a future version of the Shell and Utilities volume of IEEE Std 1003.1-200x.

10363 Historical systems have varied in their permissions requirements. To match **f\*/bar** has required  
10364 read permissions on the **f\*** directories in the System V shell, but the Shell and Utilities volume of  
10365 IEEE Std 1003.1-200x, the C shell, and KornShell require only search permissions.

**10366 C.2.14 Special Built-In Utilities**

10367 See the RATIONALE sections on the individual reference pages.

**10368 C.3 Batch Environment Services and Utilities****10369 Scope of the Batch Environment Option**

10370 This section summarizes the deliberations of the IEEE P1003.15 (Batch Environment) working  
10371 group in the development of the Batch Environment option, which covers a set of services and  
10372 utilities defining a batch processing system.

10373 This informative section contains historical information concerning the contents of the  
10374 amendment and describes why features were included or discarded by the working group.

**10375 History of Batch Systems**

10376 The supercomputing technical committee began as a “Birds Of a Feather” (BOF) at the January  
10377 1987 Usenix meeting. There was enough general interest to form a supercomputing attachment  
10378 to the /usr/group working groups. Several subgroups rapidly formed. Of those subgroups, the  
10379 batch group was the most ambitious. The first early meetings were spent evaluating user needs  
10380 and existing batch implementations.

10381 To evaluate user needs, individuals from the supercomputing community came and presented  
10382 their needs. Common requests were flexibility, interoperability, control of resources, and ease-  
10383 of-use. Backwards-compatibility was not an issue. The working group then evaluated some  
10384 existing systems. The following different systems were evaluated:

- 10385 • PROD
- 10386 • Convex Distributed Batch
- 10387 • NQS
- 10388 • CTSS
- 10389 • MDQS from Ballistics Research Laboratory (BRL)

10390 Finally, NQS was chosen as a model because it satisfied not only the most user requirements, but  
10391 because it was public domain, already implemented on a variety of hardware platforms, and  
10392 networked-based.

**10393 Historical Implementations of Batch Systems**

10394 Deferred processing of work under the control of a scheduler has been a feature of most  
10395 proprietary operating systems from the earliest days of multi-user systems in order to maximize  
10396 utilization of the computer.

10397 The arrival of UNIX systems proved to be a dilemma to many hardware providers and users  
10398 because it did not include the sophisticated batch facilities offered by the proprietary systems.  
10399 This omission was rectified in 1986 by NASA Ames Research Center who developed the  
10400 Network Queuing System (NQS) as a portable UNIX application that allowed the routing and  
10401 processing of batch “jobs” in a network. To encourage its usage, the product was later put into  
10402 the public domain. It was promptly picked up by UNIX hardware providers, and ported and  
10403 developed for their respective hardware and UNIX implementations.

10404 Many major vendors, who traditionally offer a batch-dominated environment, ported the  
 10405 public-domain product to their systems, customized it to support the capabilities of their  
 10406 systems, and added many customer-requested features.

10407 Due to the strong hardware provider and customer acceptance of NQS, it was decided to use  
 10408 NQS as the basis for the POSIX Batch Environment amendment in 1987. Other batch systems  
 10409 considered at the time included CTSS, MDQS (a forerunner of NQS from the Ballistics Research  
 10410 Laboratory), and PROD (a Los Alamos Labs development). None were thought to have both the  
 10411 functionality and acceptability of NQS.

#### 10412 **NQS Differences from the *at* utility**

10413 The base standard *at* and *batch* utilities are not sufficient to meet the batch processing needs in a  
 10414 supercomputing environment and additional functionality in the areas of resource management,  
 10415 job scheduling, system management, and control of output is required.

#### 10416 **Batch Environment Option Definitions**

10417 The concept of a batch job is closely related to a session with a session leader. The main  
 10418 difference is that a batch job does not have a controlling terminal. There has been much debate  
 10419 over whether to use the term *request* or *job*. *Job* was the final choice because of the historical use  
 10420 of this term in the batch environment.

10421 The current definition for job identifiers is not sufficient with the model of destinations. The  
 10422 current definition is:

```
10423     sequence_number.originating_host
```

10424 Using the model of destination, a host may include multiple batch nodes, the location of which is  
 10425 identified uniquely by a name or directory service. If the current definition is used, batch nodes  
 10426 running on the same host would have to coordinate their use of sequence numbers, as sequence  
 10427 numbers are assigned by the originating host. The alternative is to use the originating batch node  
 10428 name instead of the originating host name.

10429 The reasons for wishing to run more than one batch system per host could be the following:

10430 A test and production batch system are maintained on a single host. This is most likely in a  
 10431 development facility, but could also arise when a site is moving from one version to another.  
 10432 The new batch system could be installed as a test version that is completely separate from the  
 10433 production batch system, so that problems can be isolated to the test system. Requiring the batch  
 10434 nodes to coordinate their use of sequence numbers creates a dependency between the two  
 10435 nodes, and that defeats the purpose of running two nodes.

10436 A site has multiple departments using a single host, with different management policies. An  
 10437 example of contention might be in job selection algorithms. One group might want a FIFO type  
 10438 of selection, while another group wishes to use a more complex algorithm based on resource  
 10439 availability. Again, requiring the batch nodes to coordinate is an unnecessary binding.

10440 The proposal eventually accepted was to replace originating host with originating batch node.  
 10441 This supplies sufficient granularity to ensure unique job identifiers. If more than one batch node  
 10442 is on a particular host, they each have their own unique name.

10443 The queue portion of a destination is not part of the job identifier as these are not required to be  
 10444 unique between batch nodes. For instance, two batch nodes may both have queues called small,  
 10445 medium, and large. It is only the batch node name that is uniquely identifiable throughout the  
 10446 batch system. The queue name has no additional function in this context.

10447 Assume there are three batch nodes, each of which has its own name server. On batch node one,  
10448 there are no queues. On batch node two, there are fifty queues. On batch node three, there are  
10449 forty queues. The system administrator for batch node one does not have to configure queues,  
10450 because there are none implemented. However, if a user wishes to send a job to either batch  
10451 node two or three, the system administrator for batch node one must configure a destination  
10452 that maps to the appropriate batch node and queue. If every queue is to be made accessible from  
10453 batch node one, the system administrator has to configure ninety destinations.

10454 To avoid requiring this, there should be a mechanism to allow a user to separate the destination  
10455 into a batch node name and a queue name. Then, an implementation that is configured to get to  
10456 all the batch nodes does not need any more configuration to allow a user to get to all of the  
10457 queues on all of the batch nodes. The node name is used to locate the batch node, while the  
10458 queue name is sent unchanged to that batch node.

10459 The following are requirements that a destination identifier must be capable of providing:

- 10460 • The ability to direct a job to a queue in a particular batch node.
- 10461 • The ability to direct a job to a particular batch node.
- 10462 • The ability to group at a higher level than just one queue. This includes grouping similar  
10463 queues across multiple batch nodes (this is a pipe queue today).
- 10464 • The ability to group batch nodes. This allows a user to submit a job to a group name with no  
10465 knowledge of the batch node configuration. This also provides aliasing as a special case.  
10466 Aliasing is a group containing only one batch node name. The group name is the alias.

10467 In addition, the administrator has the following requirements:

- 10468 • The ability to control access to the queues.
- 10469 • The ability to control access to the batch nodes.
- 10470 • The ability to control access to groups of queues (pipe queues).
- 10471 • The ability to configure retry time intervals and durations.

10472 The requirements of the user are met by destination as explained in the following:

10473 The user has the ability to specify a queue name, which is known only to the batch node  
10474 specified. There is no configuration of these queues required on the submitting node.

10475 The user has the ability to specify a batch node whose name is network-unique. The  
10476 configuration required is that the batch node be defined as an application, just as other  
10477 applications such as FTP are configured.

10478 Once a job reaches a queue, it can again become a user of the batch system. The batch node can  
10479 choose to send the job to another batch node or queue or both. In other words, the routing is at  
10480 an application level, and it is up to the batch system to choose where the job will be sent.  
10481 Configuration is up to the batch node where the queue resides. This provides grouping of  
10482 queues across batch nodes or within a batch node. The user submits the job to a queue, which by  
10483 definition routes the job to other queues or nodes or both.

10484 A node name may be given to a naming service, which returns multiple addresses as opposed to  
10485 just one. This provides grouping at a batch node level. This is a local issue, meaning that the  
10486 batch node must choose only one of these addresses. The list of addresses is not sent with the  
10487 job, and once the job is accepted on another node, there is no connection between the list and the  
10488 job. The requirements of the administrator are met by destination as explained in the following:

10489 The control of queues is a batch system issue, and will be done using the batch administrative  
10490 utilities.

- 10491 The control of nodes is a network issue, and will be done through whatever network facilities  
10492 are available.
- 10493 The control of access to groups of queues (pipe queues) is covered by the control of any other  
10494 queue. The fact that the job may then be sent to another destination is not relevant.
- 10495 The propagation of a job across more than one point-to-point connection was dropped because  
10496 of its complexity and because all of the issues arising from this capability could not be resolved.  
10497 It could be provided as additional functionality at some time in the future.
- 10498 The addition of *network* as a defined term was done to clarify the difference between a network  
10499 of batch nodes as opposed to a network of hosts. A network of batch nodes is referred to as a  
10500 batch system. The network refers to the actual host configuration. A single host may have  
10501 multiple batch nodes.
- 10502 In the absence of a standard network naming convention, this option establishes its own  
10503 convention for the sake of consistency and expediency. This is subject to change, should a future  
10504 working group develop a standard naming convention for network pathnames.

### 10505 C.3.1 Batch General Concepts

- 10506 During the development of the Batch Environment option, a number of topics were discussed at  
10507 length which influenced the wording of the normative text but could not be included in the final  
10508 text. The following items are some of the most significant terms and concepts of those discussed:
- 10509 • Small and Consistent Command Set
 

10510 Often, conventional utilities from UNIX systems have a very complicated utility syntax and  
10511 usage. This can often result in confusion and errors when trying to use them. The Batch  
10512 Environment option utility set, on the other hand, has been paired to a small set of robust  
10513 utilities with an orthogonal calling sequence.
  - 10514 • Checkpoint/Restart
 

10515 This feature permits an already executing process to checkpoint or save its contents. Some  
10516 implementations permit this at both the batch utility level; for example, checkpointing this  
10517 job upon its abnormal termination or from within the job itself via a system call. Support of  
10518 checkpoint/restart is optional. A conscious, careful effort was made to make the *qsub* and  
10519 *qmgr* utilities consistently refer to checkpoint/restart as optional functionality.
  - 10520 • Rerunability
 

10521 When a user submits a job for batch processing, they can designate it “rerunnable” in that it  
10522 will automatically resume execution from the start of the job if the machine on which it was  
10523 executing crashes for some reason. The decision on whether the job will be rerun or not is  
10524 entirely up to the submitter of the job and no decisions will be made within the batch system.  
10525 A job that is rerunnable and has been submitted with the proper checkpoint/restart switch  
10526 will first be checkpointed and execution begun from that point. Furthermore, use of the  
10527 implementation-defined checkpoint/restart feature will be not be defined in this context.
  - 10528 • Error Codes
 

10529 All utilities exit with error status zero (0) if successful, one (1) if a user error occurred, and  
10530 two (2) for an internal Batch Environment option error.
  - 10531 • Level of Portability
 

10532 Portability is specified at both the user, operator, and administrator levels. A conforming  
10533 batch implementation prevents identical functionality and behavior at all these levels.  
10534 Additionally, portable batch shell scripts with embedded Batch Environment option utilities

- 10535 adds an additional level of portability.
- 10536
- Resource Specification
- 10537 A small set of globally understood resources, such as memory and CPU time, is specified. All  
10538 conforming batch implementations are able to process them in a manner consistent with the  
10539 yet-to-be-developed resource management model. Resources not in this amendment set are  
10540 ignored and passed along as part of the argument stream of the utility.
- 10541
- Queue Position
- 10542 Queue position is the place a job occupies in a queue. It is dependent on a variety of factors  
10543 such as submission time and priority. Since priority may be affected by the implementation  
10544 of fair share scheduling, the definition of queue position is implementation-defined.
- 10545
- Queue ID
- 10546 A numerical queue ID is an external requirement for purposes of accounting. The  
10547 identification number was chosen over queue name for processing convenience.
- 10548
- Job ID
- 10549 A common notion of “jobs” is a collection of processes whose process group cannot be  
10550 altered and is used for resource management and accounting. This concept is  
10551 implementation-defined and, as such, has been omitted from the batch amendment.
- 10552
- Bytes *versus* Words
- 10553 Except for one case, bytes are used as the standard unit for memory size. Furthermore, the  
10554 definition of a word varies from machine to machine. Therefore, bytes will be the default unit  
10555 of memory size.
- 10556
- Regular Expressions
- 10557 The standard definition of regular expressions is much too broad to be used in the batch  
10558 utility syntax. All that is needed is a simple concept of “all”; for example, delete all my jobs  
10559 from the named queue. For this reason, regular expressions have been eliminated from the  
10560 batch amendment.
- 10561
- Display Privacy
- 10562 How much data should be displayed locally through functions? Local policy dictates the  
10563 amount of privacy. Library functions must be used to create and enforce local policy.  
10564 Network and local *qstats* must reflect the policy of the server machine.
- 10565
- Remote Host Naming Convention
- 10566 It was decided that host names would be a maximum of 255 characters in length, with at  
10567 most 15 characters being shown in displays. The 255 character limit was chosen because it is  
10568 consistent with BSD. The 15-character limit was an arbitrary decision.
- 10569
- Network Administration
- 10570 Network administration is important, but is outside the scope of the batch amendment.  
10571 Network administration could done with *rsh*. However, authentication becomes two-sided.
- 10572
- Network Administration Philosophy
- 10573 Keep it simple. Centralized management should be possible. For example, Los Alamos needs  
10574 a dumb set of CPUs to be managed by a central system *versus* several independently-  
10575 managed systems as is the general case for the Batch Environment option.

- 10576 • Operator Utility Defaults (that is, Default Host, User, Account, and so on)
- 10577 It was decided that usability would override orthogonality and syntactic consistency.
- 10578 • The Batch System Manager and Operator Distinction
- 10579 The distinction between manager and operator is that operators can only control the flow of
- 10580 jobs. A manager can alter the batch system configuration in addition to job flow. POSIX
- 10581 makes a distinction between user and system administrator but goes no further. The
- 10582 concepts of manager and operator privileges fall under local policy. The distinction between
- 10583 manager and operator is historical in batch environments, and the Batch Environment option
- 10584 has continued that distinction.
- 10585 • The Batch System Administrator
- 10586 An administrator is equivalent to a batch system manager.

### 10587 C.3.2 Batch Services

- 10588 This rationale is provided as informative rather than normative text, to avoid placing  
 10589 requirements on implementors regarding the use of symbolic constants, but at the same time to  
 10590 give implementors a preferred practice for assigning values to these constants to promote  
 10591 interoperability.
- 10592 The *Checkpoint* and *Minimum\_Cpu\_Interval* attributes induce a variety of behavior depending  
 10593 upon their values. Some jobs cannot or should not be checkpointed. Other users will simply  
 10594 need to ensure job continuation across planned downtimes; for example, scheduled preventive  
 10595 maintenance. For users consuming expensive resources, or for jobs that run longer than the  
 10596 mean time between failures, however, periodic checkpointing may be essential. However,  
 10597 system administrators must be able to set minimum checkpoint intervals on a queue-by-queue  
 10598 basis to guard against; for example, naive users specifying interval values too small on memory  
 10599 intensive jobs. Otherwise, system overhead would adversely affect performance.
- 10600 The use of symbolic constants, such as `NO_CHECKPOINT`, was introduced to lend a degree of  
 10601 formalism and portability to this option.
- 10602 Support for checkpointing is optional for servers. However, clients must provide for the `-c`  
 10603 option, since in a distributed environment the job may run on a server that does provide such  
 10604 support, even if the host of the client does not support the checkpoint feature.
- 10605 If the user does not specify the `-c` option, the default action is left unspecified by this option.  
 10606 Some implementations may wish to do checkpointing by default; others may wish to checkpoint  
 10607 only under an explicit request from the user.
- 10608 The *Priority* attribute has been made non-optional. All clients already had been required to  
 10609 support the `-p` option. The concept of prioritization is common in historical implementations.  
 10610 The default priority is left to the server to establish.
- 10611 The *Hold\_Types* attribute has been modified to allow for implementation-defined hold types to  
 10612 be passed to a batch server.
- 10613 It was the intent of the IEEE P1003.15 working group to mandate the support for the  
 10614 *Resource\_List* attribute in this option by referring to another amendment, specifically P1003.1a.  
 10615 However, during the development of P1003.1a this was excluded. As such this requirement has  
 10616 been removed from the normative text.
- 10617 The *Shell\_Path* attribute has been modified to accept a list of shell paths that are associated with  
 10618 a host. The name of the attribute has been changed to *Shell\_Path\_List*.

10619 **C.3.3 Common Behavior for Batch Environment Utilities**

10620 This section was defined to meet the goal of a “Small and Consistent Command Set” for this  
10621 option.

10622 **C.4 Utilities**

10623 For the utilities included in IEEE Std 1003.1-200x, see the RATIONALE sections on the individual  
10624 reference pages.

10625 **Exclusion of Utilities**

10626 The set of utilities contained in IEEE Std 1003.1-200x is drawn from the base documents, with  
10627 one addition: the *c99* utility. This section contains rationale for some of the deliberations that led  
10628 to this set of utilities, and why certain utilities were excluded.

10629 Many utilities were evaluated by the standard developers; more historical utilities were  
10630 excluded from the base documents than included. The following list contains many common  
10631 UNIX system utilities that were not included as mandatory utilities, in the UPE, in the XSI  
10632 extension, or in one of the software development groups. It is logistically difficult for this  
10633 rationale to distribute correctly the reasons for not including a utility among the various utility  
10634 options. Therefore, this section covers the reasons for all utilities not included in  
10635 IEEE Std 1003.1-200x.

10636 This rationale is limited to a discussion of only those utilities actively or indirectly evaluated by  
10637 the standard developers of the base documents, rather than the list of all known UNIX utilities  
10638 from all its variants.

10639 *adb* The intent of the various software development utilities was to assist in the  
10640 installation (rather than the actual development and debugging) of applications.  
10641 This utility is primarily a debugging tool. Furthermore, many useful aspects of *adb*  
10642 are very hardware-specific.

10643 *as* Assemblers are hardware-specific and are included implicitly as part of the  
10644 compilers in IEEE Std 1003.1-200x.

10645 *banner* The only known use of this command is as part of the *lp* printer header pages. It  
10646 was decided that the format of the header is implementation-defined, so this utility  
10647 is superfluous to application portability.

10648 *calendar* This reminder service program is not useful to conforming applications. |

10649 *cancel* The *lp* (line printer spooling) system specified is the most basic possible and did  
10650 not need this level of application control.

10651 *chroot* This is primarily of administrative use, requiring superuser privileges.

10652 *col* No utilities defined in IEEE Std 1003.1-200x produce output requiring such a filter.  
10653 The *nroff* text formatter is present on many historical systems and will continue to  
10654 remain as an extension; *col* is expected to be shipped by all the systems that ship  
10655 *nroff*.

10656 *cpio* This has been replaced by *pax*, for reasons explained in the rationale for that utility.

10657 *cpp* This is subsumed by *c99*.

10658 *cu* This utility is terminal-oriented and is not useful from shell scripts or typical  
10659 application programs. |

10660	<i>dc</i>	The functionality of this utility can be provided by the <i>bc</i> utility; <i>bc</i> was selected because it was easier to use and had superior functionality. Although the historical versions of <i>bc</i> are implemented using <i>dc</i> as a base, IEEE Std 1003.1-200x prescribes the interface and not the underlying mechanism used to implement it.
10661		
10662		
10663		
10664	<i>dircmp</i>	Although a useful concept, the historical output of this directory comparison program is not suitable for processing in application programs. Also, the <i>diff -r</i> command gives equivalent functionality.
10665		
10666		
10667		
10667	<i>dis</i>	Disassemblers are hardware-specific.
10668	<i>emacs</i>	The community of <i>emacs</i> editing enthusiasts was adamant that the full <i>emacs</i> editor not be included in the base documents because they were concerned that an attempt to standardize this very powerful environment would encourage vendors to ship versions conforming strictly to the standard, but lacking the extensibility required by the community. The author of the original <i>emacs</i> program also expressed his desire to omit the program. Furthermore, there were a number of historical UNIX systems that did not include <i>emacs</i> , or included it without supporting it, but there were very few that did not include and support <i>vi</i> .
10669		
10670		
10671		
10672		
10673		
10674		
10675		
10676	<i>ld</i>	This is subsumed by <i>c99</i> .
10677	<i>line</i>	The functionality of <i>line</i> can be provided with <i>read</i> .
10678	<i>lint</i>	This technology is partially subsumed by <i>c99</i> . It is also hard to specify the degree of checking for possible error conditions in programs in any compiler, and specifying what <i>lint</i> would do in these cases is equally difficult.
10679		
10680		
10681		
10682		It is fairly easy to specify what a compiler does. It requires specifying the language, what it does with that language, and stating that the interpretation of any incorrect program is unspecified. Unfortunately, any description of <i>lint</i> is required to specify what to do with erroneous programs. Since the number of possible errors and questionable programming practices is infinite, one cannot require <i>lint</i> to detect all errors of any given class.
10683		
10684		
10685		
10686		
10687		
10687		Additionally, some vendors complained that since many compilers are distributed in a binary form without a <i>lint</i> facility (because the ISO C standard does not require one), implementing the standard as a stand-alone product will be much harder. Rather than being able to build upon a standard compiler component (simply by providing <i>c99</i> as an interface), source to that compiler would most likely need to be modified to provide the <i>lint</i> functionality. This was considered a major burden on system providers for a very small gain to developers (users).
10688		
10689		
10690		
10691		
10692		
10693		
10694	<i>login</i>	This utility is terminal-oriented and is not useful from shell scripts or typical application programs.
10695		
10696	<i>lorder</i>	This utility is an aid in creating an implementation-defined detail of object libraries that the standard developers did not feel required standardization.
10697		
10698	<i>lpstat</i>	The <i>lp</i> system specified is the most basic possible and did not need this level of application control.
10699		
10700	<i>mail</i>	This utility was omitted in favor of <i>mailx</i> because there was a considerable functionality overlap between the two.
10701		
10702	<i>mknod</i>	This was omitted in favor of <i>mkfifo</i> , as <i>mknod</i> has too many implementation-defined functions.
10703		

10704	<i>news</i>	This utility is terminal-oriented and is not useful from shell scripts or typical application programs.
10705		
10706	<i>pack</i>	This compression program was considered inferior to <i>compress</i> .
10707	<i>passwd</i>	This utility was proposed in a historical draft of the base documents but met with too many objections to be included. There were various reasons:
10708		
10709		<ul style="list-style-type: none"> <li>• Changing a password should not be viewed as a command, but as part of the login sequence. Changing a password should only be done while a trusted path is in effect.</li> </ul>
10710		
10711		
10712		<ul style="list-style-type: none"> <li>• Even though the text in early drafts was intended to allow a variety of implementations to conform, the security policy for one site may differ from another site running with identical hardware and software. One site might use password authentication while the other did not. Vendors could not supply a <i>passwd</i> utility that would conform to IEEE Std 1003.1-200x for all sites using their system.</li> </ul>
10713		
10714		
10715		
10716		
10717		
10718		<ul style="list-style-type: none"> <li>• This is really a subject for a system administration working group or a security working group.</li> </ul>
10719		
10720	<i>pcat</i>	This compression program was considered inferior to <i>zcat</i> .
10721	<i>pg</i>	This duplicated many of the features of the <i>more</i> pager, which was preferred by the standard developers.
10722		
10723	<i>prof</i>	The intent of the various software development utilities was to assist in the installation (rather than the actual development and debugging) of applications. This utility is primarily a debugging tool.
10724		
10725		
10726	RCS	RCS was originally considered as part of a version control utilities portion of the scope. However, this aspect was abandoned by the standard developers. SCCS is now included as an optional part of the XSI extension.
10727		
10728		
10729	<i>red</i>	Restricted editor. This was not considered by the standard developers because it never provided the level of security restriction required.
10730		
10731	<i>rsh</i>	Restricted shell. This was not considered by the standard developers because it does not provide the level of security restriction that is implied by historical documentation.
10732		
10733		
10734	<i>sdb</i>	The intent of the various software development utilities was to assist in the installation (rather than the actual development and debugging) of applications. This utility is primarily a debugging tool. Furthermore, some useful aspects of <i>sdb</i> are very hardware-specific.
10735		
10736		
10737		
10738	<i>sdiff</i>	The “side-by-side <i>diff</i> ” utility from System V was omitted because it is used infrequently, and even less so by conforming applications. Despite being in System V, it is not in the SVID or XPG.
10739		
10740		
10741	<i>shar</i>	Any of the numerous “shell archivers” were excluded because they did not meet the requirement of existing practice.
10742		
10743	<i>shl</i>	This utility is terminal-oriented and is not useful from shell scripts or typical application programs. The job control aspects of the shell command language are generally more useful.
10744		
10745		
10746	<i>size</i>	The intent of the various software development utilities was to assist in the installation (rather than the actual development and debugging) of applications.
10747		

10748		This utility is primarily a debugging tool.
10749	<i>spell</i>	This utility is not useful from shell scripts or typical application programs. The <i>spell</i> utility was considered, but was omitted because there is no known technology that can be used to make it recognize general language for user-specified input without providing a complete dictionary along with the input file.
10750		
10751		
10752		
10753	<i>su</i>	This utility is not useful from shell scripts or typical application programs. (There was also sentiment to avoid security-related utilities.)
10754		
10755	<i>sum</i>	This utility was renamed <i>cksum</i> .
10756	<i>tar</i>	This has been replaced by <i>pax</i> , for reasons explained in the rationale for that utility.
10757	<i>tsort</i>	This utility is an aid in creating an implementation-defined detail of object libraries that the standard developers did not feel required standardization.
10758		
10759	<i>unpack</i>	This compression program was considered inferior to <i>uncompress</i> .
10760	<i>wall</i>	This utility is terminal-oriented and is not useful from shell scripts or typical application programs. It is generally used only by system administrators.
10761		



10763 / *Rationale (Informative)*

10764 **Part D:**

10765 **Portability Considerations**

10766 *The Open Group*

10767 *The Institute of Electrical and Electronics Engineers, Inc.*



## Portability Considerations (Informative)

10769

10770 This section contains information to satisfy various international requirements:

- 10771 • Section D.1 describes perceived user requirements.
- 10772 • Section D.2 (on page 3558) indicates how the facilities of IEEE Std 1003.1-200x satisfy those  
10773 requirements.
- 10774 • Section D.3 (on page 3565) offers guidance to writers of profiles on how the configurable  
10775 options, limits, and optional behavior of IEEE Std 1003.1-200x should be cited in profiles.

### 10776 D.1 User Requirements

10777 This section describes the user requirements that were perceived by the developers of  
10778 IEEE Std 1003.1-200x. The primary source for these requirements was an analysis of historical  
10779 practice in widespread use, as typified by the base documents listed in Section A.1.1 (on page  
10780 3293).

10781 IEEE Std 1003.1-200x addresses the needs of users requiring open systems solutions for source  
10782 code portability of applications. It currently addresses users requiring open systems solutions  
10783 for source-code portability of applications involving multi-programming and process  
10784 management (creating processes, signaling, and so on); access to files and directories in a  
10785 hierarchy of file systems (opening, reading, writing, deleting files, and so on); access to  
10786 asynchronous communications ports and other special devices; access to information about  
10787 other users of the system; facilities supporting applications requiring bounded (realtime)  
10788 response.

10789 The following users are identified for IEEE Std 1003.1-200x:

- 10790 • Those employing applications written in high-level languages, such as C, Ada, or FORTRAN.
- 10791 • Users who desire conforming applications that do not necessarily require the characteristics |  
10792 of high-level languages (for example, the speed of execution of compiled languages or the  
10793 relative security of source code intellectual property inherent in the compilation process).
- 10794 • Users who desire conforming applications that can be developed quickly and can be |  
10795 modified readily without the use of compilers and other system components that may be  
10796 unavailable on small systems or those without special application development capabilities.
- 10797 • Users who interact with a system to achieve general-purpose time-sharing capabilities  
10798 common to most business or government offices or academic environments: editing, filing,  
10799 inter-user communications, printing, and so on.
- 10800 • Users who develop applications for POSIX-conformant systems.
- 10801 • Users who develop applications for UNIX systems.

10802 An acknowledged restriction on applicable users is that they are limited to the group of  
10803 individuals who are familiar with the style of interaction characteristic of historically-derived  
10804 systems based on one of the UNIX operating systems (as opposed to other historical systems  
10805 with different models, such as MS/DOS, Macintosh, VMS, MVS, and so on). Typical users  
10806 would include program developers, engineers, or general-purpose time-sharing users.

10807 The requirements of users of IEEE Std 1003.1-200x can be summarized as a single goal:  
10808 *application source portability*. The requirements of the user are stated in terms of the requirements  
10809 of portability of applications. This in turn becomes a requirement for a standardized set of  
10810 syntax and semantics for operations commonly found on many operating systems.

10811 The following sections list the perceived requirements for application portability.

#### 10812 **D.1.1 Configuration Interrogation**

10813 An application must be able to determine whether and how certain optional features are  
10814 provided and to identify the system upon which it is running, so that it may appropriately adapt  
10815 to its environment.

10816 Applications must have sufficient information to adapt to varying behaviors of the system.

#### 10817 **D.1.2 Process Management**

10818 An application must be able to manage itself, either as a single process or as multiple processes.  
10819 Applications must be able to manage other processes when appropriate.

10820 Applications must be able to identify, control, create, and delete processes, and there must be  
10821 communication of information between processes and to and from the system.

10822 Applications must be able to use multiple flows of control with a process (threads) and  
10823 synchronize operations between these flows of control.

#### 10824 **D.1.3 Access to Data**

10825 Applications must be able to operate on the data stored on the system, access it, and transmit it  
10826 to other applications. Information must have protection from unauthorized or accidental access  
10827 or modification.

#### 10828 **D.1.4 Access to the Environment**

10829 Applications must be able to access the external environment to communicate their input and  
10830 results.

#### 10831 **D.1.5 Access to Determinism and Performance Enhancements**

10832 Applications must have sufficient control of resource allocation to ensure the timeliness of  
10833 interactions with external objects.

#### 10834 **D.1.6 Operating System-Dependent Profile**

10835 The capabilities of the operating system may make certain optional characteristics of the base  
10836 language in effect no longer optional, and this should be specified.

**10837 D.1.7 I/O Interaction**

10838 The interaction between the C language I/O subsystem (*stdio*) and the I/O subsystem of  
10839 IEEE Std 1003.1-200x must be specified.

**10840 D.1.8 Internationalization Interaction**

10841 The effects of the environment of IEEE Std 1003.1-200x on the internationalization facilities of the  
10842 C language must be specified.

**10843 D.1.9 C-Language Extensions**

10844 Certain functions in the C language must be extended to support the additional capabilities  
10845 provided by IEEE Std 1003.1-200x.

**10846 D.1.10 Command Language**

10847 Users should be able to define procedures that combine simple tools and/or applications into  
10848 higher-level components that perform to the specific needs of the user. The user should be able  
10849 to store, recall, use, and modify these procedures. These procedures should employ a powerful  
10850 command language that is used for recurring tasks in conforming applications (scripts) in the  
10851 same way that it is used interactively to accomplish one-time tasks. The language and the  
10852 utilities that it uses must be consistent between systems to reduce errors and retraining.

**10853 D.1.11 Interactive Facilities**

10854 Use the system to accomplish individual tasks at an interactive terminal. The interface should be  
10855 consistent, intuitive, and offer usability enhancements to increase the productivity of terminal  
10856 users, reduce errors, and minimize retraining costs. Online documentation or usage assistance  
10857 should be available.

**10858 D.1.12 Accomplish Multiple Tasks Simultaneously**

10859 Access applications and interactive facilities from a single terminal without requiring serial  
10860 execution: switch between multiple interactive tasks; schedule one-time or periodic background  
10861 work; display the status of all work in progress or scheduled; influence the priority scheduling of  
10862 work, when authorized.

**10863 D.1.13 Complex Data Manipulation**

10864 Manipulate data in files in complex ways: sort, merge, compare, translate, edit, format, pattern  
10865 match, select subsets (strings, columns, fields, rows, and so on). These facilities should be  
10866 available to both conforming applications and interactive users.

**10867 D.1.14 File Hierarchy Manipulation**

10868 Create, delete, move/rename, copy, backup/archive, and display files and directories. These  
10869 facilities should be available to both conforming applications and interactive users.

**10870 D.1.15 Locale Configuration**

10871 Customize applications and interactive sessions for the cultural and language conventions of the  
10872 user. Employ a wide variety of standard character encodings. These facilities should be available  
10873 to both conforming applications and interactive users. |

**10874 D.1.16 Inter-User Communication**

10875 Send messages or transfer files to other users on the same system or other systems on a network.  
10876 These facilities should be available to both conforming applications and interactive users. |

**10877 D.1.17 System Environment**

10878 Display information about the status of the system (activities of users and their interactive and  
10879 background work, file system utilization, system time, configuration, and presence of optional  
10880 facilities) and the environment of the user (terminal characteristics, and so on). Inform the  
10881 system operator/administrator of problems. Control access to user files and other resources.

**10882 D.1.18 Printing**

10883 Output files on a variety of output device classes, accessing devices on local or network-  
10884 connected systems. Control (or influence) the formatting, priority scheduling, and output  
10885 distribution of work. These facilities should be available to both conforming applications and |  
10886 interactive users.

**10887 D.1.19 Software Development**

10888 Develop (create and manage source files, compile/interpret, debug) portable open systems  
10889 applications and package them for distribution to, and updating of, other systems.

**10890 D.2 Portability Capabilities**

10891 This section describes the significant portability capabilities of IEEE Std 1003.1-200x and  
10892 indicates how the user requirements listed in Section D.1 (on page 3555) are addressed. The  
10893 capabilities are listed in the same format as the preceding user requirements; they are  
10894 summarized below:

- 10895 • Configuration Interrogation
- 10896 • Process Management
- 10897 • Access to Data
- 10898 • Access to the Environment
- 10899 • Access to Determinism and Performance Enhancements
- 10900 • Operating System-Dependent Profile
- 10901 • I/O Interaction
- 10902 • Internationalization Interaction
- 10903 • C-Language Extensions
- 10904 • Command Language
- 10905 • Interactive Facilities

- 10906 • Accomplish Multiple Tasks Simultaneously
- 10907 • Complex Data Manipulation
- 10908 • File Hierarchy Manipulation
- 10909 • Locale Configuration
- 10910 • Inter-User Communication
- 10911 • System Environment
- 10912 • Printing
- 10913 • Software Development

#### 10914 **D.2.1 Configuration Interrogation**

10915 The *uname()* operation provides basic identification of the system. The *sysconf()*, *pathconf()*, and  
 10916 *fpathconf()* functions and the *getconf* utility provide means to interrogate the implementation to  
 10917 determine how to adapt to the environment in which it is running. These values can be either  
 10918 static (indicating that all instances of the implementation have the same value) or dynamic  
 10919 (indicating that different instances of the implementation have the different values, or that the  
 10920 value may vary for other reasons, such as reconfiguration).

#### 10921 **Unsatisfied Requirements**

10922 None directly. However, as new areas are added, there will be a need for additional capability in  
 10923 this area.

#### 10924 **D.2.2 Process Management**

10925 The *fork()*, *exec* family, and *spawn()* functions provide for the creation of new processes or the  
 10926 insertion of new applications into existing processes. The *\_Exit()*, *\_exit()*, *exit()*, and *abort()*  
 10927 functions allow for the termination of a process by itself. The *wait()* and *waitpid()* functions  
 10928 allow one process to deal with the termination of another.

10929 The *times()* function allows for basic measurement of times used by a process. Various  
 10930 functions, including *fstat()*, *getegid()*, *geteuid()*, *getgid()*, *getrgid()*, *getgrnam()*, *getlogin()*,  
 10931 *getpid()*, *getppid()*, *getpwnam()*, *getpwuid()*, *getuid()*, *lstat()*, and *stat()*, provide for access to the  
 10932 identifiers of processes and the identifiers and names of owners of processes (and files).

10933 The various functions operating on environment variables provide for communication of  
 10934 information (primarily user-configurable defaults) from a parent to child processes.

10935 The operations on the current working directory control and interrogate the directory from  
 10936 which relative filename searches start. The *umask()* function controls the default protections  
 10937 applied to files created by the process.

10938 The *alarm()*, *pause()*, *sleep()*, *ualarm()*, and *usleep()* operations allow the process to suspend until  
 10939 a timer has expired or to be notified when a period of time has elapsed. The *time()* operation  
 10940 interrogates the current time and date.

10941 The signal mechanism provides for communication of events either from other processes or  
 10942 from the environment to the application, and the means for the application to control the effect  
 10943 of these events. The mechanism provides for external termination of a process and for a process  
 10944 to suspend until an event occurs. The mechanism also provides for a value to be associated with  
 10945 an event.

10946 Job control provides a means to group processes and control them as groups, and to control their  
 10947 access to the function between the user and the system (the *controlling terminal*). It also provides  
 10948 the means to suspend and resume processes.

10949 The Process Scheduling option provides control of the scheduling and priority of a process.

10950 The Message Passing option provides a means for interprocess communication involving small  
 10951 amounts of data.

10952 The Memory Management facilities provide control of memory resources and for the sharing of  
 10953 memory. This functionality is mandatory on XSI-conformant systems.

10954 The Threads facilities provide multiple flows of control with a process (threads),  
 10955 synchronization between threads, association of data with threads, and controlled cancelation of  
 10956 threads.

10957 The XSI interprocess communications functionality provide an alternate set of facilities to  
 10958 manipulate semaphores, message queues, and shared memory. These are provided on XSI-  
 10959 conformant systems to support conforming applications developed to run on UNIX systems.

### 10960 D.2.3 Access to Data

10961 The *open()*, *close()*, *fclose()*, *fopen()*, and *pipe()* functions provide for access to files and data.  
 10962 Such files may be regular files, interprocess data channels (pipes), or devices. Additional types  
 10963 of objects in the file system are permitted and are being contemplated for standardization.

10964 The *access()*, *chmod()*, *chown()*, *dup()*, *dup2()*, *fchmod()*, *fcntl()*, *fstat()*, *ftruncate()*, *lstat()*,  
 10965 *readlink()*, *realpath()*, *stat()*, and *utime()* functions allow for control and interrogation of file and  
 10966 file-related objects, (including symbolic links) and their ownership, protections, and timestamps.

10967 The *fgetc()*, *fputc()*, *fread()*, *fseek()*, *fsetpos()*, *fwrite()*, *getc()*, *getch()*, *lseek()*, *putchar()*, *putc()*,  
 10968 *read()*, and *write()* functions provide for data transfer from the application to files (in all their  
 10969 forms).

10970 The *closedir()*, *link()*, *mkdir()*, *opendir()*, *readdir()*, *rename()*, *rmdir()*, *rewinddir()*, and *unlink()*  
 10971 functions provide for a complete set of operations on directories. Directories can arbitrarily  
 10972 contain other directories, and a single file can be mentioned in more than one directory.

10973 The file-locking mechanism provides for advisory locking (protection during transactions) of  
 10974 ranges of bytes (in effect, records) in a file.

10975 The *confstr()*, *fpathconf()*, *pathconf()*, and *sysconf()* functions provide for enquiry as to the  
 10976 behavior of the system where variability is permitted.

10977 The Synchronized Input and Output option provides for assured commitment of data to media.

10978 The Asynchronous Input and Output option provides for initiation and control of asynchronous  
 10979 data transfers.

### 10980 D.2.4 Access to the Environment

10981 The operations and types in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11,  
 10982 General Terminal Interface are provided for access to asynchronous serial devices. The primary  
 10983 intended use for these is the controlling terminal for the application (the interaction point  
 10984 between the user and the system). They are general enough to be used to control any  
 10985 asynchronous serial device. The functions are also general enough to be used with many other  
 10986 device types as a user interface when some emulation is provided.

10987 Less detailed access is provided for other device types, but in many instances an application  
10988 need not know whether an object in the file system is a device or a regular file to operate  
10989 correctly.

#### 10990 **Unsatisfied Requirements**

10991 Detailed control of common device classes, specifically magnetic tape, is not provided.

### 10992 **D.2.5 Bounded (Realtime) Response**

10993 The Realtime Signals Extension provides queued signals and the prioritization of the handling of  
10994 signals. The SCHED\_FIFO, SCHED\_SPORADIC, and SCHED\_RR scheduling policies provide  
10995 control over processor allocation. The Semaphores option provides high-performance  
10996 synchronization. The Memory Management functions provide memory locking for control of  
10997 memory allocation, file mapping for high-performance, and shared memory for high-  
10998 performance interprocess communication. The Message Passing option provides for interprocess  
10999 communication without being dependent on shared memory.

11000 The Timers option provides a high resolution function called *nanosleep()* with a finer resolution  
11001 than the *sleep()* function.

11002 The Typed Memory Objects option, the Monotonic Clock option, and the Timeouts option  
11003 provide further facilities for applications to use to obtain predictable bounded response.

### 11004 **D.2.6 Operating System-Dependent Profile**

11005 IEEE Std 1003.1-200x makes no distinction between text and binary files. The values of  
11006 EXIT\_SUCCESS and EXIT\_FAILURE are further defined.

#### 11007 **Unsatisfied Requirements**

11008 None known, but the ISO C standard may contain some additional options that could be  
11009 specified.

### 11010 **D.2.7 I/O Interaction**

11011 IEEE Std 1003.1-200x defines how each of the ISO C standard *stdio* functions interact with the  
11012 POSIX.1 operations, typically specifying the behavior in terms of POSIX.1 operations.

#### 11013 **Unsatisfied Requirements**

11014 None.

### 11015 **D.2.8 Internationalization Interaction**

11016 The IEEE Std 1003.1-200x environment operations provide a means to define the environment  
11017 for *setlocale()* and time functions such as *ctime()*. The *tzset()* function is provided to set time  
11018 conversion information.

11019 The *nl\_langinfo()* function is provided as an XSI extension to query locale-specific cultural  
11020 settings.

11021 **Unsatisfied Requirements**

11022 None.

11023 **D.2.9 C-Language Extensions**11024 The *setjmp()* and *longjmp()* functions are not defined to be cognizant of the signal masks defined  
11025 for POSIX.1. The *sigsetjmp()* and *siglongjmp()* functions are provided to fill this gap.11026 The *\_setjmp()* and *\_longjmp()* functions are provided as XSI extensions to support historic  
11027 practice.11028 **Unsatisfied Requirements**

11029 None.

11030 **D.2.10 Command Language**11031 The shell command language, as described in Shell and Utilities volume of IEEE Std 1003.1-200x,  
11032 Chapter 2, Shell Command Language, is a common language useful in batch scripts, through an  
11033 API to high-level languages (for the C-Language Binding option, *system()* and *popen()*) and  
11034 through an interactive terminal (see the *sh* utility). The shell language has many of the  
11035 characteristics of a high-level language, but it has been designed to be more suitable for user  
11036 terminal entry and includes interactive debugging facilities. Through the use of pipelining,  
11037 many complex commands can be constructed from combinations of data filters and other  
11038 common components. Shell scripts can be created, stored, recalled, and modified by the user  
11039 with simple editors.11040 In addition to the basic shell language, the following utilities offer features that simplify and  
11041 enhance programmatic access to the utilities and provide features normally found only in high-  
11042 level languages: *basename*, *bc*, *command*, *dirname*, *echo*, *env*, *expr*, *false*, *printf*, *read*, *sleep*, *tee*, *test*,  
11043 *time\**,<sup>2</sup> *true*, *wait*, *xargs*, and all of the special built-in utilities in the Shell and Utilities volume of  
11044 IEEE Std 1003.1-200x, Section 2.14, Special Built-In Utilities .11045 **Unsatisfied Requirements**

11046 None.

11047 **D.2.11 Interactive Facilities**11048 The utilities offer a common style of command-line interface through conformance to the Utility  
11049 Syntax Guidelines (see the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2, Utility  
11050 Syntax Guidelines) and the common utility defaults (see the Shell and Utilities volume of  
11051 IEEE Std 1003.1-200x, Section 1.11, Utility Description Defaults). The *sh* utility offers an  
11052 interactive command-line history and editing facility. The following utilities in the User  
11053 Portability Utilities option have been customized for interactive use: *alias*, *ex*, *fc*, *mailx*, *more*, *talk*,  
11054 *vi*, *unalias*, and *write*; the *man* utility offers online access to system documentation.

11055 \_\_\_\_\_

11056 2. The utilities listed with an asterisk here and later in this section are present only on systems which support the User Portability  
11057 Utilities option. There may be further restrictions on the utilities offered with various configuration option combinations; see the  
11058 individual utility descriptions. |

11059 **Unsatisfied Requirements**

11060 The command line interface to individual utilities is as intuitive and consistent as historical  
 11061 practice allows. Work underway based on graphical user interfaces may be more suitable for  
 11062 novice or occasional users of the system.

11063 **D.2.12 Accomplish Multiple Tasks Simultaneously**

11064 The shell command language offers background processing through the asynchronous list  
 11065 command form; see the Shell and Utilities volume of IEEE Std 1003.1-200x, Section 2.9, Shell  
 11066 Commands. The *nohup* utility makes background processing more robust and usable. The *kill*  
 11067 utility can terminate background jobs. When the User Portability Utilities option is supported,  
 11068 the following utilities allow manipulation of jobs: *bg*, *fg*, and *jobs*. Also, if the User Portability  
 11069 Utilities option is supported, the following can support periodic job scheduling, control, and  
 11070 display: *at*, *batch*, *crontab*, *nice*, *ps*, and *renice*.

11071 **Unsatisfied Requirements**

11072 Terminals with multiple windows may be more suitable for some multi-tasking interactive uses  
 11073 than the job control approach in IEEE Std 1003.1-200x. See the comments on graphical user  
 11074 interfaces in Section D.2.11 (on page 3562). The *nice* and *renice* utilities do not necessarily take  
 11075 advantage of complex system scheduling algorithms that are supported by the realtime options  
 11076 within IEEE Std 1003.1-200x.

11077 **D.2.13 Complex Data Manipulation**

11078 The following utilities address user requirements in this area: *asa*, *awk*, *bc*, *cmp*, *comm*, *csplit\**, *cut*,  
 11079 *dd*, *diff*, *ed*, *ex\**, *expand\**, *expr*, *find*, *fold*, *grep*, *head*, *join*, *od*, *paste*, *pr*, *printf*, *sed*, *sort*, *split\**, *tabs\**, *tail*,  
 11080 *tr*, *unexpand\**, *uniq*, *uudecode\**, *uuencode\**, and *wc*.

11081 **Unsatisfied Requirements**

11082 Sophisticated text formatting utilities, such as *troff* or *TeX*, are not included. Standards work in  
 11083 the area of SGML may satisfy this.

11084 **D.2.14 File Hierarchy Manipulation**

11085 The following utilities address user requirements in this area: *basename*, *cd*, *chgrp*, *chmod*, *chown*,  
 11086 *cksum*, *cp*, *dd*, *df\**, *diff*, *dirname*, *du\**, *find*, *ls*, *ln*, *mkdir*, *mkfifo*, *mv*, *patch\**, *pathchk*, *pax*, *pwd*, *rm*, *rmdir*,  
 11087 *test*, and *touch*.

11088 **Unsatisfied Requirements**

11089 Some graphical user interfaces offer more intuitive file manager components that allow file  
 11090 manipulation through the use of icons for novice users.

**11091 D.2.15 Locale Configuration**

11092 The standard utilities are affected by the various *LC\_* variables to achieve locale-dependent  
11093 operation: character classification, collation sequences, regular expressions and shell pattern  
11094 matching, date and time formats, numeric formatting, and monetary formatting. When the  
11095 POSIX2\_LOCALEDEF option is supported, applications can provide their own locale definition  
11096 files. The following utilities address user requirements in this area: *date*, *ed*, *ex\**, *find*, *grep*, *locale*,  
11097 *localedef*, *more\**, *sed*, *sh*, *sort*, *tr*, *uniq*, and *vi\**.

11098 The *iconv()*, *iconv\_close()*, and *iconv\_open()* functions are available to allow an application to  
11099 convert character data between supported character sets.

11100 The *genccat* utility and the *catopen()*, *catclose()*, and *catgets()* functions for message catalog  
11101 manipulation are available on XSI-conformant systems.

**11102 Unsatisfied Requirements**

11103 Some aspects of multi-byte character and state-encoded character encodings have not yet been  
11104 addressed. The C-language functions, such as *getopt()*, are generally limited to single-byte  
11105 characters. The effect of the *LC\_MESSAGES* variable on message formats is only suggested at  
11106 this time.

**11107 D.2.16 Inter-User Communication**

11108 The following utilities address user requirements in this area: *cksum*, *mailx\**, *mesg\**, *patch\**, *pax*,  
11109 *talk\**, *uudecode\**, *uuencode\**, *who\**, and *write\**.

11110 The historical UUCP utilities are included on XSI-conformant systems.

**11111 Unsatisfied Requirements**

11112 None.

**11113 D.2.17 System Environment**

11114 The following utilities address user requirements in this area: *chgrp*, *chmod*, *chown*, *df\**, *du\**, *env*,  
11115 *getconf*, *id*, *logger*, *logname*, *mesg\**, *newgrp\**, *ps\**, *stty*, *tput\**, *tty*, *umask*, *uname*, and *who\**.

11116 The *closelog()*, *openlog()*, *setlogmask()*, and *syslog()* functions provide System Logging facilities  
11117 on XSI-conformant systems; these are analogous to the *logger* utility.

**11118 Unsatisfied Requirements**

11119 None.

**11120 D.2.18 Printing**

11121 The following utilities address user requirements in this area: *pr* and *lp*.

11122 **Unsatisfied Requirements**

11123 There are no features to control the formatting or scheduling of the print jobs.

11124 **D.2.19 Software Development**

11125 The following utilities address user requirements in this area: *ar*, *asa*, *awk*, *c99*, *ctags\**, *fort77*,  
11126 *getconf*, *getopts*, *lex*, *localedef*, *make*, *nm\**, *od*, *patch\**, *pax*, *strings\**, *strip*, *time\**, and *yacc*.

11127 The *system()*, *popen()*, *pclose()*, *regcomp()*, *regexec()*, *regerror()*, *regfree()*, *fnmatch()*, *getopt()*,  
11128 *glob()*, *globfree()*, *wordexp()*, and *wordfree()* functions allow C-language programmers to access  
11129 some of the interfaces used by the utilities, such as argument processing, regular expressions,  
11130 and pattern matching.

11131 The SCCS source-code control system utilities are available on systems supporting the XSI  
11132 Development option.

11133 **Unsatisfied Requirements**

11134 There are no language-specific development tools related to languages other than C and  
11135 FORTRAN. The C tools are more complete and varied than the FORTRAN tools. There is no  
11136 data dictionary or other CASE-like development tools.

11137 **D.2.20 Future Growth**

11138 It is arguable whether or not all functionality to support applications is potentially within the  
11139 scope of IEEE Std 1003.1-200x. As a simple matter of practicality, it cannot be. Areas such as |  
11140 graphics, application domain-specific functionality, windowing, and so on, should be in unique |  
11141 standards. As such, they are properly “Unsatisfied Requirements” in terms of providing fully |  
11142 conforming applications, but ones which are outside the scope of IEEE Std 1003.1-200x. |

11143 However, as the standards evolve, certain functionality once considered “exotic” enough to be |  
11144 part of a separate standard become common enough to be included in a core standard such as |  
11145 this. Realtime and networking, for example, have both moved from separate standards (with |  
11146 much difficult cross-referencing) into IEEE Std 1003.1 over time, and although no specific areas |  
11147 have been identified for inclusion in future revisions, such inclusions seem likely. |

11148 **D.3 Profiling Considerations**

11149 This section offers guidance to writers of profiles on how the configurable options, limits, and  
11150 optional behavior of IEEE Std 1003.1-200x should be cited in profiles. Profile writers should  
11151 consult the general guidance in POSIX.0 when writing POSIX Standardized Profiles.

11152 The information in this section is an inclusive list of features that should be considered by profile |  
11153 writers. Subsetting of IEEE Std 1003.1-200x should follow the Base Definitions volume of |  
11154 IEEE Std 1003.1-200x, Section 2.1.5.1, Subprofiling Considerations. A set of profiling options is |  
11155 described in Appendix E (on page 3579). |

11156 **D.3.1 Configuration Options**

11157 There are two set of options suggested by IEEE Std 1003.1-200x: those for POSIX-conforming  
 11158 systems and those for X/Open System Interface (XSI) conformance. The requirements for XSI  
 11159 conformance are documented in the Base Definitions volume of IEEE Std 1003.1-200x and not  
 11160 discussed further here, as they superset the POSIX conformance requirements.

11161 **D.3.2 Configuration Options (Shell and Utilities)**

11162 There are three broad optional configurations for the Shell and Utilities volume of  
 11163 IEEE Std 1003.1-200x: basic execution system, development system, and user portability  
 11164 interactive system. The options to support these, and other minor configuration options, are  
 11165 listed in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 2, Conformance. Profile  
 11166 writers should consult the following list and the comments concerning user requirements  
 11167 addressed by various components in Section D.2 (on page 3558).

## 11168 POSIX2\_UPE

11169 The system supports the User Portability Utilities option.

11170 This option is a requirement for a user portability interactive system. It is required  
 11171 frequently except for those systems, such as embedded realtime or dedicated application  
 11172 systems, that support little or no interactive time-sharing work by users or operators. XSI-  
 11173 conformant systems support this option.

## 11174 POSIX2\_SW\_DEV

11175 The system supports the Software Development Utilities option.

11176 This option is required by many systems, even those in which actual software development  
 11177 does not occur. The *make* utility, in particular, is required by many application software  
 11178 packages as they are installed onto the system. If POSIX2\_C\_DEV is supported,  
 11179 POSIX2\_SW\_DEV is almost a mandatory requirement because of *ar* and *make*.

## 11180 POSIX2\_C\_BIND

11181 The system supports the C-Language Bindings option.

11182 This option is required on some implementations developing complex C applications or on  
 11183 any system installing C applications in source form that require the functions in this option.  
 11184 The *system()* and *popen()* functions, in particular, are widely used by applications; the  
 11185 others are rather more specialized.

## 11186 POSIX2\_C\_DEV

11187 The system supports the C-Language Development Utilities option.

11188 This option is required by many systems, even those in which actual C-language software  
 11189 development does not occur. The *c99* utility, in particular, is required by many application  
 11190 software packages as they are installed onto the system. The *lex* and *yacc* utilities are used  
 11191 less frequently.

## 11192 POSIX2\_FORT\_DEV

11193 The system supports the FORTRAN Development Utilities option

11194 As with C, this option is needed on any system developing or installing FORTRAN  
 11195 applications in source form.

## 11196 POSIX2\_FORT\_RUN

11197 The system supports the FORTRAN Runtime Utilities option.

11198 This option is required for some FORTRAN applications that need the *asa* utility to convert  
 11199 Hollerith printing statement output. It is unknown how frequently this occurs.

- 11200 POSIX2\_LOCALEDEF  
 11201 The system supports the creation of locales.
- 11202 This option is needed if applications require their own customized locale definitions to  
 11203 operate. It is presently unknown whether many applications are dependent on this.  
 11204 However, the option is virtually mandatory for systems in which internationalized  
 11205 applications are developed.
- 11206 XSI-conformant systems support this option.
- 11207 POSIX2\_PBS  
 11208 The system supports the Batch Environment option.
- 11209 POSIX2\_PBS\_ACCOUNTING  
 11210 The system supports the optional feature of accounting within the Batch Environment  
 11211 option. It will be required in servers that implement the optional feature of accounting.
- 11212 POSIX2\_PBS\_CHECKPOINT  
 11213 The systems supports the optional feature of checkpoint/restart within the Batch  
 11214 Environment option.
- 11215 POSIX2\_PBS\_LOCATE  
 11216 The system supports the optional feature of locating batch jobs within the Batch  
 11217 Environment option.
- 11218 POSIX2\_PBS\_MESSAGE  
 11219 The system supports the optional feature of sending messages to batch jobs within the  
 11220 Batch Environment option.
- 11221 POSIX2\_PBS\_TRACK  
 11222 The system supports the optional feature of tracking batch jobs within the Batch  
 11223 Environment option.
- 11224 POSIX2\_CHAR\_TERM  
 11225 The system supports at least one terminal type capable of all operations described in  
 11226 IEEE Std 1003.1-200x.
- 11227 On systems with POSIX2\_UPE, this option is almost always required. It was developed  
 11228 solely to allow certain specialized vendors and user applications to bypass the requirement  
 11229 for general-purpose asynchronous terminal support. For example, an application and  
 11230 system that was suitable for block-mode terminals, such as IBM 3270s, would not need this  
 11231 option.
- 11232 XSI-conformant systems support this option.

### 11233 D.3.3 Configurable Limits

- 11234 Very few of the limits need to be increased for profiles. No profile can cite lower values.
- 11235 {POSIX2\_BC\_BASE\_MAX}  
 11236 {POSIX2\_BC\_DIM\_MAX}  
 11237 {POSIX2\_BC\_SCALE\_MAX}  
 11238 {POSIX2\_BC\_STRING\_MAX}
- 11239 No increase is anticipated for any of these *bc* values, except for very specialized applications  
 11240 involving huge numbers.
- 11241 {POSIX2\_COLL\_WEIGHTS\_MAX}
- 11242 Some natural languages with complex collation requirements require an increase from the  
 11243 default 2 to 4; no higher numbers are anticipated.

- 11244 {POSIX2\_EXPR\_NEST\_MAX}  
 11245 No increase is anticipated.
- 11246 {POSIX2\_LINE\_MAX}  
 11247 This number is much larger than most historical applications have been able to use. At some  
 11248 future time, applications may be rewritten to take advantage of even larger values.
- 11249 {POSIX2\_RE\_DUP\_MAX}  
 11250 No increase is anticipated.
- 11251 {POSIX2\_VERSION}  
 11252 This is actually not a limit, but a standard version stamp. Generally, a profile should specify  
 11253 Shell and Utilities volume of IEEE Std 1003.1-200x, Chapter 2, Shell Command Language by  
 11254 name in the normative references section, not this value.

#### 11255 D.3.4 Configuration Options (System Interfaces)

- 11256 {NGROUPS\_MAX}  
 11257 A non-zero value indicates that the implementation supports supplementary groups.
- 11258 This option is needed where there is a large amount of shared use of files, but where a  
 11259 certain amount of protection is needed. Many profiles<sup>3</sup> are known to require this option; it  
 11260 should only be required if needed, but it should never be prohibited.
- 11261 \_POSIX\_ADVISORY\_INFO  
 11262 The system provides advisory information for file management.
- 11263 This option allows the application to specify advisory information that can be used to  
 11264 achieve better or even deterministic response time in file manager or input and output  
 11265 operations.
- 11266 \_POSIX\_ASYNCHRONOUS\_IO  
 11267 The system provides concurrent process execution and input and output transfers.
- 11268 This option was created to support historical systems that did not provide the feature. It  
 11269 should only be required if needed, but it should never be prohibited.
- 11270 \_POSIX\_BARRIERS  
 11271 The system supports barrier synchronization.
- 11272 This option was created to allow efficient synchronization of multiple parallel threads in  
 11273 multi-processor systems in which the operation is supported in part by the hardware  
 11274 architecture.
- 11275 \_POSIX\_CHOWN\_RESTRICTED  
 11276 The system restricts the right to “give away” files to other users.
- 11277 This option should be carefully investigated before it is required. Some applications expect  
 11278 that they can change the ownership of files in this way. It is provided where either security  
 11279 or system account requirements cause this ability to be a problem. It is also known to be  
 11280 specified in many profiles.

11281 \_\_\_\_\_ |  
 11282 3. There are no formally approved profiles of IEEE Std 1003.1-200x at the time of publication; the reference here is to various |  
 11283 profiles generated by private bodies or governments. |

- 11284 `_POSIX_CLOCK_SELECTION`  
11285 The system supports the Clock Selection option.
- 11286 This option allows applications to request a high resolution sleep in order to suspend a  
11287 thread during a relative time interval, or until an absolute time value, using the desired  
11288 clock. It also allows the application to select the clock used in a `pthread_cond_timedwait()`  
11289 function call.
- 11290 `_POSIX_CPUTIME`  
11291 The system supports the Process CPU-Time Clocks option.
- 11292 This option allows applications to use a new clock that measures the execution times of  
11293 processes or threads, and the possibility to create timers based upon these clocks, for  
11294 runtime detection (and treatment) of execution time overruns.
- 11295 `_POSIX_FSYNC`  
11296 The system supports file synchronization requests.
- 11297 This option was created to support historical systems that did not provide the feature.  
11298 Applications that are expecting guaranteed completion of their input and output operations  
11299 should require the `_POSIX_SYNC_IO` option. This option should never be prohibited.
- 11300 XSI-conformant systems support this option.
- 11301 `_POSIX_IPV6`  
11302 The system supports facilities related to Internet Protocol Version 6 (IPv6).
- 11303 This option was created to allow systems to transition to IPv6.
- 11304 `_POSIX_JOB_CONTROL`  
11305 Job control facilities are mandatory in IEEE Std 1003.1-200x.
- 11306 The option was created primarily to support historical systems that did not provide the  
11307 feature. Many existing profiles now require it; it should only be required if needed, but it  
11308 should never be prohibited. Most applications that use it can run when it is not present,  
11309 although with a degraded level of user convenience.
- 11310 `_POSIX_MAPPED_FILES`  
11311 The system supports the mapping of regular files into the process address space.
- 11312 XSI-conformant systems support this option.
- 11313 Both this option and the Shared Memory Objects option provide shared access to memory  
11314 objects in the process address space. The functions defined under this option provide the  
11315 functionality of existing practice for mapping regular files. This functionality was deemed  
11316 unnecessary, if not inappropriate, for embedded systems applications and, hence, is  
11317 provided under this option. It should only be required if needed, but it should never be  
11318 prohibited.
- 11319 `_POSIX_MEMLOCK`  
11320 The system supports the locking of the address space.
- 11321 This option was created to support historical systems that did not provide the feature. It  
11322 should only be required if needed, but it should never be prohibited.
- 11323 `_POSIX_MEMLOCK_RANGE`  
11324 The system supports the locking of specific ranges of the address space.
- 11325 For applications that have well-defined sections that need to be locked and others that do  
11326 not, IEEE Std 1003.1-200x supports an optional set of functions to lock or unlock a range of  
11327 process addresses. The following are two reasons for having a means to lock down a

- 11328           specific range:
- 11329           1. An asynchronous event handler function that must respond to external events in a  
11330           deterministic manner such that page faults cannot be tolerated
- 11331           2. An input/output “buffer” area that is the target for direct-to-process I/O, and the  
11332           overhead of implicit locking and unlocking for each I/O call cannot be tolerated
- 11333           It should only be required if needed, but it should never be prohibited.
- 11334           \_POSIX\_MEMORY\_PROTECTION  
11335           The system supports memory protection.
- 11336           XSI-conformant systems support this option.
- 11337           The provision of this option typically imposes additional hardware requirements. It should  
11338           never be prohibited.
- 11339           \_POSIX\_PRIORITIZED\_IO  
11340           The system provides prioritization for input and output operations.
- 11341           The use of this option may interfere with the ability of the system to optimize input and  
11342           output throughput. It should only be required if needed, but it should never be prohibited.
- 11343           \_POSIX\_MESSAGE\_PASSING  
11344           The system supports the passing of messages between processes.
- 11345           This option was created to support historical systems that did not provide the feature. The  
11346           functionality adds a high-performance XSI interprocess communication facility for local  
11347           communication. It should only be required if needed, but it should never be prohibited.
- 11348           \_POSIX\_MONOTONIC\_CLOCK  
11349           The system supports the Monotonic Clock option.
- 11350           This option allows realtime applications to rely on a monotonically increasing clock that  
11351           does not jump backwards, and whose value does not change except for the regular ticking  
11352           of the clock.
- 11353           \_POSIX\_PRIORITY\_SCHEDULING  
11354           The system provides priority-based process scheduling.
- 11355           Support of this option provides predictable scheduling behavior, allowing applications to  
11356           determine the order in which processes that are ready to run are granted access to a  
11357           processor. It should only be required if needed, but it should never be prohibited.
- 11358           \_POSIX\_REALTIME\_SIGNALS  
11359           The system provides prioritized, queued signals with associated data values.
- 11360           This option was created to support historical systems that did not provide the features. It  
11361           should only be required if needed, but it should never be prohibited.
- 11362           \_POSIX\_REGEX  
11363           Support for regular expression facilities are mandatory in IEEE Std 1003.1-200x.
- 11364           \_POSIX\_SAVED\_IDS  
11365           Support for this feature is mandatory in IEEE Std 1003.1-200x.
- 11366           Certain classes of applications rely on it for proper operation, and there is no alternative  
11367           short of giving the application root privileges on most implementations that did not provide  
11368           \_POSIX\_SAVED\_IDS.

- 11369     \_POSIX\_SEMAPHORES  
11370         The system provides counting semaphores.
- 11371         This option was created to support historical systems that did not provide the feature. It  
11372         should only be required if needed, but it should never be prohibited.
- 11373     \_POSIX\_SHARED\_MEMORY\_OBJECTS  
11374         The system supports the mapping of shared memory objects into the process address space.
- 11375         Both this option and the Memory Mapped Files option provide shared access to memory  
11376         objects in the process address space. The functions defined under this option provide the  
11377         functionality of existing practice for shared memory objects. This functionality was deemed  
11378         appropriate for embedded systems applications and, hence, is provided under this option. It  
11379         should only be required if needed, but it should never be prohibited.
- 11380     \_POSIX\_SHELL  
11381         Support for the *sh* utility command line interpreter is mandatory in IEEE Std 1003.1-200x.
- 11382     \_POSIX\_SPAWN  
11383         The system supports the spawn option.
- 11384         This option provides applications with an efficient mechanism to spawn execution of a new  
11385         process.
- 11386     \_POSIX\_SPINLOCKS  
11387         The system supports spin locks.
- 11388         This option was created to support a simple and efficient synchronization mechanism for  
11389         threads executing in multi-processor systems.
- 11390     \_POSIX\_SPORADIC\_SERVER  
11391         The system supports the sporadic server scheduling policy.
- 11392         This option provides applications with a new scheduling policy for scheduling aperiodic  
11393         processes or threads in hard realtime applications.
- 11394     \_POSIX\_SYNCHRONIZED\_IO  
11395         The system supports guaranteed file synchronization.
- 11396         This option was created to support historical systems that did not provide the feature.  
11397         Applications that are expecting guaranteed completion of their input and output operations  
11398         should require this option, rather than the File Synchronization option. It should only be  
11399         required if needed, but it should never be prohibited.
- 11400     \_POSIX\_THREADS  
11401         The system supports multiple threads of control within a single process.
- 11402         This option was created to support historical systems that did not provide the feature.  
11403         Applications written assuming a multi-threaded environment would be expected to require  
11404         this option. It should only be required if needed, but it should never be prohibited.
- 11405         XSI-conformant systems support this option.
- 11406     \_POSIX\_THREAD\_ATTR\_STACKADDR  
11407         The system supports specification of the stack address for a created thread.
- 11408         Applications may take advantage of support of this option for performance benefits, but  
11409         dependence on this feature should be minimized. This option should never be prohibited.
- 11410         XSI-conformant systems support this option.

- 11411 \_POSIX\_THREAD\_ATTR\_STACKSIZE  
11412 The system supports specification of the stack size for a created thread.
- 11413 Applications may require this option in order to ensure proper execution, but such usage  
11414 limits portability and dependence on this feature should be minimized. It should only be  
11415 required if needed, but it should never be prohibited.
- 11416 XSI-conformant systems support this option.
- 11417 \_POSIX\_THREAD\_PRIORITY\_SCHEDULING  
11418 The system provides priority-based thread scheduling.
- 11419 Support of this option provides predictable scheduling behavior, allowing applications to  
11420 determine the order in which threads that are ready to run are granted access to a processor.  
11421 It should only be required if needed, but it should never be prohibited.
- 11422 \_POSIX\_THREAD\_PRIO\_INHERIT  
11423 The system provides mutual exclusion operations with priority inheritance.
- 11424 Support of this option provides predictable scheduling behavior, allowing applications to  
11425 determine the order in which threads that are ready to run are granted access to a processor.  
11426 It should only be required if needed, but it should never be prohibited.
- 11427 \_POSIX\_THREAD\_PRIO\_PROTECT  
11428 The system supports a priority ceiling emulation protocol for mutual exclusion operations.
- 11429 Support of this option provides predictable scheduling behavior, allowing applications to  
11430 determine the order in which threads that are ready to run are granted access to a processor.  
11431 It should only be required if needed, but it should never be prohibited.
- 11432 \_POSIX\_THREAD\_PROCESS\_SHARED  
11433 The system provides shared access among multiple processes to synchronization objects.
- 11434 This option was created to support historical systems that did not provide the feature. It  
11435 should only be required if needed, but it should never be prohibited.
- 11436 XSI-conformant systems support this option.
- 11437 \_POSIX\_THREAD\_SAFE\_FUNCTIONS  
11438 The system provides thread-safe versions of all of the POSIX.1 functions.
- 11439 This option is required if the Threads option is supported. This is a separate option because  
11440 thread-safe functions are useful in implementations providing other mechanisms for  
11441 concurrency. It should only be required if needed, but it should never be prohibited.
- 11442 XSI-conformant systems support this option.
- 11443 \_POSIX\_THREAD\_SPORADIC\_SERVER  
11444 The system supports the thread sporadic server scheduling policy.
- 11445 Support for this option provides applications with a new scheduling policy for scheduling  
11446 aperiodic threads in hard realtime applications.
- 11447 \_POSIX\_TIMEOUTS  
11448 The system provides timeouts for some blocking services.
- 11449 This option was created to provide a timeout capability to system services, thus allowing  
11450 applications to include better error detection, and recovery capabilities.
- 11451 \_POSIX\_TIMERS  
11452 The system provides higher resolution clocks with multiple timers per process.

11453 This option was created to support historical systems that did not provide the features. This  
 11454 option is appropriate for applications requiring higher resolution timestamps or needing to  
 11455 control the timing of multiple activities. It should only be required if needed, but it should  
 11456 never be prohibited.

11457 `_POSIX_TRACE`

11458 The system supports the trace option.

11459 This option was created to allow applications to perform tracing.

11460 `_POSIX_TRACE_EVENT_FILTER`

11461 The system supports the trace event filter option.

11462 This option is dependent on support of the Trace option.

11463 `_POSIX_TRACE_INHERIT`

11464 The system supports the trace inherit option.

11465 This option is dependent on support of the Trace option.

11466 `_POSIX_TRACE_LOG`

11467 The system supports the trace log option.

11468 This option is dependent on support of the Trace option.

11469 `_POSIX_TYPED_MEMORY_OBJECTS`

11470 The system supports typed memory objects.

11471 This option was created to allow realtime applications to access different kinds of physical  
 11472 memory, and allow processes in these applications to share portions of this memory.

### 11473 D.3.5 Configurable Limits

11474 In general, the configurable limits in the `<limits.h>` header defined in the Base Definitions  
 11475 volume of IEEE Std 1003.1-200x have been set to minimal values; many applications or  
 11476 implementations may require larger values. No profile can cite lower values.

11477 `{AIO_LISTIO_MAX}`

11478 The current minimum is likely to be inadequate for most applications. It is expected that  
 11479 this value will be increased by profiles requiring support for list input and output  
 11480 operations.

11481 `{AIO_MAX}`

11482 The current minimum is likely to be inadequate for most applications. It is expected that  
 11483 this value will be increased by profiles requiring support for asynchronous input and  
 11484 output operations.

11485 `{AIO_PRIO_DELTA_MAX}`

11486 The functionality associated with this limit is needed only by sophisticated applications. It  
 11487 is not expected that this limit would need to be increased under a general-purpose profile.

11488 `{ARG_MAX}`

11489 The current minimum is likely to need to be increased for profiles, particularly as larger  
 11490 amounts of information are passed through the environment. Many implementations are  
 11491 believed to support larger values.

11492 `{CHILD_MAX}`

11493 The current minimum is suitable only for systems where a single user is not running  
 11494 applications in parallel. It is significantly too low for any system also requiring windows,  
 11495 and if `_POSIX_JOB_CONTROL` is specified, it should be raised.

- 11496 {CLOCKRES\_MIN}  
11497 It is expected that profiles will require a finer granularity clock, perhaps as fine as 1  $\mu$ s,  
11498 represented by a value of 1 000 for this limit.
- 11499 {DELAYTIMER\_MAX}  
11500 It is believed that most implementations will provide larger values.
- 11501 {LINK\_MAX}  
11502 For most applications and usage, the current minimum is adequate. Many implementations  
11503 have a much larger value, but this should not be used as a basis for raising the value unless  
11504 the applications to be used require it.
- 11505 {LOGIN\_NAME\_MAX}  
11506 This is not actually a limit, but an implementation parameter. No profile should impose a  
11507 requirement on this value.
- 11508 {MAX\_CANON}  
11509 For most purposes, the current minimum is adequate. Unless high-speed burst serial  
11510 devices are used, it should be left as is.
- 11511 {MAX\_INPUT}  
11512 See {MAX\_CANON}.
- 11513 {MQ\_OPEN\_MAX}  
11514 The current minimum should be adequate for most profiles.
- 11515 {MQ\_PRIO\_MAX}  
11516 The current minimum corresponds to the required number of process scheduling priorities.  
11517 Many realtime practitioners believe that the number of message priority levels ought to be  
11518 the same as the number of execution scheduling priorities.
- 11519 {NAME\_MAX}  
11520 Many implementations now support larger values, and many applications and users  
11521 assume that larger names can be used. Many existing profiles also specify a larger value.  
11522 Specifying this value will reduce the number of conforming implementations, although this  
11523 might not be a significant consideration over time. Values greater than 255 should not be  
11524 required.
- 11525 {NGROUPS\_MAX}  
11526 The value selected will typically be 8 or larger.
- 11527 {OPEN\_MAX}  
11528 The historically common value for this has been 20. Many implementations support larger  
11529 values. If applications that use larger values are anticipated, an appropriate value should be  
11530 specified.
- 11531 {PAGESIZE}  
11532 This is not actually a limit, but an implementation parameter. No profile should impose a  
11533 requirement on this value.
- 11534 {PATH\_MAX}  
11535 Historically, the minimum has been either 1024 or indefinite, depending on the  
11536 implementation. Few applications actually require values larger than 256, but some users  
11537 may create file hierarchies that must be accessed with longer paths. This value should only  
11538 be changed if there is a clear requirement.
- 11539 {PIPE\_BUF}  
11540 The current minimum is adequate for most applications. Historically, it has been larger. If  
11541 applications that write single transactions larger than this are anticipated, it should be

11542           increased. Applications that write lines of text larger than this probably do not need it  
11543           increased, as the text line is delimited by a newline.

11544           {POSIX\_VERSION}  
11545           This is actually not a limit, but a standard version stamp. Generally, a profile should specify  
11546           IEEE Std 1003.1-200x by a name in the normative references section, not this value.

11547           {PTHREAD\_DESTRUCTOR\_ITERATIONS}  
11548           It is unlikely that applications will need larger values to avoid loss of memory resources.

11549           {PTHREAD\_KEYS\_MAX}  
11550           The current value should be adequate for most profiles.

11551           {PTHREAD\_STACK\_MIN}  
11552           This should not be treated as an actual limit, but as an implementation parameter. No  
11553           profile should impose a requirement on this value.

11554           {PTHREAD\_THREADS\_MAX}  
11555           It is believed that most implementations will provide larger values.

11556           {RTSIG\_MAX}  
11557           The current limit was chosen so that the set of POSIX.1 signal numbers can fit within a 32-  
11558           bit field. It is recognized that most existing implementations define many more signals than  
11559           are specified in POSIX.1 and, in fact, many implementations have already exceeded 32  
11560           signals (including the “null signal”). Support of {\_POSIX\_RTSIG\_MAX} additional signals  
11561           may push some implementations over the single 32-bit word line, but is unlikely to push  
11562           any implementations that are already over that line beyond the 64 signal line.

11563           {SEM\_NSEMS\_MAX}  
11564           The current value should be adequate for most profiles.

11565           {SEM\_VALUE\_MAX}  
11566           The current value should be adequate for most profiles.

11567           {SSIZE\_MAX}  
11568           This limit reflects fundamental hardware characteristics (the size of an integer), and should  
11569           not be specified unless it is clearly required. Extreme care should be taken to assure that  
11570           any value that might be specified does not unnecessarily eliminate implementations  
11571           because of accidents of hardware design.

11572           {STREAM\_MAX}  
11573           This limit is very closely related to {OPEN\_MAX}. It should never be larger than  
11574           {OPEN\_MAX}, but could reasonably be smaller for application areas where most files are  
11575           not accessed through *stdio*. Some implementations may limit {STREAM\_MAX} to 20 but  
11576           allow {OPEN\_MAX} to be considerably larger. Such implementations should be allowed for  
11577           if the applications permit.

11578           {TIMER\_MAX}  
11579           The current limit should be adequate for most profiles, but it may need to be larger for  
11580           applications with a large number of asynchronous operations.

11581           {TTY\_NAME\_MAX}  
11582           This is not actually a limit, but an implementation parameter. No profile should impose a  
11583           requirement on this value.

11584           {TZNAME\_MAX}  
11585           The minimum has been historically adequate, but if longer timezone names are anticipated  
11586           (particularly such values as UTC-1), this should be increased.

11587 **D.3.6 Optional Behavior**

11588 In IEEE Std 1003.1-200x, there are no instances of the terms unspecified, undefined,  
11589 implementation-defined, or with the verbs “may” or “need not”, that the developers of  
11590 IEEE Std 1003.1-200x anticipate or sanction as suitable for profile or test method citation. All of |  
11591 these are merely warnings to conforming applications to avoid certain areas that can vary from |  
11592 system to system, and even over time on the same system. In many cases, these terms are used  
11593 explicitly to support extensions, but profiles should not anticipate and require such extensions; |  
11594 future versions of IEEE Std 1003.1-200x may do so. |

11595	<b>/ Rationale (Informative)</b>	
11596	<b>Part E:</b>	
11597	<b>Subprofiling Considerations</b>	
11598	<i>The Open Group</i>	
11599	<i>The Institute of Electrical and Electronics Engineers, Inc.</i>	



## Subprofiling Considerations (Informative)

11601

11602 This section contains further information to satisfy the requirement that the project scope enable  
 11603 subprofiling of IEEE Std 1003.1-200x. The original intent was to have included a set of options  
 11604 similar to the “Units of Functionality” contained in IEEE Std 1003.13-1998. However, as the  
 11605 development of IEEE Std 1003.1-200x continued, the standard developers felt it premature to fix  
 11606 these in normative text. The approach instead has been to include a general requirement in  
 11607 normative text regarding subprofiling and to include an informative section (here) containing a  
 11608 proposed set of subprofiling options.

### 11609 E.1 Subprofiling Option Groups

11610 The following Option Groups<sup>4</sup> are defined to support profiling. Systems claiming support to  
 11611 IEEE Std 1003.1-200x need not implement these options apart from the requirements stated in  
 11612 the Base Definitions volume of IEEE Std 1003.1-200x, Section 2.1.3, POSIX Conformance. These  
 11613 Option Groups allow profiles to subset the System Interfaces volume of IEEE Std 1003.1-200x by  
 11614 collecting sets of related functions.

11615 POSIX\_C\_LANG\_JUMP: Jump Functions

11616 *longjmp()*, *setjmp()*

11617 POSIX\_C\_LANG\_MATH: Maths Library

11618 *acos()*, *acosf()*, *acosh()*, *acoshf()*, *acoshl()*, *acosl()*, *asin()*, *asinf()*, *asinh()*, *asinhf()*, *asinhl()*,  
 11619 *asinl()*, *atan()*, *atan2()*, *atan2f()*, *atan2l()*, *atanf()*, *atanh()*, *atanhf()*, *atanhl()*, *atanl()*, *cabs()*,  
 11620 *cabsf()*, *cabsl()*, *cacos()*, *cacosf()*, *cacosh()*, *cacoshf()*, *cacoshl()*, *cacosl()*, *carg()*, *cargf()*, *cargl()*,  
 11621 *casin()*, *casinf()*, *casinh()*, *casinhf()*, *casinhl()*, *casinl()*, *catan()*, *catanf()*, *catanh()*, *catanhf()*,  
 11622 *catanhl()*, *catanl()*, *cbrt()*, *cbrtf()*, *cbrtl()*, *ccos()*, *ccosf()*, *ccosh()*, *ccoshf()*, *ccoshl()*, *ccosl()*,  
 11623 *ceil()*, *ceilf()*, *ceill()*, *cexp()*, *cexpf()*, *cexpl()*, *cimag()*, *cimagf()*, *cimagl()*, *clog()*, *clogf()*, *clogl()*,  
 11624 *conj()*, *conjf()*, *conjl()*, *copysign()*, *copysignf()*, *copysignl()*, *cos()*, *cosf()*, *cosh()*, *coshf()*,  
 11625 *coshl()*, *cosl()*, *cpow()*, *cpowf()*, *cpowl()*, *cproj()*, *cprojf()*, *cprojl()*, *creal()*, *crealf()*, *creall()*,  
 11626 *csin()*, *csinf()*, *csinh()*, *csinhf()*, *csinhl()*, *csinl()*, *csqrt()*, *csqrtf()*, *csqrtl()*, *ctan()*, *ctanf()*,  
 11627 *ctanh()*, *ctanhf()*, *ctanhl()*, *ctanl()*, *erf()*, *erfc()*, *erfcf()*, *erfcl()*, *erff()*, *erfl()*, *exp()*, *exp2()*,  
 11628 *exp2f()*, *exp2l()*, *expf()*, *expl()*, *expm1()*, *expm1f()*, *expm1l()*, *fabs()*, *fabsf()*, *fabsl()*, *fdim()*,  
 11629 *fdimf()*, *fdiml()*, *floor()*, *floorf()*, *floorl()*, *fma()*, *fmaf()*, *fmal()*, *fmax()*, *fmaxf()*, *fmaxl()*, *fmin()*,  
 11630 *fminf()*, *fminl()*, *fmod()*, *fmodf()*, *fmodl()*, *fpclassify()*, *frexp()*, *frexpf()*, *frexpl()*, *hypot()*,  
 11631 *hypotf()*, *hypotl()*, *ilogb()*, *ilogbf()*, *ilogbl()*, *isfinite()*, *isgreater()*, *isgreaterequal()*, *isinf()*,  
 11632 *isless()*, *islessequal()*, *islessgreater()*, *isnan()*, *isnormal()*, *isunordered()*, *ldexp()*, *ldexpf()*,  
 11633 *ldexpl()*, *lgamma()*, *lgammaf()*, *lgammal()*, *llrint()*, *llrintf()*, *llrintl()*, *llround()*, *llroundf()*,  
 11634 *llroundl()*, *log()*, *log10()*, *log10f()*, *log10l()*, *log1p()*, *log1pf()*, *log1pl()*, *log2()*, *log2f()*, *log2l()*,  
 11635 *logb()*, *logbf()*, *logbl()*, *logf()*, *logl()*, *lrint()*, *lrintf()*, *lrintl()*, *lround()*, *lroundf()*, *lroundl()*,  
 11636 *modf()*, *modff()*, *modfl()*, *nan()*, *nanf()*, *nanl()*, *nearbyint()*, *nearbyintf()*, *nearbyintl()*,  
 11637 *nextafter()*, *nextafterf()*, *nextafterl()*, *nexttoward()*, *nexttowardf()*, *nexttowardl()*, *pow()*, *powf()*,  
 11638 *powl()*, *remainder()*, *remainderf()*, *remainderl()*, *remquo()*, *remquof()*, *remquoq()*, *rint()*, *rintf()*,  
 11639 *rintl()*, *round()*, *roundf()*, *roundl()*, *scalbln()*, *scalblnf()*, *scalblnl()*, *scalbn()*, *scalbnf()*, *scalbnl()*,  
 11640 *signbit()*, *sin()*, *sinf()*, *sinh()*, *sinhf()*, *sinhl()*, *sinl()*, *sqrt()*, *sqrtf()*, *sqrtl()*, *tan()*, *tanf()*,

11641

11642 4. These are equivalent to the Units of Functionality from IEEE Std 1003.13-1998.

11643	<i>tanh()</i> , <i>tanhf()</i> , <i>tanhL()</i> , <i>tanl()</i> , <i>tgamma()</i> , <i>tgammaf()</i> , <i>tgammaL()</i> , <i>trunc()</i> , <i>truncf()</i> , <i>truncL()</i>	
11644	POSIX_C_LANG_SUPPORT: General ISO C Library	
11645	<i>abs()</i> , <i>asctime()</i> , <i>atof()</i> , <i>atoi()</i> , <i>atol()</i> , <i>bsearch()</i> , <i>calloc()</i> , <i>ctime()</i> , <i>difftime()</i> , <i>div()</i> ,	
11646	<i>feclearexcept()</i> , <i>fegetenv()</i> , <i>fegetexceptflag()</i> , <i>fegetround()</i> , <i>fehldexcept()</i> , <i>feraiseexcept()</i> ,	
11647	<i>fesetenv()</i> , <i>fesetexceptflag()</i> , <i>fesetround()</i> , <i>fetestexcept()</i> , <i>feupdateenv()</i> , <i>free()</i> , <i>gmtime()</i> ,	
11648	<i>imaxabs()</i> , <i>imaxdiv()</i> , <i>isalnum()</i> , <i>isalpha()</i> , <i>isblank()</i> , <i>iscntrl()</i> , <i>isdigit()</i> , <i>isgraph()</i> , <i>islower()</i> ,	
11649	<i>isprint()</i> , <i>ispunct()</i> , <i>isspace()</i> , <i>isupper()</i> , <i>isxdigit()</i> , <i>labs()</i> , <i>ldiv()</i> , <i>llabs()</i> , <i>lldiv()</i> , <i>localeconv()</i> ,	
11650	<i>localtime()</i> , <i>malloc()</i> , <i>memchr()</i> , <i>memcmp()</i> , <i>memcpy()</i> , <i>memmove()</i> , <i>memset()</i> , <i>mktime()</i> ,	
11651	<i>qsort()</i> , <i>rand()</i> , <i>realloc()</i> , <i>setlocale()</i> , <i>snprintf()</i> , <i>sprintf()</i> , <i>srand()</i> , <i>sscanf()</i> , <i>strcat()</i> , <i>strchr()</i> ,	
11652	<i>strcmp()</i> , <i>strcoll()</i> , <i>strcpy()</i> , <i>strcspn()</i> , <i>strerror()</i> , <i>strftime()</i> , <i>strlen()</i> , <i>strncat()</i> , <i>strncpy()</i> ,	
11653	<i>strncpy()</i> , <i>strpbrk()</i> , <i>strrchr()</i> , <i>strspn()</i> , <i>strstr()</i> , <i>strtod()</i> , <i>strtof()</i> , <i>strtoimax()</i> , <i>strtok()</i> , <i>strtol()</i> ,	
11654	<i>strtold()</i> , <i>strtoll()</i> , <i>strtol()</i> ,	
11655	<i>tzname</i> , <i>tzset()</i> , <i>va_arg()</i> , <i>va_copy()</i> , <i>va_end()</i> , <i>va_start()</i> , <i>vsprintf()</i> , <i>vsprintf()</i> , <i>vsscanf()</i>	
11656	POSIX_C_LANG_SUPPORT_R: Thread-Safe General ISO C Library	
11657	<i>asctime_r()</i> , <i>ctime_r()</i> , <i>gmtime_r()</i> , <i>localtime_r()</i> , <i>rand_r()</i> , <i>strerror_r()</i> , <i>strtok_r()</i>	
11658	POSIX_C_LANG_WIDE_CHAR: Wide-Character ISO C Library	
11659	<i>btowc()</i> , <i>iswalnum()</i> , <i>iswalpha()</i> , <i>iswblank()</i> , <i>iswcntrl()</i> , <i>iswctype()</i> , <i>iswdigit()</i> , <i>iswgraph()</i> ,	
11660	<i>iswlower()</i> , <i>iswprint()</i> , <i>iswpunct()</i> , <i>iswspace()</i> , <i>iswupper()</i> , <i>iswxdigit()</i> , <i>mblen()</i> , <i>mbrlen()</i> ,	
11661	<i>mbrtowc()</i> , <i>mbsinit()</i> , <i>mbsrtowcs()</i> , <i>mbstowcs()</i> , <i>mbtowc()</i> , <i>swprintf()</i> , <i>swscanf()</i> , <i>towctrans()</i> ,	
11662	<i>towlower()</i> , <i>towupper()</i> , <i>vswprintf()</i> , <i>vswscanf()</i> , <i>wcrtomb()</i> , <i>wcscat()</i> , <i>wcschr()</i> , <i>wcscmp()</i> ,	
11663	<i>wcscoll()</i> , <i>wcscpy()</i> , <i>wcscspn()</i> , <i>wcsftime()</i> , <i>wcslen()</i> , <i>wcsncat()</i> , <i>wcsncmp()</i> , <i>wcsncpy()</i> ,	
11664	<i>wcspbrk()</i> , <i>wcsrchr()</i> , <i>wcrtombs()</i> , <i>wcsspn()</i> , <i>wcsstr()</i> , <i>wcstod()</i> , <i>wcstof()</i> , <i>wcstoimax()</i> ,	
11665	<i>wcstok()</i> , <i>wcstol()</i> , <i>wcstold()</i> , <i>wcstoll()</i> , <i>wcstombs()</i> , <i>wcstoul()</i> , <i>wcstoull()</i> , <i>wcstoumax()</i> ,	
11666	<i>wcsxfrm()</i> , <i>wctob()</i> , <i>wctomb()</i> , <i>wctrans()</i> , <i>wctype()</i> , <i>wmemchr()</i> , <i>wmemcmp()</i> , <i>wmemcpy()</i> ,	
11667	<i>wmemmove()</i> , <i>wmemset()</i>	
11668	POSIX_C_LIB_EXT: General C Library Extension	
11669	<i>fnmatch()</i> , <i>getopt()</i> , <i>optarg</i> , <i>opterr</i> , <i>optind</i> , <i>optopt</i>	
11670	POSIX_DEVICE_IO: Device Input and Output	
11671	<i>FD_CLR()</i> , <i>FD_ISSET()</i> , <i>FD_SET()</i> , <i>FD_ZERO()</i> , <i>clearerr()</i> , <i>close()</i> , <i>fclose()</i> , <i>fdopen()</i> , <i>feof()</i> ,	
11672	<i>ferror()</i> , <i>fflush()</i> , <i>fgetc()</i> , <i>fgets()</i> , <i>fileno()</i> , <i>fopen()</i> , <i>fprintf()</i> , <i>fputc()</i> , <i>fputs()</i> , <i>fread()</i> , <i>freopen()</i> ,	
11673	<i>fscanf()</i> , <i>fwrite()</i> , <i>getc()</i> , <i>getchar()</i> , <i>gets()</i> , <i>open()</i> , <i>perror()</i> , <i>printf()</i> , <i>pselect()</i> , <i>putc()</i> , <i>putchar()</i> ,	
11674	<i>puts()</i> , <i>read()</i> , <i>scanf()</i> , <i>select()</i> , <i>setbuf()</i> , <i>setvbuf()</i> , <i>stderr</i> , <i>stdin</i> , <i>stdout</i> , <i>ungetc()</i> , <i>vfprintf()</i> ,	
11675	<i>vscanf()</i> , <i>vprintf()</i> , <i>vscanf()</i> , <i>write()</i>	
11676	POSIX_DEVICE_SPECIFIC: General Terminal	
11677	<i>cfgetispeed()</i> , <i>cfgetospeed()</i> , <i>cfsetispeed()</i> , <i>cfsetospeed()</i> , <i>ctermid()</i> , <i>isatty()</i> , <i>tcdrain()</i> , <i>tclflow()</i> ,	
11678	<i>tcflush()</i> , <i>tcgetattr()</i> , <i>tcsendbreak()</i> , <i>tcsetattr()</i> , <i>ttyname()</i>	
11679	POSIX_DEVICE_SPECIFIC_R: Thread-Safe General Terminal	
11680	<i>ttyname_r()</i>	
11681	POSIX_FD_MGMT: File Descriptor Management	
11682	<i>dup()</i> , <i>dup2()</i> , <i>fcntl()</i> , <i>fgetpos()</i> , <i>fseek()</i> , <i>fseeko()</i> , <i>fsetpos()</i> , <i>ftell()</i> , <i>ftello()</i> , <i>ftruncate()</i> , <i>lseek()</i> ,	
11683	<i>rewind()</i>	
11684	POSIX_FIFO: FIFO	
11685	<i>mkfifo()</i>	
11686	POSIX_FILE_ATTRIBUTES: File Attributes	
11687	<i>chmod()</i> , <i>chown()</i> , <i>fchmod()</i> , <i>fchown()</i> , <i>umask()</i>	
11688	POSIX_FILE_LOCKING: Thread-Safe Stdio Locking	
11689	<i>flockfile()</i> , <i>ftrylockfile()</i> , <i>funlockfile()</i> , <i>getc_unlocked()</i> , <i>getchar_unlocked()</i> , <i>putc_unlocked()</i> ,	

11690	<i>putchar_unlocked()</i>	
11691	POSIX_FILE_SYSTEM: File System	
11692	<i>access(), chdir(), closedir(), creat(), fpathconf(), fstat(), getcwd(), link(), mkdir(), opendir(),</i>	
11693	<i>pathconf(), readdir(), remove(), rename(), rewinddir(), rmdir(), stat(), tmpfile(), tmpnam(),</i>	
11694	<i>unlink(), utime()</i>	
11695	POSIX_FILE_SYSTEM_EXT: File System Extensions	
11696	<i>glob(), globfree()</i>	
11697	POSIX_FILE_SYSTEM_R: Thread-Safe File System	
11698	<i>readdir_r()</i>	
11699	POSIX_JOB_CONTROL: Job Control	
11700	<i>setpgid(), tcgetpgrp(), tcsetpgrp()</i>	
11701	POSIX_MULTI_PROCESS: Multiple Processes	
11702	<i>_Exit(), _exit(), assert(), atexit(), clock(), execl(), execl(), execlp(), execv(), execve(), execvp(),</i>	
11703	<i>exit(), fork(), getpgrp(), getpid(), getppid(), setsid(), sleep(), times(), wait(), waitpid()</i>	
11704	POSIX_NETWORKING: Networking	
11705	<i>accept(), bind(), connect(), endhostent(), endnetent(), endprotoent(), endservent(),</i>	
11706	<i>freeaddrinfo(), gai_strerror(), getaddrinfo(), gethostbyaddr(), gethostbyname(), gethostent(),</i>	
11707	<i>gethostname(), getnameinfo(), getnetbyaddr(), getnetbyname(), getnetent(), getpeername(),</i>	
11708	<i>getprotobyname(), getprotobynumber(), getprotoent(), getservbyname(), getservbyport(),</i>	
11709	<i>getservent(), getsockname(), getsockopt(), h_errno, htonl(), htons(), if_freenameindex(),</i>	
11710	<i>if_indextoname(), if_nameindex(), if_nametoindex(), inet_addr(), inet_ntoa(), inet_ntop(),</i>	
11711	<i>inet_pton(), listen(), ntohl(), ntohs(), recv(), recvfrom(), recvmsg(), send(), sendmsg(), sendto(),</i>	
11712	<i>sethostent(), setnetent(), setprotoent(), setservent(), setsockopt(), shutdown(), socket(),</i>	
11713	<i>socketpair()</i>	
11714	POSIX_PIPE: Pipe	
11715	<i>pipe()</i>	
11716	POSIX_REGEX: Regular Expressions	
11717	<i>regcomp(), regerror(), regexexec(), regfree()</i>	
11718	POSIX_SHELL_FUNC: Shell and Utilities	
11719	<i>pclose(), popen(), system(), wordexp(), wordfree()</i>	
11720	POSIX_SIGNALS: Signal	
11721	<i>abort(), alarm(), kill(), pause(), raise(), sigaction(), sigaddset(), sigdelset(), sigemptyset(),</i>	
11722	<i>sigfillset(), sigismember(), signal(), sigpending(), sigprocmask(), sigsuspend(), sigwait()</i>	
11723	POSIX_SIGNAL_JUMP: Signal Jump Functions	
11724	<i>siglongjmp(), sigsetjmp()</i>	
11725	POSIX_SINGLE_PROCESS: Single Process	
11726	<i>confstr(), environ, errno, getenv(), setenv(), sysconf(), uname(), unsetenv()</i>	
11727	POSIX_SYMBOLIC_LINKS: Symbolic Links	
11728	<i>lstat(), readlink(), symlink()</i>	
11729	POSIX_SYSTEM_DATABASE: System Database	
11730	<i>getgrgid(), getgrnam(), getpwnam(), getpwuid()</i>	
11731	POSIX_SYSTEM_DATABASE_R: Thread-Safe System Database	
11732	<i>getgrgid_r(), getgrnam_r(), getpwnam_r(), getpwuid_r()</i>	

11733	POSIX_USER_GROUPS: User and Group	
11734	<i>getegid()</i> , <i>geteuid()</i> , <i>getgid()</i> , <i>getgroups()</i> , <i>getlogin()</i> , <i>getuid()</i> , <i>setegid()</i> , <i>seteuid()</i> , <i>setgid()</i> ,	
11735	<i>setuid()</i>	
11736	POSIX_USER_GROUPS_R: Thread-Safe User and Group	
11737	<i>getlogin_r()</i>	
11738	POSIX_WIDE_CHAR_DEVICE_IO: Device Input and Output	
11739	<i>fgetwc()</i> , <i>fgetws()</i> , <i>fputwc()</i> , <i>fputws()</i> , <i>fwide()</i> , <i>fwprintf()</i> , <i>fwscanf()</i> , <i>getwc()</i> , <i>getwchar()</i> ,	
11740	<i>putwc()</i> , <i>putwchar()</i> , <i>ungetwc()</i> , <i>vfwprintf()</i> , <i>vfwscanf()</i> , <i>vwprintf()</i> , <i>vwscanf()</i> , <i>wprintf()</i> ,	
11741	<i>wscanf()</i>	
11742	XSI_C_LANG_SUPPORT: XSI General C Library	
11743	<i>_tolower()</i> , <i>_toupper()</i> , <i>a64l()</i> , <i>daylight()</i> , <i>drand48()</i> , <i>erand48()</i> , <i>ffs()</i> , <i>getcontext()</i> , <i>getdate()</i> ,	
11744	<i>getsubopt()</i> , <i>hcreate()</i> , <i>hdestroy()</i> , <i>hsearch()</i> , <i>iconv()</i> , <i>iconv_close()</i> , <i>iconv_open()</i> , <i>initstate()</i> ,	
11745	<i>insque()</i> , <i>isascii()</i> , <i>jrand48()</i> , <i>l64a()</i> , <i>lcong48()</i> , <i>lfind()</i> , <i>lrand48()</i> , <i>lsearch()</i> , <i>makecontext()</i> ,	
11746	<i>memccpy()</i> , <i>mrand48()</i> , <i>nrand48()</i> , <i>random()</i> , <i>remque()</i> , <i>seed48()</i> , <i>setcontext()</i> , <i>setstate()</i> ,	
11747	<i>signgam()</i> , <i>srand48()</i> , <i>srandom()</i> , <i>strcasecmp()</i> , <i>strdup()</i> , <i>strfmon()</i> , <i>strncasecmp()</i> , <i>strptime()</i> ,	
11748	<i>swab()</i> , <i>swapcontext()</i> , <i>tdelete()</i> , <i>tfind()</i> , <i>timezone()</i> , <i>toascii()</i> , <i>tsearch()</i> , <i>twalk()</i>	
11749	XSI_DBM: XSI Database Management	
11750	<i>dbm_clearerr()</i> , <i>dbm_close()</i> , <i>dbm_delete()</i> , <i>dbm_error()</i> , <i>dbm_fetch()</i> , <i>dbm_firstkey()</i> ,	
11751	<i>dbm_nextkey()</i> , <i>dbm_open()</i> , <i>dbm_store()</i>	
11752	XSI_DEVICE_IO: XSI Device Input and Output	
11753	<i>fntmsg()</i> , <i>poll()</i> , <i>pread()</i> , <i>pwrite()</i> , <i>readv()</i> , <i>writenv()</i>	
11754	XSI_DEVICE_SPECIFIC: XSI General Terminal	
11755	<i>grantpt()</i> , <i>posix_openpt()</i> , <i>ptsname()</i> , <i>unlockpt()</i>	
11756	XSI_DYNAMIC_LINKING: XSI Dynamic Linking	
11757	<i>dlclose()</i> , <i>dlerror()</i> , <i>dlopen()</i> , <i>dlsym()</i>	
11758	XSI_FD_MGMT: XSI File Descriptor Management	
11759	<i>truncate()</i>	
11760	XSI_FILE_SYSTEM: XSI File System	
11761	<i>basename()</i> , <i>dirname()</i> , <i>fchdir()</i> , <i>fstatvfs()</i> , <i>ftw()</i> , <i>getwd()</i> , <i>lchown()</i> , <i>lockf()</i> , <i>mknod()</i> ,	
11762	<i>mkstemp()</i> , <i>mktemp()</i> , <i>nftw()</i> , <i>realpath()</i> , <i>seekdir()</i> , <i>statvfs()</i> , <i>sync()</i> , <i>telldir()</i> , <i>tempnam()</i> ,	
11763	<i>utimes()</i>	
11764	XSI_I18N: XSI Internationalization	
11765	<i>catclose()</i> , <i>catgets()</i> , <i>catopen()</i> , <i>nl_langinfo()</i>	
11766	XSI_IPC: XSI Interprocess Communication	
11767	<i>ftok()</i> , <i>msgctl()</i> , <i>msgget()</i> , <i>msgrcv()</i> , <i>msgsnd()</i> , <i>semctl()</i> , <i>semget()</i> , <i>semop()</i> , <i>shmat()</i> , <i>shmctl()</i> ,	
11768	<i>shmdt()</i> , <i>shmget()</i>	
11769	XSI_JOB_CONTROL: XSI Job Control	
11770	<i>tcgetsid()</i>	
11771	XSI_JUMP: XSI Jump Functions	
11772	<i>_longjmp()</i> , <i>_setjmp()</i>	
11773	XSI_MATH: XSI Maths Library	
11774	<i>j0()</i> , <i>j1()</i> , <i>jn()</i> , <i>scalb()</i> , <i>y0()</i> , <i>y1()</i> , <i>yn()</i>	
11775	XSI_MULTI_PROCESS: XSI Multiple Process	
11776	<i>getpgid()</i> , <i>getpriority()</i> , <i>getrlimit()</i> , <i>getrusage()</i> , <i>getsid()</i> , <i>nice()</i> , <i>setpgrp()</i> , <i>setpriority()</i> ,	
11777	<i>setrlimit()</i> , <i>ulimit()</i> , <i>usleep()</i> , <i>vfork()</i> , <i>waitid()</i>	

11778	XSI_SIGNALS: XSI Signal	
11779	<i>bsd_signal()</i> , <i>killpg()</i> , <i>sigaltstack()</i> , <i>sighold()</i> , <i>sigignore()</i> , <i>siginterrupt()</i> , <i>sigpause()</i> , <i>sigrelse()</i> ,	
11780	<i>sigset()</i> , <i>ualarm()</i>	
11781	XSI_SINGLE_PROCESS: XSI Single Process	
11782	<i>ftime()</i> , <i>gethostid()</i> , <i>gettimeofday()</i> , <i>putenv()</i>	
11783	XSI_SYSTEM_DATABASE: XSI System Database	
11784	<i>endpwent()</i> , <i>getpwent()</i> , <i>setpwent()</i>	
11785	XSI_SYSTEM_LOGGING: XSI System Logging	
11786	<i>closelog()</i> , <i>openlog()</i> , <i>setlogmask()</i> , <i>syslog()</i>	
11787	XSI_THREAD_MUTEX_EXT: XSI Thread Mutex Extensions	
11788	<i>pthread_mutexattr_gettype()</i> , <i>pthread_mutexattr_settype()</i>	
11789	XSI_THREADS_EXT: XSI Threads Extensions	
11790	<i>pthread_attr_getguardsize()</i> , <i>pthread_attr_getstack()</i> , <i>pthread_attr_setguardsize()</i> ,	
11791	<i>pthread_attr_setstack()</i> , <i>pthread_getconcurrency()</i> , <i>pthread_setconcurrency()</i>	
11792	XSI_TIMERS: XSI Timers	
11793	<i>getitimer()</i> , <i>setitimer()</i>	
11794	XSI_USER_GROUPS: XSI User and Group	
11795	<i>endgrent()</i> , <i>endutxent()</i> , <i>getgrent()</i> , <i>getutxent()</i> , <i>getutxid()</i> , <i>getutxline()</i> , <i>pututxline()</i> ,	
11796	<i>setgrent()</i> , <i>setregid()</i> , <i>setreuid()</i> , <i>setutxent()</i>	
11797	XSI_WIDE_CHAR: XSI Wide-Character Library	
11798	<i>wcswcs()</i> , <i>wcswidth()</i> , <i>wcwidth()</i>	



# Index

1

2	/dev/tty .....	3311	_POSIX_THREAD_PRIO_PROTECT .....	3572
3	/etc/passwd .....	3324	_POSIX_THREAD_PROCESS_SHARED .....	3572
4	<pthread.h> .....	3450	_POSIX_THREAD_SAFE_FUNCTIONS .....	3572
5	_asm_builtin_atoi() .....	3374	_POSIX_THREAD_SPORADIC_SERVER .....	3572
6	_exit() .....	3390, 3408	_POSIX_TIMEOUTS .....	3572
7	_Exit() .....	3559	_POSIX_TIMERS .....	3572
8	_exit() .....	3559	_POSIX_TRACE .....	3573
9	_longjmp() .....	3562	_POSIX_TRACE_EVENT_FILTER .....	3573
10	_POSIX_ADVISORY_INFO .....	3568	_POSIX_TRACE_INHERIT .....	3573
11	_POSIX_ASYNCHRONOUS_IO .....	3568	_POSIX_TRACE_LOG .....	3573
12	_POSIX_BARRIERS .....	3568	_POSIX_TYPED_MEMORY_OBJECTS .....	3573
13	_POSIX_CHOWN_RESTRICTED .....	3568	_POSIX_TZNAME_MAX .....	3346
14	_POSIX_CLOCK_SELECTION .....	3569	_SC_PAGESIZE .....	3407, 3409
15	_POSIX_CPUTIME .....	3569	_setjmp() .....	3562
16	_POSIX_C_SOURCE .....	3375, 3379	__errno() .....	3382
17	_POSIX_FSYNC .....	3569	abort() .....	3559
18	_POSIX_IPV6 .....	3569	access .....	3560
19	_POSIX_JOB_CONTROL .....	3569, 3573	access() .....	3323
20	_POSIX_MAPPED_FILES .....	3569	active trace stream .....	3492
21	_POSIX_MEMLOCK .....	3569	adb, rationale for omission .....	3548
22	_POSIX_MEMLOCK_RANGE .....	3569	addressing .....	3466
23	_POSIX_MEMORY_PROTECTION .....	3570	advisory information .....	3394
24	_POSIX_MESSAGE_PASSING .....	3570	aio_cancel() .....	3402-3403
25	_POSIX_MONOTONIC_CLOCK .....	3570	aio_fsync() .....	3387, 3401
26	_POSIX_PRIORITIZED_IO .....	3570	AIO_LISTIO_MAX .....	3573
27	_POSIX_PRIORITY_SCHEDULING .....	3570	AIO_MAX .....	3573
28	_POSIX_REALTIME_SIGNALS .....	3570	AIO_PRIO_DELTA_MAX .....	3573
29	_POSIX_REGEX .....	3570	aio_read() .....	3403
30	_POSIX_RTSIG_MAX .....	3384, 3575	aio_suspend() .....	3402, 3427
31	_POSIX_SAVED_IDS .....	3570	aio_write() .....	3403
32	_POSIX_SEMAPHORES .....	3571	alarm .....	3559
33	_POSIX_SHARED_MEMORY_OBJECTS .....	3571	alarm() .....	3392, 3426
34	_POSIX_SHELL .....	3571	alias .....	3562
35	_POSIX_SOURCE .....	3375	alias substitution .....	3520
36	_POSIX_SPAWN .....	3571	AND lists .....	3536
37	_POSIX_SPINLOCKS .....	3571	application instrumentation .....	3480
38	_POSIX_SPORADIC_SERVER .....	3571	appropriate privilege .....	3303
39	_POSIX_SS_REPL_MAX .....	3423	appropriate privileges .....	3303, 3315
40	_POSIX_SYNCHRONIZED_IO .....	3571	ar .....	3565-3566
41	_POSIX_SYNC_IO .....	3569	ARG_MAX .....	3312, 3510, 3573
42	_POSIX_THREADS .....	3571	arithmetic expansion .....	3527
43	_POSIX_THREAD_ATTR_STACKADDR .....	3571	as, rationale for omission .....	3548
44	_POSIX_THREAD_ATTR_STACKSIZE .....	3572	asa .....	3563, 3565-3566
45	_POSIX_THREAD_PRIORITY_SCHEDULING .....	3572	ASCII .....	3314
46	.....	3572	async-cancel safety .....	3464
47	_POSIX_THREAD_PRIO_INHERIT .....	3572	async-signal-safe .....	3391

48	asynchronous error .....	3467	canonical mode input processing .....	3358
49	asynchronous I/O.....	3401, 3560	case .....	3536
50	asynchronous lists.....	3535	case folding.....	3324-3325
51	at .....	3563	cat.....	3516
52	atexit() .....	3463	catclose() .....	3564
53	atoi().....	3374	catgets().....	3564
54	awk.....	3563, 3565	catopen() .....	3564
55	background.....	3308-3311, 3357	cd.....	3563
56	backslash .....	3518	CEO .....	3351
57	banner, rationale for omission.....	3548	character .....	3303
58	barriers.....	3443	character encoding.....	3332
59	barrier_wait().....	3463	character set.....	3331
60	basename.....	3562-3563	character set description file .....	3332
61	basic regular expression.....	3350	character set, portable filename .....	3314
62	batch.....	3563	character, rationale.....	3303
63	general concepts .....	3545	CHARCLASS_NAME_MAX .....	3337
64	batch environment .....	3542	CHAR_MAX.....	3339
65	option definitions.....	3543	chgrp .....	3516, 3563-3564
66	batch environment utilities		CHILD_MAX .....	3494, 3510, 3535, 3573
67	common behavior .....	3548	chmod.....	3516, 3560, 3563-3564
68	batch services .....	3547	chmod().....	3316
69	batch systems		chown.....	3516, 3560, 3563-3564
70	historical implementations .....	3542	chown() .....	3316
71	history.....	3542	chroot().....	3314
72	bc.....	3562-3563, 3567	chroot, rationale for omission.....	3548
73	BC_BASE_MAX .....	3510	cksum.....	3516, 3563-3564
74	BC_DIM_MAX.....	3510	clock .....	3423
75	BC_SCALE_MAX .....	3510	clock tick.....	3303
76	bg .....	3563	clock tick, rationale .....	3303
77	bounded response .....	3561	CLOCKRES_MIN.....	3574
78	bracket expression		clocks.....	3423
79	grammar.....	3354	clock_getcpuclockid().....	3429, 3431
80	BRE		clock_nanosleep() .....	3427-3428
81	expression anchoring .....	3352	CLOCK_PROCESS_CPUTIME_ID .....	3429, 3431
82	grammar lexical conventions .....	3354	CLOCK_REALTIME.....	3423-3426, 3428
83	matching a collating element.....	3350	CLOCK_THREAD_CPUTIME_ID .....	3429, 3431
84	matching a single character .....	3350	close.....	3560
85	matching multiple characters.....	3351	close() .....	3408
86	ordinary character.....	3350	closedir.....	3560
87	periods.....	3350	closelog().....	3564
88	precedence .....	3352	cmp.....	3516, 3563
89	special character .....	3350	col, rationale for omission .....	3548
90	BSD.....	3305, 3308, 3311, 3314	collating element order .....	3351
91	.....	3356-3358, 3383-3386, 3389-3390, 3493	COLL_WEIGHTS_MAX .....	3510
92	C Shell .....	3308-3310	column position.....	3304
93	C-language extensions.....	3557, 3562	COLUMNS .....	3346
94	c99.....	3565	comm .....	3563
95	calendar, rationale for omission.....	3548	command.....	3304, 3562
96	cancel, rationale for omission .....	3548	command execution.....	3533
97	cancelation cleanup handler.....	3462-3463	command language .....	3557, 3562
98	cancelation cleanup stack.....	3462	command search.....	3533

## Index

99	command substitution .....	3526	directory files.....	3354
100	compilation environment.....	3375	directory protection .....	3323
101	complex data manipulation.....	3557, 3563	directory structure.....	3354
102	compound commands.....	3536	directory, root.....	3314
103	concurrent execution.....	3323	dirname.....	3562-3563
104	conditional construct		dis, rationale for omission .....	3549
105	case .....	3536	display .....	3305
106	if.....	3537	dot.....	3306
107	configurable limits.....	3567, 3573	dot-dot .....	3306, 3314, 3328
108	configuration interrogation .....	3556, 3559	double-quotes.....	3518
109	configuration options .....	3566	du.....	3516, 3563-3564
110	shell and utilities .....	3566	dup.....	3560
111	system interfaces.....	3568	dup() .....	3408
112	conformance .....	3295, 3299, 3301, 3303, 3325, 3493	dup2 .....	3560
113	conformance document .....	3295	dup2() .....	3408
114	conformance document, rationale .....	3295	EBUSY .....	3381, 3450
115	conforming application.....	3301, 3383, 3513, 3515	ECANCELED .....	3380
116	conforming application, strictly .....	3299, 3301, 3390	echo .....	3562
117	confstr .....	3560	ECHOE .....	3360
118	connection indication queue.....	3467	ECHOK.....	3360
119	control mode .....	3360	ECHONL.....	3360
120	controlling terminal .....	3305, 3357, 3560	ed.....	3563-3564
121	core .....	3324	EDOM .....	3381
122	covert channel.....	3325	EFAULT .....	3379-3380
123	cp.....	3516, 3563	effective user ID.....	3323
124	cpio, rationale for omission.....	3548	EFTYPE.....	3380
125	cpp, rationale for omission.....	3548	EILSEQ.....	3381
126	creat() .....	3408, 3516	EINPROGRESS .....	3402
127	crontab.....	3563	EINTR .....	3380, 3382, 3392
128	CSIZE.....	3360	EINVAL .....	3380
129	csplit.....	3563	ELOOP .....	3380
130	ctags.....	3565	emacs, rationale for omission .....	3549
131	ctime() .....	3561	ENAMETOOLONG.....	3381
132	cu, rationale for omission .....	3548	endgrent() .....	3322
133	cut .....	3563	endpwent() .....	3322
134	data access.....	3556, 3560	ENOMEM .....	3381
135	data type.....	3493	ENOSYS .....	3381, 3405
136	date .....	3564	ENOTSUP .....	3381
137	dc, rationale for omission .....	3549	ENOTTY.....	3355, 3380-3381
138	dd.....	3516, 3563	env.....	3562, 3564
139	DELAYTIMER_MAX.....	3574	environment access .....	3556, 3560
140	determinism.....	3556	environment variable .....	3345
141	device number.....	3305	definition.....	3345
142	device, logical.....	3311	EPERM.....	3459
143	df.....	3516, 3563-3564	EPIPE.....	3381
144	diff.....	3563	Epoch.....	3306, 3329, 3423
145	dircmp, rationale for omission .....	3549	ERANGE .....	3381
146	direct I/O .....	3305	ERASE.....	3358
147	directory .....	3305	ERE .....	3352
148	directory device .....	3354	alternation.....	3353
149	directory entry.....	3305	bracket expression .....	3353

150	expression anchoring .....	3353	file classes.....	3306
151	grammar.....	3354	file descriptors.....	3392
152	grammar lexical conventions .....	3354	file format notation .....	3331
153	matching a collating element.....	3353	file hierarchy.....	3323
154	matching a single character .....	3353	file hierarchy manipulation .....	3557, 3563
155	matching multiple characters.....	3353	file permissions.....	3323, 3357
156	ordinary character.....	3353	file system .....	3306
157	periods.....	3353	file system, mounted .....	3312
158	precedence .....	3353	file system, root.....	3314
159	special character .....	3353	file system, root of.....	3315
160	EROFS.....	3381	file times update .....	3325
161	errno .....	3379	file, passwd.....	3313
162	per-thread .....	3382	filename .....	3306
163	error conditions.....	3330	filenames .....	3324
164	error handling .....	3540	fileno() .....	3313
165	error numbers .....	3379, 3383	filtering of trace event types .....	3489
166	errors .....	3531	filtering trace event types .....	3489
167	escape character.....	3518	find.....	3563-3564
168	ex .....	3562-3564	flockfile().....	3382
169	exec family .....	3308, 3406, 3463, 3538, 3559	fnmatch() .....	3565
170	exec() .....	3486	fold.....	3563
171	exec, family.....	3511	fopen .....	3560
172	Execution Time Monitoring.....	3429	fopen().....	3307, 3516
173	execution time, measurement.....	3325	for loop .....	3536
174	exit status .....	3531	foreground .....	3308-3311, 3356-3357
175	exit() .....	3390, 3559	fork().....	3308, 3356, 3397, 3406
176	EXIT_FAILURE.....	3561	.....	3408, 3486, 3494-3495, 3559
177	EXIT_SUCCESS .....	3561	fort77 .....	3565
178	expand .....	3563	fpathconf .....	3560
179	expr.....	3562-3563	fpathconf().....	3559
180	EXPR_NEST_MAX.....	3510	fputc .....	3560
181	extended regular expression.....	3352	fread .....	3560
182	extended security controls .....	3323	free() .....	3391, 3462
183	false.....	3562	fseek.....	3560
184	fc .....	3562	fseeko().....	3493
185	fchmod.....	3560	fsetpos.....	3560
186	fclose .....	3560	fsetpos() .....	3493
187	fcntl.....	3560	fstat .....	3559-3560
188	fcntl().....	3355, 3380, 3408, 3411, 3493	fsync().....	3401
189	fcntl() locks.....	3465	truncate .....	3560
190	fdopen() .....	3408	truncate().....	3407-3408, 3410
191	FD_CLOEXEC.....	3411	ftw() .....	3516
192	feature test macro.....	3375-3376, 3493	function definition command.....	3537
193	fg .....	3563	functions	
194	fgetc .....	3560	implementation of.....	3374
195	fgetpos().....	3493	use of.....	3374
196	field splitting .....	3529	fwrite .....	3560
197	FIFO.....	3306, 3314, 3417	genat .....	3564
198	FIFO special file .....	3306	general terminal interface.....	3355
199	file .....	3306	getc .....	3560
200	file access permissions .....	3323	getc().....	3453

## Index

201	getch	3560	implementation, hosted	3307
202	getconf	3509, 3559, 3564-3565	implementation, native	3312
203	getegid	3559	implementation, specific	3307
204	geteuid	3559	implementation-defined	3296-3297
205	getgid	3559	implementation-defined, rationale	3296
206	getgrent()	3322	incomplete pathname	3308
207	getgrgid	3559	input file descriptor	
208	getgrgid()	3322, 3454	duplication	3531
209	getgrnam	3559	input mode	3359
210	getgrnam()	3322, 3325, 3454	input processing	3358
211	getgroups()	3315	canonical mode	3358
212	getlogin	3559	non-canonical mode	3358
213	getopt()	3362, 3564-3565	inter-user communication	3558, 3564
214	getopts	3565	interactive facilities	3557, 3562
215	getpgrp()	3310	interface characteristics	3356
216	getpid	3559	interfaces	3466
217	getpid()	3392	internationalization variable	3345
218	getppid	3559	interprocess communication	3393
219	getpriority()	3418	invalid	
220	getpwent()	3322	use in RE	3349
221	getpwnam	3559	ioctl()	3355, 3381
222	getpwnam()	3322, 3325, 3454	IPC	3393
223	getpwuid	3559	ISO C standard	3301, 3303, 3329, 3355
224	getpwuid()	3322, 3454	3374-3376, 3378-3379, 3381, 3383, 3390, 3493	
225	getrlimit()	3516	ISO/IEC 646: 1991 standard	3314
226	getrusage()	3431	ISTRIP	3359
227	getty	3357	job control	3308-3311, 3313
228	getuid	3559	3356-3357, 3383, 3390, 3560	
229	getuid()	3392, 3494	job control, implementing applications	3310
230	gid_t	3322	job control, implementing shells	3308
231	glob()	3565	job control, implementing systems	3311
232	globfree()	3565	jobs	3563
233	gmtime()	3329	join	3563
234	grep	3563-3564	kernel	3311
235	group database	3307	kernel entity	3456
236	group database access	3322	kill	3563
237	group file	3307	kill()	3383-3385, 3387-3388, 3390, 3494
238	grouping commands	3536	last close	3408
239	head	3563	lchown()	3316
240	headers	3364	LC_COLLATE	3337
241	here-document	3531	LC_CTYPE	3336
242	historical implementations	3307	LC_MESSAGES	3341
243	hosted implementation	3307	LC_MONETARY	3339
244	ICANON	3358, 3360	LC_NUMERIC	3340, 3363
245	iconv()	3564	LC_TIME	3340
246	iconv_close()	3564	ld, rationale for omission	3549
247	iconv_open()	3564	legacy	3296
248	id	3564	legacy, rationale	3296
249	if	3537	lex	3565-3566
250	implementation	3307	library routine	3311
251	implementation, historical	3307	line, rationale for omission	3549

252	LINES.....	3346	MAX_CANON.....	3358, 3511, 3574
253	LINE_MAX.....	3312, 3321, 3510	MAX_INPUT.....	3511, 3574
254	link.....	3560	may.....	3296
255	link().....	3305, 3316	may, rationale.....	3296
256	LINK_MAX.....	3511, 3574	MCL_FUTURE.....	3405
257	lint, rationale for omission.....	3549	MCL_INHERIT.....	3406
258	lio_listio().....	3387, 3402	memlockall().....	3405
259	lists.....	3535	memory locking.....	3404
260	ln.....	3516, 3563	memory management.....	3404, 3560
261	local mode.....	3360	memory management unit.....	3405
262	locale.....	3334, 3564	memory object.....	3312
263	definition example.....	3341	memory synchronization.....	3326
264	definition grammar.....	3341	memory-resident.....	3311
265	grammar.....	3341	mesg.....	3564
266	lexical conventions.....	3341	message passing.....	3395, 3560-3561
267	locale configuration.....	3558, 3564	message queue.....	3395
268	locale definition.....	3335	mkdir.....	3560, 3563
269	localedef.....	3564-3565	mkfifo.....	3563
270	localtime().....	3329, 3453	mkfifo().....	3307
271	logger.....	3564	mknod().....	3307
272	logical device.....	3311	mknod, rationale for omission.....	3549
273	login, rationale for omission.....	3549	mktime().....	3329
274	LOGIN_NAME_MAX.....	3574	mmap().....	3407-3409, 3411
275	LOGNAME.....	3347	MMU.....	3405
276	logname.....	3564	modem disconnect.....	3359
277	longjmp().....	3380, 3391, 3460, 3462, 3562	monotonic clock.....	3427
278	LONG_MAX.....	3361	more.....	3562, 3564
279	LONG_MIN.....	3361	mount point.....	3312
280	lorder, rationale for omission.....	3549	mount().....	3312
281	lp.....	3564	mounted file system.....	3312
282	lpstat, rationale for omission.....	3549	mprotect().....	3408
283	ls.....	3516, 3563	mq_open().....	3396
284	lseek.....	3560	MQ_OPEN_MAX.....	3574
285	lseek().....	3401-3402, 3408, 3452, 3493	MQ_PRIO_MAX.....	3574
286	lseeko().....	3493	mq_receive().....	3397
287	lstat.....	3559-3560	mq_send().....	3397
288	lstat().....	3316, 3516	mq_timedreceive().....	3428
289	lutime().....	3316	mq_timedsend().....	3428
290	macro.....	3297	msg*().....	3393
291	mail, rationale for omission.....	3549	msgctl().....	3393
292	mailx.....	3562, 3564	msgget().....	3393
293	make.....	3565-3566	msgrcv().....	3393
294	malloc().....	3391, 3411, 3413, 3441	msgsnd().....	3393
295	.....	3453-3454, 3462, 3464	msync().....	3408
296	man.....	3562	multi-byte character.....	3306, 3358-3359
297	map.....	3311	multiple tasks.....	3557, 3563
298	mapped.....	3311	munmap().....	3407-3409, 3411, 3414
299	margin code notation.....	3299	mutex.....	3445
300	mathematical functions		mutex attributes	
301	error conditions.....	3330	extended.....	3448
302	NaN arguments.....	3330	mutex initialization.....	3459

## Index

303	mv.....	3516, 3563	parent directory .....	3314
304	name space.....	3376	passwd file .....	3313
305	name space pollution.....	3375-3376	passwd, rationale for omission.....	3550
306	NAME_MAX.....	3511, 3574	paste .....	3563
307	NaN arguments .....	3330	patch.....	3563-3565
308	nanosleep().....	3424, 3426, 3428, 3561	PATH.....	3347
309	native implementation .....	3312	pathchk .....	3563
310	newgrp.....	3564	pathconf.....	3560
311	news, rationale for omission.....	3550	pathconf() .....	3307, 3509, 3559
312	NGROUPS_MAX .....	3315, 3511, 3568, 3574	pathname expansion.....	3529
313	nice.....	3563	pathname resolution.....	3327
314	nice value .....	3312	pathname, incomplete.....	3308
315	nice() .....	3418	PATH_MAX.....	3381, 3511, 3574
316	nl_langinfo().....	3561	pattern matching	
317	nm.....	3565	multiple character .....	3541
318	noclobber option.....	3529	notation .....	3540
319	nohup.....	3563	single character.....	3540
320	non-canonical mode input processing.....	3358	patterns	
321	non-printable.....	3321	filename expansion.....	3541
322	NQS.....	3543	pause .....	3559
323	obsolescent.....	3296	pause() .....	3389, 3392
324	obsolescent, rationale .....	3296	pax .....	3563-3565
325	od.....	3563, 3565	pcat, rationale for omission.....	3550
326	open.....	3560	pclose().....	3565
327	open file description .....	3313	Pending error.....	3467
328	open file descriptors .....	3531	per-thread errno.....	3382
329	open() .....	3307, 3357, 3408-3410, 3516	performance enhancements.....	3556
330	opendir.....	3560	pg, rationale for omission.....	3550
331	openlog().....	3564	PID_MAX.....	3494
332	OPEN_MAX.....	3511, 3574-3575	pipe .....	3308-3309, 3314, 3560
333	option definitions .....	3543	pipe() .....	3389
334	optional behavior .....	3576	pipelines .....	3534
335	options .....	3467	PIPE_BUF.....	3511, 3574
336	OR lists.....	3536	popen().....	3562, 3565-3566
337	orphaned process group.....	3313, 3390	portability codes.....	3298
338	output device .....	3355	portable character set .....	3331
339	output file descriptor		portable filename character set .....	3314
340	duplication.....	3531	portable parameters.....	3522
341	output mode.....	3360	POSIX locale .....	3335
342	output processing.....	3359	POSIX.1 symbols .....	3375
343	overrun conditions.....	3492	POSIX.13.....	3411
344	overrun in dumping trace streams .....	3492	POSIX2_BC_BASE_MAX.....	3567
345	overrun in trace streams .....	3492	POSIX2_BC_DIM_MAX.....	3567
346	pack, rationale for omission.....	3550	POSIX2_BC_SCALE_MAX.....	3567
347	page .....	3313, 3407, 3411	POSIX2_BC_STRING_MAX .....	3567
348	PAGESIZE.....	3407, 3451, 3574	POSIX2_CHAR_TERM .....	3567
349	parallel I/O.....	3452	POSIX2_COLL_WEIGHTS_MAX.....	3567
350	parameter expansion.....	3526	POSIX2_C_BIND.....	3512, 3566
351	parameters.....	3359, 3522	POSIX2_C_DEV .....	3512, 3566
352	parameters, positional.....	3522	POSIX2_EXPR_NEST_MAX .....	3568
353	parameters, special.....	3522	POSIX2_FORT_DEV .....	3512, 3566

354	POSIX2_FORT_RUN.....	3512, 3566	process management.....	3556, 3559
355	POSIX2_LINE_MAX.....	3568	process scheduling.....	3417, 3560
356	POSIX2_LOCALEDEF.....	3512, 3564, 3567	prof, rationale for omission.....	3550
357	POSIX2_PBS.....	3567	profiling.....	3565
358	POSIX2_PBS_ACCOUNTING.....	3567	programming manipulation.....	3483
359	POSIX2_PBS_CHECKPOINT.....	3567	prompting.....	3524
360	POSIX2_PBS_LOCATE.....	3567	ps.....	3563-3564
361	POSIX2_PBS_MESSAGE.....	3567	pthread.....	3490
362	POSIX2_PBS_TRACK.....	3567	pthread_attr_getguardsize().....	3452
363	POSIX2_RE_DUP_MAX.....	3568	pthread_attr_setguardsize().....	3452
364	POSIX2_SW_DEV.....	3512, 3566	PTHREAD_BARRIER_SERIAL_THREAD.....	3443
365	POSIX2_SYMLINKS.....	3511	pthread_barrier_wait().....	3443
366	POSIX2_UPE.....	3512, 3566-3567	pthread_cond_init().....	3440
367	POSIX2_VERSION.....	3568	pthread_cond_timedwait().....	3382, 3427, 3449, 3569
368	POSIX_ALLOC_SIZE_MIN.....	3394	pthread_cond_wait().....	3382, 3399, 3449
369	posix_fadvise().....	3394	pthread_create().....	3440-3441
370	POSIX_FADV_DONTNEED.....	3394	PTHREAD_CREATE_DETACHED.....	3461
371	POSIX_FADV_NOREUSE.....	3394	PTHREAD_DESTRUCTOR_ITERATIONS.....	3575
372	POSIX_FADV_RANDOM.....	3394	pthread_detach().....	3461
373	POSIX_FADV_SEQUENTIAL.....	3394	pthread_getconcurrency().....	3451
374	POSIX_FADV_WILLNEED.....	3394	pthread_getcpuclockid().....	3430-3431
375	posix_madvise().....	3394	pthread_join().....	3382, 3461
376	POSIX_MADV_DONTNEED.....	3394	PTHREAD_KEYS_MAX.....	3575
377	POSIX_MADV_RANDOM.....	3394	pthread_key_create().....	3442
378	POSIX_MADV_SEQUENTIAL.....	3394	pthread_mutexattr_gettype().....	3449
379	POSIX_MADV_WILLNEED.....	3394	pthread_mutexattr_settype().....	3449
380	posix_mem_offset().....	3411-3412	PTHREAD_MUTEX_DEFAULT.....	3448
381	POSIX_REC_INCR_XFER_SIZE.....	3395	PTHREAD_MUTEX_ERRORCHECK.....	3448
382	POSIX_REC_MAX_XFER_SIZE.....	3395	pthread_mutex_init().....	3440
383	POSIX_REC_MIN_XFER_SIZE.....	3395	pthread_mutex_lock().....	3382, 3448, 3462
384	POSIX_REC_XFER_ALIGN.....	3394	PTHREAD_MUTEX_NORMAL.....	3448
385	posix_spawn().....	3495	PTHREAD_MUTEX_RECURSIVE.....	3448
386	posix_spawnp().....	3495	pthread_mutex_timedlock().....	3428
387	posix_trace_eventid_open().....	3487	pthread_mutex_trylock().....	3448
388	POSIX_TRACE_LOOP.....	3492	pthread_mutex_unlock().....	3448
389	posix_typed_mem_get_info().....	3411	PTHREAD_PROCESS_PRIVATE.....	3450
390	posix_typed_mem_open().....	3411	PTHREAD_PROCESS_SHARED.....	3450
391	POSIX_VERSION.....	3575	pthread_rwlockattr_destroy().....	3450
392	post-mortem filtering of trace event types.....	3489	pthread_rwlockattr_getpshared().....	3450
393	pr.....	3563-3564	pthread_rwlockattr_init().....	3450
394	pread().....	3452	pthread_rwlockattr_setpshared().....	3450
395	printf.....	3562-3563	pthread_rwlock_init().....	3450
396	printing.....	3558	PTHREAD_RWLOCK_INITIALIZER.....	3450
397	privilege.....	3323	pthread_rwlock_rdlock().....	3450
398	process group.....	3356	pthread_rwlock_t.....	3450
399	process group ID.....	3308-3309, 3356	pthread_rwlock_tryrdlock().....	3450
400	process group lifetime.....	3356	pthread_rwlock_trywrlock().....	3450
401	process group, orphaned.....	3313, 3390	pthread_rwlock_unlock().....	3451, 3464
402	process groups, concepts in job control.....	3308	pthread_rwlock_wrlck().....	3450
403	process ID reuse.....	3328	pthread_self().....	3442
404	process ID, rationale.....	3494	pthread_setconcurrency().....	3451

## Index

405	pthread_setprio()	3459	remove()	3316
406	pthread_setschedparam()	3459	rename	3560
407	pthread_setspecific()	3442	rename()	3316
408	pthread_spin_lock()	3444, 3463	renice	3563
409	pthread_spin_trylock()	3444	replenishment period	3420
410	PTHREAD_STACK_MIN	3575	reserved words	3521
411	PTHREAD_THREADS_MAX	3575	rewinddir	3560
412	putc	3560	RE_DUP_MAX	3511
413	putc()	3453	rm	3516, 3563
414	putchar	3560	rmdir	3560, 3563
415	pwd	3563	rmdir()	3316, 3381
416	pwrite()	3452	root directory	3314, 3328
417	queuing of waiting threads	3464	root file system	3314
418	quote removal	3529	root of a file system	3315
419	quoting	3518	routing	3466
420	rand()	3462	rsh, rationale for omission	3550
421	RCS, rationale for omission	3550	RTSIG_MAX	3575
422	RE		samefile()	3493
423	grammar	3354	SA_NOCLDSTOP	3309
424	RE bracket expression	3350	SA_SIGINFO	3387-3388
425	read	3560, 3562	scheduling allocation domain	3457
426	read lock	3450	scheduling contention scope	3457-3458
427	read()	3308, 3357-3358, 3380	scheduling documentation	3458
428	.....	3389-3390, 3392, 3401-3403, 3407-3408	scheduling policy	3329
429	.....	3452, 3462, 3494	SCHED_FIFO	3418-3419, 3457, 3464, 3561
430	read-write attributes	3449	SCHED_OTHER	3418
431	read-write locks	3449	SCHED_RR	3418, 3457, 3464, 3561
432	readdir	3560	SCHED_SPORADIC	3561
433	reading an active trace stream	3492	sdb, rationale for omission	3550
434	reading data	3358	sdiff, rationale for omission	3550
435	readlink	3560	seconds since the Epoch	3329
436	readlink()	3316	security considerations	3303, 3306
437	realpath	3560	.....	3311, 3322-3323, 3357
438	realtime	3394	security, monolithic privileges	3303
439	realtime signal delivery	3386	sed	3563-3564
440	realtime signal generation	3386	sem*()	3393
441	realtime signals	3399	semaphore	3330
442	red, rationale for omission	3550	semaphores	3397, 3561
443	redirect input	3530	semctl()	3393
444	redirect output	3530	semget()	3393
445	redirection	3529	semop()	3393
446	regcomp()	3565	sem_init()	3397
447	regerror()	3565	SEM_NSEMS_MAX	3575
448	regexec()	3565	sem_open()	3397
449	regfree()	3565	sem_timedwait()	3428
450	regular expression	3347	sem_trywait()	3382, 3399
451	definitions	3348	SEM_VALUE_MAX	3575
452	general requirements	3349	sem_wait()	3382, 3399
453	grammar	3353	sequential lists	3535
454	regular file	3314	session	3309, 3313, 3357
455	rejected utilities	3548	set	3523

456	setgid()	3315	siglongjmp()	3380, 3391, 3562
457	setgrent()	3322	signal	3315
458	setjmp()	3562	signal acceptance	3384
459	setlocale()	3561	signal actions	3389
460	setlogmask()	3564	signal concepts	3383
461	setpgid()	3308-3310, 3356	signal delivery	3384
462	setpriority()	3418	signal generation	3384
463	setpwent()	3322	signal names	3383
464	setrlimit()	3516	signal()	3383, 3385
465	setsid()	3356	signals	3467, 3540
466	setuid()	3315	SIGPIPE	3320, 3381
467	sh	3564, 3571	sigprocmask()	3385
468	shall	3296	SIGRTMAX	3387-3388
469	shall, rationale	3296	SIGRTMIN	3387-3388
470	shar, rationale for omission	3550	SIGSEGV	3320, 3383, 3388
471	shared memory	3409	sigsetjmp()	3562
472	shell	3308-3311	sigset_t	3383
473	SHELL	3347	SIGSTOP	3390
474	shell	3356-3357, 3383, 3389-3390	sigsuspend()	3389, 3392
475	shell commands	3532	SIGSYS	3383
476	shell errors	3531	SIGTERM	3383
477	shell execution environment	3522, 3540	sigtimedwait()	3382, 3402, 3427
478	shell grammar	3539	SIGTRAP	3383
479	lexical conventions	3540	SIGTSTP	3310, 3390
480	rules	3540	SIGTTIN	3310, 3357, 3390
481	shell variables	3523	SIGTTOU	3309, 3357, 3390
482	shell, job control	3308, 3383, 3390	SIGUSR1	3383
483	shl, rationale for omission	3550	SIGUSR2	3383
484	shm*()	3393	sigwait()	3382, 3462
485	shmctl()	3393	sigwaitinfo()	3382, 3402
486	shmdt()	3393	sigwait_multiple()	3385
487	shm_open()	3408-3411	SIG_DFL	3385-3386, 3389
488	shm_unlink()	3409-3411	SIG_IGN	3309, 3385-3386, 3389, 3391
489	should	3296	simple commands	3532
490	should, rationale	3296	single-quotes	3518
491	SIGABRT	3320, 3383	size, rationale for omission	3550
492	sigaction()	3385, 3387	SI_USER	3387-3388
493	SIGBUS	3320, 3383	sleep	3559, 3562
494	SIGCHLD	3309, 3385, 3389	sleep()	3391-3392, 3561
495	SIGCLD	3389	socket I/O mode	3466
496	SIGCONT	3309, 3386, 3389-3390	socket out-of-band data state	3467
497	SIGEMT	3383	socket owner	3467
498	SIGEV_NONE	3386	socket queue limit	3467
499	SIGEV_NOTIFY	3386	socket receive queue	3467
500	SIGEV_SIGNAL	3386	socket types	3466
501	SIGFPE	3320, 3383, 3385, 3388	sockets	3466
502	SIGHUP	3390	Internet Protocols	3467
503	SIGILL	3320, 3383	IPv4	3467
504	SIGINT	3310, 3455	IPv6	3467
505	SIGIOT	3383	local UNIX connection	3467
506	SIGKILL	3383, 3386, 3390	software development	3558, 3565

## Index

507	sort	3563-3564	system reboot	3321
508	spawn example	3495	System V	3305, 3311, 3384-3385, 3389
509	spawn()	3495, 3559	system()	3562, 3565-3566
510	special built-in	3537	tabs	3563
511	special built-in utilities	3542	tail	3563
512	special characters	3359	talk	3562, 3564
513	special control character	3360	tar, rationale for omission	3551
514	special parameters	3522	tcgetattr()	3309
515	specific implementation	3307	tcgetpgrp()	3310, 3356
516	spell, rationale for omission	3551	tcsetattr()	3309, 3355
517	spin locks	3444-3445	tcsetpgrp()	3309-3310
518	split	3563	tee	3562
519	sporadic server scheduling policy	3420	TERM	3346
520	SSIZE_MAX	3494, 3575	terminal access control	3357
521	SS_REPL_MAX	3423	terminal device file	3356
522	standard I/O streams	3392	closing	3359
523	stat	3516, 3559-3560	terminal type	3355
524	stat()	3305, 3408, 3516	termios structure	3359
525	state-dependent character encoding	3332	test	3562-3563
526	statvfs()	3516	TeX	3563
527	STREAMS	3392	text file	3321
528	STREAM_MAX	3575	thread	3321
529	strings	3565	thread cancelability states	3462
530	strip	3565	thread cancelability type	3462
531	structures, additions to	3376	thread cancelation	3460, 3462
532	stty	3346, 3564	thread cancelation points	3462
533	su, rationale for omission	3551	thread concurrency level	3451
534	subshells	3310	thread creation attributes	3439
535	successfully completed	3321	thread ID	3322, 3455
536	sum	3516	thread interactions	3466
537	sum, rationale for omission	3551	thread mutex	3456
538	superuser	3303, 3315, 3323, 3524, 3548	thread read-write lock	3464
539	supplementary group ID	3315	thread scheduling	3456
540	supplementary groups	3323	thread stack guard size	3451
541	symbolic constant	3298	thread-safe	3322
542	symbolic link	3316	thread-safe function	3322
543	symbolic name	3298	thread-safety	3330, 3452
544	symbols	3375	thread-safety, rationale	3330
545	SYMLOOP_MAX	3380	thread-specific data	3441
546	synchronized I/O	3402, 3560	threads	3439
547	data integrity completion	3321, 3402	implementation models	3441
548	file integrity completion	3321, 3402	tilde expansion	3525
549	synchronously-generated signal	3320	time	3562, 3565
550	sysconf()	3307, 3405, 3407	time()	3380
551		3409, 3455, 3509, 3559-3560	timeouts	3432
552	syslog()	3564	timers	3423
553	system call	3320	TIMER_ABSTIME	3424-3425
554	system documentation	3296	TIMER_MAX	3575
555	system environment	3558, 3564	timer_settime()	3424-3425
556	System III	3314, 3493	times()	3380, 3431, 3559
557	system interfaces	3495	timestamp clock	3491

558	time_t	3329	user database	3322
559	token recognition	3520	user database access	3322
560	TOSTOP	3309	user requirements	3555
561	touch	3516, 3563	usleep	3559
562	tput	3564	utility	3330
563	tr	3563-3564	utility argument syntax	3361
564	trace analyzer	3481	utility conventions	3361
565	trace event type-filtering	3489	utility syntax guidelines	3362
566	trace event types	3489	utime	3560
567	trace examples	3478	utime()	3316
568	trace model	3473	uucp	3564
569	trace operation control	3478	uudecode	3563-3564
570	trace storage	3477	uuencode	3563-3564
571	trace stream attribute	3483	variable assignment	3330
572	trace stream states	3476	variables	3522
573	tracing	3330, 3468	VEOF	3360
574	tracing all processes	3477	VEOL	3360
575	tracing, detailed objectives	3469	Version 7	3328, 3518
576	triggering	3491	vhangup()	3311
577	troff	3563	vi	3562, 3564
578	trojan horse	3303	virtual processor	3322
579	true	3562	VMIN	3360
580	tsort, rationale for omission	3551	VTIME	3360
581	tty	3564	wait	3562
582	ttyname()	3453	wait()	3380, 3383, 3389-3390, 3392, 3559
583	TTY_NAME_MAX	3575	waitpid()	3309, 3313, 3390, 3494, 3559
584	typed memory	3411	wall, rationale for omission	3551
585	TZNAME_MAX	3575	wc	3563
586	tzset()	3561	WERASE	3358
587	ualarm	3559	while loop	3537
588	UID_MAX	3494	who	3564
589	uid_t	3322	wide-character codes	3332
590	ULONG_MAX	3509	word expansions	3524
591	umask	3564	wordexp()	3565
592	umask()	3559	wordfree()	3565
593	umount()	3312	write	3560, 3562, 3564
594	unalias	3562	write lock	3450
595	uname	3564	write()	3308-3309, 3357, 3380
596	uname()	3559	.....3389-3390, 3392, 3401-3403, 3407-3408, 3494	
597	unbounded priority inversion	3459	writing data	3359
598	undefined	3297	WUNTRACED	3309
599	undefined, rationale	3297	xargs	3562
600	unexpand	3563	XSI	3323
601	uniq	3563-3564	XSI IPC	3393
602	unlink	3560	XSI supported functions	3446
603	unlink()	3316, 3381, 3408-3409, 3411	XSI threads extensions	3447
604	unpack, rationale for omission	3551	yacc	3565-3566
605	unsafe functions	3391		
606	unspecified	3297		
607	unspecified, rationale	3297		
608	until loop	3537		